PROGRAMMI

IN

LINGUAGGIO C

Corso di **Programmazione Modulo B**
e Laboratorio
Prof. GIULIANO LACCETTI

Studentessa
Bellino Virginia
Matr. 408466
Corso di Diploma in Informatica

http://www.dicecca.net

Indice

1. Algoritmi di ordinamento
- Bubble Sort - iterativo
- Insertion Sort - iterativo
- Selection Sort - iterativo
- Shell Sort

2. Algebra lineare numerica
- Gaxpy
- Prodotto tra due matrici
- Prodotto tra due vettori
- Saxpy

3. Algoritmi di ricerca
- Ricerca binaria
- Ricerca calcolata hash
- Ricerca sequenziale

4. Algoritmi floating-point
- Epsilon macchina
- Minimo rappresentabile

5. Algoritmi di complessità computazionale
- Valutazione di un polinomio in un punto (algoritmo di Horner)

6. Ricorsione
- Binary search
- Bubble sort
- Insertion sort
- Merge
- Quick sort
- Selection sort

7. Gestione di stringhe
- Manipolazione di stringhe

ALGORITMI DI ORDINAMENTO

BUBBLE SORT (ordinamento a bolle)

Documentazione esterna

- **Scopo:** BUBBLE SORT ordina in senso solitamente crescente un insieme disordinato di n elementi utilizzando un metodo basato su scambi.

- **Specifiche:**

  void BUBBLE_SORT(int n, a[ ])
  int i, k, t, d;

- **Descrizione:** l'approccio utilizzato da questo algoritmo consiste nel confrontare coppie di elementi contigui nella parte di array ancora disordinata: se risulta che essi non si trovano in ordine crescente, vengono scambiati di posto. In tal modo l'elemento massimo migra progressivamente verso l'estrema destra.
  Per implementare l'algoritmo vengono utilizzati 3 cicli innestati: WHILE, FOR e IF.

- **Lista dei parametri:**
  a[ ]: array che rappresenta l'insieme di elementi da ordinare. In input è disordinato. In output viene visualizzato ordinato.

  n: intero. E' la dimensione reale dell'array.

  i, j: interi. Rappresentano indici.

  t: intera. Variabile temporanea.

  D: indicatore logico che segnala quando l'array risulta ordinato.

- **Raccomandazioni d'uso:** siccome questo algoritmo si basa in modo piuttosto marcato sugli scambi, non risulta molto efficiente quando si tratta di ordinare insiemi con una grossa percentuale di elementi disordinati. E' pertanto consigliabile usarlo solo quando l'insieme da ordinare presenta una percentuale minima di elementi disordinati.

- **Complessità computazionale:** la complessità di tempo di questo algoritmo dipende dal valore dei dati forniti, i quali influenzano il numero di confronti e scambi da effettuare. Infatti, nel caso migliore(elementi già ordinati) si effettueranno solo n-1 confronti e nessuno scambio. Se invece l'array è completamente disordinato, allora bisognerà effettuare n-1 passaggi ed un numero di confronti e scambi pari a n(n-1)/2.

♦ **Esempio d'uso:**

esempio di programma chiamante

```
#include <stdio.h>
#include <stdlib.h>

/*prototipo della funzione BUBBLE SORT*/
        void BUBBLE_SORT (int n, int a[ ] );

main ( )
{

/*dichiarazione di variabili */
        int n, i;
        int *a;

/* inserimento dimensione reale dell'array */
        printf ("\nQuanti elementi vuoi inserire?");
        scanf ("%d", &n);


/*allocazione dinamica */
a=(int*) calloc (n, sizeof (int));

        for (i=0; i<n; i++)
        {
                printf ("Elemento numero%d:",i+1);
                scanf ("%d",&a[i]);
        }

/*chiamata della function BUBBLE_SORT*/
        BUBBLE_SORT (n, a);

/*stampa dell'array ordinato */
printf ("\nL'array ordinato è:\n");
        for (i=0; i<n; i++)
        {
                printf ("%d\n", a[i]);
        }
printf ("\n Programma terminato\n");
}

/*Fine del programma */
```

esempio di esecuzione

```
Quanti elementi vuoi inserire? 7
elemento numero 1: 28
elemento numero 2:  -5
elemento numero 3:   8
elemento numero 4:   3
elemento numero 5:  15
elemento numero 6: -16
elemento numero 7:  42

L'array ordinato è:
-16  -5  3  8  15  28  42
**********************************************************************

Quanti elementi vuoi inserire? 9
elemento numero 1: 15
elemento numero 2: 11
elemento numero 3:   9
elemento numero 4:   8
elemento numero 5:   6
elemento numero 6:   1
elemento numero 7:   7
elemento numero 8:   4
elemento numero 9:   5

L'array ordinato è:
1 4 5 6 7 8 9 11 15
```

FUNCTION BUBBLE SORT

In P-like

```
Procedure BUBBLE_SORT (in: n, inout: a[ ])
        var i, k, t: integer
        var D: logical
begin
        i:=n-1
repeat
        D:=false
for k=1 to i do
        if(a(k)>a(k+1))then
        t:=a(k)
        a(k):=a(k+1)
        a(k+1):=t
        D:=true
        endif
endfor
        i:=i-1
until (i=0 or (not D) )
end
```

In C:

```
void BUBBLE_SORT (int n, a[ ])
{
        int i, k, t, d; /*dichiarazione di variabili*/
        i=n-1;
                /*corpo della function*/
        while ( (i!=0) || (d=0) )
        {
                d=0;
                        for (k=0; k<i; k++)
                        {
                                if (a[k]>a[k+1])
                                {
                                        t=a[k];
                                        a[k]=a[k+1];
                                        a[k+1]=t;
                                        d=1;
                                }
                        }
                i=i-1;
        }
}
```

INSERTION SORT (ordinamento per inserzione)

Documentazione esterna

- **Scopo:** il presente programma si propone di ordinare in modo non decrescente un insieme di n elementi disposti in modo casuale utilizzando il metodo di inserzione.

- **Specifiche:**

  function INSERTION_SORT (int n, B[ ])
  int primo, p, i, j, x;

- **Descrizione:** l'approccio seguito da questo algoritmo ricorda molto il metodo usato dai giocatori per ordinare le carte.
  L'idea infatti è quella di costruire la soluzione completa prendendo di volta in volta un elemento dalla parte dell'insieme non ancora ordinata, e inserirlo nella parte ordinata che aumenta così di un elemento. Successivamente, l'elemento che in precedenza occupava la posizione di minimo viene spinto verso l'estrema destra, così da avere alla fine un insieme completamente ordinato.

- **Lista dei parametri:**
  primo: intero. Indica l'elemento minimo dell'array.
  p: intero. Posizione del minimo nell'array.
  i: intero. Numero di elementi ordinati ad ogni passo.
  x: elemento che deve essere inserito al passo corrente.
  n: intero. E' la dimensione effettiva dell'array.
  B[ ]: insieme di elementi da ordinare.

- **Raccomandazioni d'uso:** il programma è consigliato soltanto per ordinare piccoli insiemi di dati.

- **Complessità computazionale:** le operazioni dominanti dell'algoritmo sono confronti e inserimenti. Nel caso peggiore, quando cioè bisogna riordinare successioni ordinate in senso opposto, l'andamento riscontrato della complessità asintotica è O(n^2) [quadratico]. Invece, nel caso migliore, cioè quando l'insieme è già ordinato, la complessità risulta O(n) [lineare].

- **Esempio d'uso**

```
esempio di programma chiamante
#include <stdio.h>
#include <stdlib.h>
/*prototipo della funzione INSERTION_SORT*/
        void INSERTION_SORT (int n, int B[ ]);
```

```
main( )
{
/*dichiarazione di variabili */
        int n, i, *B;

/*dimensione dell'array */
        printf ("\nQuanti elementi vuoi inserire?");
        scanf ("%d", &n);

/*allocazione dinamica e inserimento dati */
B= (int*) malloc ( sizeof (int)*n);
                for (i=0; i<n; i++)
                {
                        printf ("Inserisci elemento numero %d:", i);
                        scanf ("%d", &B[i]);
                }

/*chiamata della function INSERTION SORT*/
        INSERTION_SORT (n, &B[0]);

/*stampa risultato */
printf ("\nL'array ordinato è: \n");
        for (i=0; i<n; i++)
        {
                printf ("%d\n", B[i]);
        }
}
```

esempio di esecuzione

```
Quanti elementi vuoi inserire? 6
Inserisci elemento numero 0: 9
Inserisci elemento numero 1:1
Inserisci elemento numero 2:5
Inserisci elemento numero 3:11
Inserisci elemento numero 4:26
Inserisci elemento numero 5:8
L'array ordinato è:
1
5
8
9
11
26
```

FUNCTION INSERTION SORT

In C:

```
void INSERTION_SORT(int n, int B [ ] )
{

/*dichiarazione variabili locali*/
        int primo, p, i, j, x;
        primo = B[0];
        p=0;
        /* ricerca del minimo: */
        for(i=1;i<n;i++)
        {
                if ( B[i]<primo)
                {
                        primo =B[i];
                        p= i;
                }
        }
        /* effettuazione scambio e spostamento del minimo in prima posizione: */
        B[p]=B[0];
        B[0]=primo;
        /* preleva il prossimo elemento da inserire: */
        for(i=2;i<n;i++)
        {
                x =B[i];
                j =i;
/*ricerca della posizione in cui inserire l'elemento corrente e avanzamento di quelli
maggiori */
                while (x<B[j-1])
                {
                        B[j] = B[j-1];
                        j=j-1;
                }
                /* inserimento elemento corrente */
                B[j]=x;
        }
}
```

SELECTION SORT (ordinamento per selezione)

Documentazione esterna

- **Scopo:** SELECTION SORT ordina (generalmente in modo crescente) un insieme di n elementi disposti casualmente utilizzando il metodo di selezione.

- **Specifiche:**

  function SELECTION SORT (int n, A[ ])
  int min, i, j;

- **Descrizione:** l'algoritmo, dopo aver definito l'array, lo considera suddiviso in due parti; una parte disordinata ed una disordinata. Utilizzando due cicli for innestati viene ricercato l'elemento minimo e spostato sempre verso sinistra, in modo da ottenere alla fine un array completamente ordinato.

- **Lista dei parametri:**

  n: intero. Indica il numero di elementi presenti nell'array.
  A[ ]: array di un generico tipo che rappresenta l'insieme dei dati da ordinare. In input è disordinato; in output viene visualizzato ordinato.
  min: elemento minimo dell'array.
  i, j: indici.

- **Raccomandazioni d'uso:** questo algoritmo è consigliato solo per l'ordinamento di piccoli insiemi. Infatti, per insiemi di grandi dimensioni esistono metodi molto più efficienti.

- **Complessità computazionale:** le operazioni dominanti dell'algoritmo sono CONFRONTI e SCAMBI. Gli scambi eseguiti sono n-1, perché pari al numero di volte che viene eseguito il ciclo for più esterno. I confronti sono invece pari a n(n-1)/2.

- **Esempio d'uso:**

```
esempio di programma chiamante
#include <stdio.h>
#include <stdlib.h>

/*prototipo della funzione SELECTION SORT*/
        void SELECTION_SORT (int n, int A[ ] );

main ( )
{
```

```
/*dichiarazione di variabili*/
        int n, i, *A;

/*inserimento dimensione dell'array*/
        printf ("\nQuanti elementi vuoi inserire?");
        scanf ("%d",&n);


/*allocazione dinamica */
A=(int*)calloc (n, sizeof (int));

        for(i=0; i<n; i++)  /*definizione array */
        {
                printf ("Inserisci elemento numero %d:", i);
                scanf ("%d", &A[i]);
        }

/*chiamata della function SELECTION SORT*/
        SELECTION_SORT (n, &A[0]);

/*stampa del risultato */
printf ("\nL'array ordinato è:\n");
        for (i=0; i<n; i++)
        {
                printf (%d\n",A[i]);
        }
}  /*fine del programma*/
```

esempio di esecuzione

```
Quanti elementi vuoi inserire? 8
Inserisci elemento numero 0: 29
Inserisci elemento numero 1: -7
Inserisci elemento numero 2: 18
Inserisci elemento numero 3:  3
Inserisci elemento numero 4: 16
Inserisci elemento numero 5:-19
Inserisci elemento numero 6: 41
Inserisci elemento numero 7:  0

L'array ordinato è:
-19  -7  0  3  16  18  29  41
```

FUNCTION SELECTION SORT

In P-like

```
Procedure Selection Sort (in: n; inout: A[])
var i, j, min: integer
        for i=1 to n-1 do
        min:=A(i)
                for j=2 to n do
                        if A(j)<min then
                        min:=A(j)
                        endif
                endfor
        A(j):=A(i)
        A(i):=min
        endfor
```

In C:

```
void SELECTION_SORT(int n, A[ ])
{
        /*dichiarazione di variabili */
        int min, i, j;

        /*corpo della function */
        for (i=0; i<n; i++)
        {
        min=A[i];
                for (j=i+1; j<n; j++)
                {
                        if (A[j]<min)
                        {
                        min=A[j];
                        A[j]=A[i];
                        A[i]=min;
                        }
                }
        }
}
```

SHELL SORT (ordinamento con il metodo di Shell)

Documentazione esterna

- **Scopo:** ordinare in modo crescente un insieme di n elementi disposti casualmente utilizzando il metodo di Shell di inserzione con diminuzione degli incrementi.

- **Specifiche:**

  void shell_sort (int *a, int n);
  int dist, j, k, x, corrente, prec ;

- **Descrizione:** utilizzando tre cicli while ed un ciclo for, la funzione suddivide l'insieme da ordinare in tante catene di due elementi poste a distanza dist; successivamente, ad ogni passo tali catene vengono ordinate utilizzando il metodo di inserzione.

- **Lista dei parametri:**

  ***a:** intero. Puntatore all'array degli elementi.
  n: intero. Dimensione dell'array.
  dist: intero. Distanza alla quale si trovano gli elementi della catena da ordinare.
  j, k: indici.
  x: intero. E' il 2° elemento della catena da ordinare che deve essere inserito.
  corrente: intero. Posizione finale di x nella catena.
  prec: intero. Indice dell'elemento che viene comparato con x.

- **Raccomandazioni d'uso:** adatto per ordinare ampi insiemi di dati, anche se esistono metodi dalle prestazioni migliori. Poiché esso si presenta più complicato degli altri sort, è adatto per dimostrare l'importanza di un progetto top down corretto e pulito.

- **Complessità computazionale:** le operazioni eseguite dalla funzione sono confronti ed inserimenti, e il loro numero dipende dal valore dei dati di input.

- **Esempio d'uso**

```
esempio di programma chiamante
#include <stdio.h>
#include <stdlib.h>

/*prototipo */
        void shell_sort (int *a, int n);

main( )
```

```
{
/*dichiarazione di variabili */
        int *a;
        int n, i;

        printf ("\nQuanti elementi vuoi inserire?  ");
        scanf ("%d", &n);

/*allocazione dinamica e inserimento dati */
a=(int *)calloc(n, sizeof(int));

 for (i=0;i<n;i++)
 {
        printf ("Inserire il %d elemento: ",i+1);
        scanf ("%d", &a[i]);
 }

 /* Richiamo della funzione di ordinamento con il metodo di Shell */
 shell_sort (a, n);

 /* Stampa risultato */
 printf ("Array ordinato con il metodo di Shell: \n");
 for (i=0;i<n;i++)
    printf ("%d \n", a[i]);
}
```

esempio di esecuzione

```
Quanti elementi vuoi inserire? 6
Inserire il 1 elemento: 71
Inserire il 2 elemento: 3
Inserire il 3 elemento: 28
Inserire il 4 elemento: 0
Inserire il 5 elemento: 5
Inserire il 6 elemento: 14
Array ordinato con il metodo di Shell:
0
3
5
14
28
71
```

FUNZIONE SHELL_SORT

In C:

```
void shell_sort (int *a, int n)
{
/*variabili locali */
        int dist, j, k, x, corrente, prec ;

        dist = n;

while (dist > 1)
{
                dist=dist/2;

        for (j=0;j<dist;j++)
        {
                k=j+dist;

                while (k < n)
                        {
                                x = a[k];
                                corrente = k;
                                prec = corrente - dist;

                                        while ((prec >= j)&&(x < a[prec]))
                                        {
                                                a[corrente]=a[prec];
                                                corrente = prec;
                                                prec = prec- dist;
                                        }

                                                a[corrente]=x;
                                                k = k + dist;

                        }


        }


}

}
```

ALGEBRA LINEARE NUMERICA

GAXPY

Documentazione esterna

- **Scopo:** il programma esegue un operazione dell'algebra lineare numerica, denominata gaxpy, che somma il prodotto tra una matrice ed un vettore ad un altro vettore.

- **Specifiche:**

  int *gaxpy(int **A, int *prod, int *sum, int righe, int colonne);

- **Descrizione:** la funzione esegue direttamente la somma tra il vettore risultante del prodotto mat x vet (ricevuto in input dal programma chiamante) ed un altro vettore, restituendo poi il risultato al programma chiamante.

- **Lista dei parametri:**

  ****A:** puntatore alla matrice.
  ***prod:** puntatore al vettore da moltiplicare.
  ***sum:** puntatore al vettore da sommare.
  righe: intero. Numero di righe della matrice e di elementi del vettore da sommare.
  colonne: intero. Colonne della matrice ed elementi del vettore da moltiplicare.

- **Complessità computazionale:** il programma prevede l'esecuzione di moltiplicazioni ed addizioni.

- **Esempio d'uso:**

esempio di programma chiamante

```
#include <stdio.h>
#include <stdlib.h>

/*prototipo della funzione gaxpy */
    int *gaxpy(int **A, int *prod, int *sum, int righe, int colonne);

main ( )
{

    /* variabili */
    int **A, *prod, *sum, *ris2;
    int i, j, righe, colonne;
```

```
        printf("Inserire il numero di righe della matrice ricordando che ");
        printf(" e' uguale al numero di elementi del vettore da sommare: \n ");
        scanf("%d", &righe);

        printf("\n Inserire il numero di colonne della matrice ricordando che ");
        printf(" e' uguale al numero di elementi del vettore da moltiplicare: \n ");
        scanf("%d", &colonne);

/*allocazione della matrice */
        A=(int**)calloc(righe, sizeof(int*));
                for(i=0; i<righe; i++)
                        A[i]=(int*)calloc(colonne, sizeof(int));

/*inserimento dati */
        printf("\n Inserire la matrice: \n");
        for(i=0; i<righe; i++)
        {
                for(j=0; j<colonne; j++)
                {
                        printf("Elemento numero(%d, %d):", i, j);
                        scanf("%d", &A[i][j]);
                }

        }

/*allocazione vettore da moltiplicare */
        prod = (int*)calloc(colonne, sizeof(int));

/*inserimento dati */
        printf("\n Inserire il vettore da moltiplicare: \n");
        for(j=0; j<colonne; j++)
        {
                printf("Elemento numero %d: ",j);
                scanf("%d", &prod[j]);
        }

/*allocazione vettore da sommare */
        sum = (int*)calloc(righe, sizeof(int));

/*inserimento dati */
        printf("\n Inserire il vettore da sommare: \n");
        for(i=0; i<righe; i++)
        {
                printf("Elemento numero %d: ",i);
```

```
                scanf("%d", &sum[i]);
        }

/*allocazione vettore risultato */
        ris2=(int*)calloc(righe, sizeof(int));

/*chiamata della funzione gaxpy */
        ris2=gaxpy(A, prod, sum, righe, colonne);

/*stampa risultato */
        printf("\n \n Risultato finale: \n");
        for(i=0; i<righe; i++)
        {
                printf ("%d ",ris2[i]);
        }

   printf("\n \n Fine Programma \n");

}
```

esempio di esecuzione

```
Inserire il numero di righe della matrice ricordando che
e' uguale al numero di elementi del vettore da sommare:
3

Inserire il numero di colonne della matrice ricordando che
e' uguale al numero di elementi del vettore da moltiplicare:
2

Inserire la matrice:
Elemento numero(0,0):4
Elemento numero(0,1):11
Elemento numero(0,2):7
Elemento numero(1,0):234
Elemento numero(1,1):78
Elemento numero(1,2):23

Inserire il vettore da moltiplicare:
Elemento numero 0: 3
Elemento numero 1: 4
Elemento numero 2: 9

Inserire il vettore da sommare:
Elemento numero 0: 7
Elemento numero 1: 2

Risultato finale:
151  146
Fine Programma
```

FUNZIONE GAXPY

In C:

```
int *gaxpy(int **A, int *prod, int *sum, int righe, int colonne )
{
        int *ris1,*ris2, i,j; /* Variabili locali */

/*allocazione vettori usati per le operazioni di calcolo */
        ris1=(int*)calloc(righe, sizeof(int));
        ris2=(int*)calloc(righe, sizeof(int));

/* Prodotto Mat x Vet */
        for (i=0; i<righe; i++)
        {
        ris1[i]=0;

                for (j=0; j<colonne; j++);
                {
                        ris1[i]=ris1[i]+ A[i][j]*prod[j];
                }

        }

   /* Somma dei due vettori */
        for (i=0; i<righe; i++)
        {
                ris2[i]=ris1[i]+ sum[i];
        }

   return ris2;
}
```

PRODOTTO TRA DUE MATRICI

Documentazione esterna

- **Scopo:** il presente programma si propone di effettuare il calcolo del prodotto tra due matrici.

- **Specifiche:**

  int **matxmat (int **A, int **B, int righe1,int colonne1, int colonne2);
  int i, j, k;
  int **C;

- **Descrizione:** la function, dopo aver allocato dinamicamente lo spazio necessario per memorizzare la matrice risultato, utilizza tre cicli for innestati per effettuare il calcolo del prodotto matriciale e poi restituisce al programma chiamante il valore della matrice risultato.

- **Lista dei parametri:**

  ****A:** intero. Puntatore alla prima matrice.
  ****B:** intero. Puntatore alla seconda matrice.
  i, j, k: interi. Rappresentano indici.
  ****C:** intero. Puntatore alla matrice risultato.
  righe1: intero. E' il numero di righe della prima matrice.
  colonne1: intero. E' il numero di colonne della prima matrice, ed equivale anche al numero di righe della seconda matrice.
  colonne2: intero. E' il numero di colonne della seconda matrice.

- **Complessità computazionale:** il programma prevede l'esecuzione di moltiplicazioni e addizioni in numero pari a **righe1*colonne1*colonne2.**

- **Esempio d'uso:**

```
esempio di programma chiamante
#include <stdio.h>
#include <stdlib.h>
/*prototipo */
        int **matxmat (int **A, int **B, int righe1,int colonne1, int colonne2);

main(  )
{/*variabili */
        int righe1, colonne1, colonne2, i, j;
        int **A,**B,**C;
        printf ("\nInserire le righe della prima matrice: \n");
```

```
        scanf("%d",&righe1);

        printf ("\nInserire le colonne della prima matrice ricordando che \n");
        printf ("corrispondono al numero di righe della seconda matrice. \n");
        scanf ("%d",&colonne1);

        printf ("\nInserire il numero di colonne della seconda matrice:\n");
        scanf ("%d",&colonne2);

/*allocazione delle due matrici da moltiplicare */
        A=(int**)malloc(righe1*sizeof(int*));
        for(i=0; i<righe1; i++)
                A[i]=(int*)malloc(colonne1*sizeof(int));

        B=(int**)malloc(colonne1*sizeof(int*));
        for(i=0; i<colonne1;i++)
                B[i]=(int*)malloc(colonne2*sizeof(int));

/*inserimento dei dati */
        for (i=0;i<righe1;i++)
        {
                for (j=0;j<colonne1;j++)
                {
                        printf("Elemento numero (%d,%d):",i,j);
                        scanf("%d", &A[i][j]);
                }
        }

        printf("\nInserimento dati seconda matrice\n");
        for (i=0;i<colonne1;i++)
        {
                for (j=0;j<colonne2;j++)
                {
                printf("Elemento numero (%d, %d):",i,j);
                scanf("%d", &B[i][j]);
                }
        }

/*allocazione della matrice risultato */
        C=(int**)malloc(righe1*sizeof(int*));
        for (i=0;i<righe1; i++)
                C[i]=(int*)malloc(colonne2*sizeof(int));

/*chiamata della funzione matxmat */
```

```
        C = matxmat (A,B,righe1,colonne1,colonne2);
/*stampa risultato */
        printf ("\n La matrice risultante avrà i seguenti elementi: \n");
        for (i=0; i<righe1; i++)
        {
                for(j=0; j<colonne2; j++)
                {
                printf("\n elemento in posizione %d %d :", C [i] [j] ) ;
                }
        }

}
```

esempio di esecuzione

```
Inserire il numero di righe della I matrice;
3
Inserire il numero di colonne della I matrice, ricordando che
corrisponde al numero di righe della II matrice;
2
Inserire il numero di colonne della II matrice;
4

Inserimento dati I matrice:
Elemento numero (0, 0) 1
Elemento numero (0, 1) 2
Elemento numero (1, 0) 3
Elemento numero (1, 1) 4
Elemento numero (2, 0) 7
Elemento numero(2, 1) 8

Dati II matrice:
Elemento numero (0, 0) 2
Elemento numero (0, 1) 5
Elemento numero (0, 2) 3
Elemento numero (0, 3) 4
Elemento numero (1, 0) 1
Elemento numero (1, 1) 9
Elemento numero (1, 2) 12
Elemento numero (1, 3) 3

La matrice risultante sarà composta dai seguenti elementi:
elemento in posizione (0, 0): 4
elemento in posizione (0, 1): 23
elemento in posizione (0, 2): 27
elemento in posizione (0, 3): 10
elemento in posizione (1, 0): 10
elemento in posizione (1, 1): 51
elemento in posizione (1, 2): 57
elemento in posizione (1, 3): 24
elemento in posizione (2, 0): 22
elemento in posizione (2, 1): 107
elemento in posizione (2, 2): 117
elemento in posizione (2, 3): 52
```

FUNZIONE MATXMAT

In C:

```
int **matxmat(int **A, int **B, int righe1,int colonne1, int colonne2)
{

/*variabili locali */
        int i, j, k;
        int **C;

C=(int**)malloc(righe1*sizeof(int*));
        for(i=0; i<righe1; i++)
        C[i]=(int*)malloc(colonne2*sizeof(int));

/*prodotto */
for(i=0; i<righe1; i++)
{
        for (j=0; j<colonne2; j++)
        {
        C[i][j]=0;
                for(k=0;k<colonne1;k++)
                {
                C [i] [j] = C [i] [j] + A [i] [k] * B [k] [j] ;
                }
        }

}

        return C;
}
```

PRODOTTO SCALARE TRA DUE VETTORI

Documentazione esterna

- **Scopo:** il programma effettua il prodotto scalare tra due vettori.

- **Specifiche:**

  int prodotto_scalare (int n, int *a, int *b)
  int i, prod;

- **Descrizione:** utilizzando un ciclo for, la funzione esegue il prodotto tra i due vettori, e poi restituisce il valore ottenuto al programma chiamante.

- **Lista dei parametri:**

  n: intero. Dimensione dei due vettori.
  ***a**: intero. Puntatore al primo vettore.
  ***b:** intero. Puntatore al secondo vettore.
  i: intero. Indice.
  prod: intero. Variabile temporanea in cui viene memorizzato il valore da restituire al programma chiamante.

- **Complessità computazionale:** il programma prevede l'esecuzione di *n* moltiplicazioni ed *n-1* addizioni.

- **Esempio d'uso:**

esempio di programma chiamante

```
#include <stdio.h>
#include <stdlib.h>

/*prototipo */
        int prodotto_scalare (int n, int *a, int *b);

main ( )
{
/*dichiarazione di variabili */
        int i, n, C;
        int *A, *B;

printf ("Inserire la dimensione dei vettori: \n");
scanf ("%d", &n);
```

```
/*Allocazione dinamica dei vettori A e B */
        A=(int*) calloc (n, sizeof(int));
        B=(int*) calloc (n, sizeof(int));

/*Inserimento elementi */
printf ("\nInserire elementi vettore A:");
        for(i=0; i<n; i++)
        {
                scanf ("%d", &A[i]);
        }


printf ("\nInserire elementi vettore B:");
        for(i=0; i<n; i++)
        {
                scanf ("%d", &B[i]);
        }


/*chiamata della funzione prodotto scalare */
        C = prodotto_scalare (n, A, B);

/*stampa risultato */
printf ("\nIl risultato del prodotto scalare è %d", C);

return;
}
```

esempio di esecuzione

Inserire la dimensione dei vettori: 5
Inserire elementi vettore A: 2 4 6 8 10
Inserire elementi vettore B: 3 4 7 9 11

Il risultato del prodotto scalare è 246

****

Inserire la dimensione dei vettori: 9
Inserire elementi vettore A: 3 5 7 9 11 2 3 1 0
Inserire elementi vettore B: 0 4 5 7 9 1 2 8 11
Il risultato del prodotto scalare è 233

FUNZIONE PRODOTTO SCALARE

In C:

```
int prodotto_scalare (int n, int *a, int *b)
{
/*variabili locali */
        int i, prod;

        prod=0;

                for (i=0; i<n; i++)
                prod = prod + a[i]*b[i];

        return prod;
}
```

In P-like:

```
function prodotto scalare(n, a, b)
begin
var: i, prod: integer

        prod:=o

        for i= 1 to n do
        prod := prod +a(i)*b(i)

        endfor

end
```

SAXPY

Documentazione esterna

- **Scopo:** il programma esegue un operazione dell'algebra lineare numerica, denominata saxpy, che somma il prodotto tra uno scalare ed un vettore ad un altro vettore.

- **Specifiche:**

  void SAXPY (int i, int n, int b, int x[ ], int y[ ])

- **Descrizione:** utilizzando un ciclo for, la funzione esegue il prodotto tra lo scalare ed il primo vettore, e poi somma il risultato al secondo vettore.

- **Lista dei parametri:**

  b: intero. Rappresenta lo scalare considerato.
  x[ ], y[ ]:interi. Sono i due vettori per il prodotto e la somma.
  n: intero. E' il numero di elementi dei due vettori.
  i: intero. E' un indice.

- **Complessità computazionale:** il programma prevede l'esecuzione di n moltiplicazioni ed n addizioni.

- **Esempio d'uso:**

```
esempio di programma chiamante
#include <stdio.h>
#include <stdlib.h>

/*prototipo della funzione SAXPY */
        void SAXPY (int i, int n, int b,  int x[ ], int y[ ]);

main( )
{
/*dichiarazione di variabili */
        int i, n, b, *x, *y;

/*inserimento dati */
        printf("Inserire il numero da moltiplicare al vettore :");
        scanf("%d", &b);

        printf("Inserire la dimensione dei due vettori:");
        scanf("%d", &n);
```

```
/*allocazione di memoria */
        x = (int*) malloc (n* sizeof(int));
        y = (int*) malloc (n* sizeof(int));

        printf("Inserisci gli elementi del primo vettore \n");
        for(i=0;i<n;++i)
        {
                scanf("%d", &x[i]);
        }


        printf("Inserisci gli elementi del vettore da addizionare \n ");
        for(i=0;i<n;++i)
        {
                scanf("%d", &y[i]);
        }

/*chiamata della funzione SAXPY */
        SAXPY (i, n, b, x, y);

/*stampa risultato */
        printf("La somma del prodotto scalare ad un vettore è \n ");
        for(i=0;i<n;++i)
        {
                printf("%d ",y[i]);
        }
/*deallocazione dello spazio occupato in memoria */
        free(x);
        free(y);

}
```

esempio di esecuzione

Inserire il numero da moltiplicare al vettore : 4

Inserire la dimensione dei due vettori: 3

Inserisci gli elementi del primo vettore
5 9 10

Inserisci gli elementi del vettore da addizionare
3 12 6

La somma del prodotto scalare ad un vettore è:
23 48 46

FUNZIONE SAXPY

In C:

```
void SAXPY (int i, int n, int b, int x[], int y[])
{

        /*corpo della funzione */
        for(i=0;i<n;i++)
        {
                y[i]=(x[i]*b)+y[i];
        }

}
```

In P-like:

```
procedure saxpy (in: n, b, x[ ], y[ ]; out b[ ] )
begin
for i =1 to n do

        y(i):=( x(i)*b )+y(i)

end
```

ALGORITMI DI RICERCA

BINARY SEARCH (ricerca binaria)

Documentazione esterna

- **Scopo:** dato un elemento x ed un insieme di dati ordinati in senso crescente, stabilire se x appartiene all'insieme.

- **Specifiche:**

  logical RICERCA_BINARIA (int *vett, int n, int k)
  int m, inizio, fine;
  logical trovato;

- **Descrizione:** dopo aver definito l'array ordinato e l'elemento da cercare, i due estremi assumono il ruolo di primo ed ultimo. Successivamente, finché il primo è maggiore dell'ultimo, si calcola l'elemento medio e lo si confronta con x, fino a che quest'ultimo non viene trovato.

- **Lista dei parametri:**

  m: punto medio dell'array.
  inizio, fine: estremi della porzione di array considerata.
  trovato: variabile logica che indica quando l'elemento è stato trovato
  n: dimensione effettiva dell'array ordinato
  ***vett:** variabile puntatore all'array ordinato

- **Complessità computazionale:** l'operazione dominante eseguita dalla function è il confronto. Nel caso migliore (x trovato immediatamente), viene eseguito un solo confronto. Nel caso peggiore la complessità computazionale equivale a log2 (n).

- **Esempio d'uso**

esempio di programma chiamante

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

/*Dichiarazione di un nuovo tipo */
        typedef enum {false, true} logical;

/*Prototipo della funzione RICERCA BINARIA */
        logical RICERCA_BINARIA (int *a, int n, int k);

main ( )
```

```
{
/*dichiarazione di variabili */
        int *a, n, chiave, i;

/* Inserimento dell'array ordinato */
        printf ("\nQuanti elementi vuoi inserire?");
        scanf ("%d", &n);

        a=(int *) calloc (n, sizeof(int));

for(i=0;i<n;i++)
        {
                printf ("Elemento numero %d:",i+1);
                scanf ("%d", &a[i]);
        }

printf ("Inserire l'elemento da ricercare ");
scanf ("%d", &chiave);

/* Visualizzazione del risultato richiamando la function RIC. BIN. */
        if (RICERCA_BINARIA (a, n, chiave))

                printf ("Elemento trovato");

        else

                printf ("Elemento non trovato");

}
```

esempio di esecuzione

```
Quanti elementi vuoi inserire? 10
elemento numero 1:  5
elemento numero 2:  6
elemento numero 3:  7
elemento numero 4:  8
elemento numero 5: 10
elemento numero 6: 11
elemento numero 7: 23
elemento numero 8: 34
elemento numero 9: 45
elemento numero 10: 51
Inserire l'elemento da ricercare  11
Elemento trovato
```

FUNZIONE RICERCA BINARIA

In C:

```
logical RICERCA_BINARIA (int *a, int n, int k)
{
/*dichiarazione di variabili locali */
        int m, inizio, fine;
        logical trovato;

/* Ricerca dell'elemento scelto */
        inizio=0;
        n--;
        fine =n;
        trovato =false;
        do
        {

                m=(inizio + fine)/2;
                if ( a[m]==k)
                trovato = true;

        else

                if (k>a[m])
                inizio=m+1;

        else

                if (k<[m])
                fine=m-1;

        }while ((!trovato)&&(inizio <= fine));

        return trovato;

}
```

RICERCA CALCOLATA HASH

Documentazione esterna

- **Scopo:** ricercare un elemento in un insieme utilizzando il metodo della ricerca calcolata hash.

- **Specifiche:**

  logical hash_search(int *table, int tablesize, int key)
  logical active, found;
  int start, position, temp;

- **Descrizione:** dopo aver definito la tabella calcolata da esaminare, la chiave cercata, la dimensione della tabella e il modo per segnalare una casella vuota, l'algoritmo (suddiviso in due funzioni) esegue la ricerca fino a che la chiave viene trovata, oppure finché incontra una casella vuota o si accorge che la tabella è piena ma la chiave non c'è.
  Tali condizioni vengono rilevate con l'utilizzo di due variabili logiche: la prima (active) segnala quando la ricerca deve terminare; la seconda (found) indica se l'elemento è stato trovato.

- **Raccomandazioni d'uso:** il programma è altamente consigliato per il recupero veloce di dati da tabelle grandi e piccole.

- **Lista dei parametri:**

  ***table:** intero. Puntatore all'array degli elementi costruito con il metodo hash.
  key: intero. Elemento da ricercare.
  active: variabile logica che indica quando arrestare la ricerca.
  found: variabile logica che indica quando l'elemento è stato trovato. Restituisce il valore al programma chiamante.
  start: intero. Modulo dell'elemento da ricercare.
  position: intero. Indice temporaneo.
  temp: intero. Variabile temporanea.

- **Funzioni ausiliarie:**

  void hash (int *array, int *table, int dim, int tablesize)
  int *vett_app, n, j, k, i;

- **Complessità computazionale:** la complessità di tempo di questo programma è molto bassa. Infatti, il numero di accessi alla tabella può arrivare ad un massimo

di tre se la chiave è presente, oppure può arrivare ad un massimo di 13 se la chiave non è presente. Questi due valori sono indipendenti dalla dimensione dell'array.

- **Esempio d'uso:**

esempio di programma chiamante

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

/*dichiarazione nuovo tipo */
typedef enum {falso, vero} logical;

/*prototipi */
logical hash_search(int *table, int tablesize, int key);
void hash (int *array, int *table, int dim, int tablesize);

main( )
{
/*dichiarazione di variabili */
int *array, *table, i, tablesize, dim, key;

printf("\nQuanti elementi vuoi inserire? ");
scanf("%d", &dim);

/*allocazione dinamica e inserimento dati dell'array originario */
array =(int *)calloc(dim, sizeof(int));

for(i=0; i<dim; i++)
{
        printf("\nInserire l'elemento %d: \n", i+1);
        scanf("%d", &array[i]);
}


/* Dimensione del nuovo array vuoto costruito con il metodo hash: tale metodo      */
/* prevede un incremento standard della dimensione originaria pari al 25% +1       */
tablesize = dim + (dim*25/100)+1;

/*allocazione dinamica di memoria per il nuovo array hash*/
table =(int *)calloc(tablesize, sizeof(int));

/*Inizializzazione del nuovo array a NULL*/
for(i=0; i<tablesize; i++)
```

```
                table[i]=NULL;

/* Chiamata della funzione per costruire il nuovo array con il metodo hash */
        hash (array, table, dim, tablesize);

/*Inserimento chiave di ricerca */
        printf("\nDigitare l'elemento da cercare: ");
        scanf("%d", &key);

/*Stampa risultato */
        if(hash_search(table, tablesize, key))
                printf("\nL'elemento e' presente nel vettore");
        else
                printf("\nL'elemento non e' presente nel vettore");

}
```

esempio di esecuzione

```
Quanti elementi vuoi inserire? 6
Inserire l'elemento 1:
3
Inserire l'elemento 2:
5
Inserire l'elemento 3:
7
Inserire l'elemento 4:
1
Inserire l'elemento 5:
2
Inserire l'elemento 6:
0
Digitare l'elemento da cercare: 4
L'elemento non e' presente nel vettore
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
Quanti elementi vuoi inserire? 5
Inserire l'elemento 1:
4
Inserire l'elemento 2:
2
Inserire l'elemento 3:
6
Inserire l'elemento 4:
9
Inserire l'elemento 5:
11
Digitare l'elemento da cercare: 9
L'elemento e' presente nel vettore
```

FUNZIONE HASH *(costruisce la tabella per la ricerca con il metodo hash)*

In C:

```
void hash (int *array, int *table, int dim, int tablesize)
{
        int vett_app; /*vettore di appoggio con la stessa dimensione dell'array originario */
        int n, j, k, i;

        vett_app = (int *) calloc ( dim , sizeof(int));
        n=0; /*indice del vettore di appoggio */

/*calcolo del modulo di ciascun elemento (elemento/dimensione dell'array costruito) che */
/*servirà da indice per stabilire in che posizione collocare tale elemento nell'array   */
/*costruito. L'indice cosi ottenuto è memorizzato in j */
for(i=0; i<dim; i++)
{
        j = array[i] % tablesize;

/*Se la posizione di indice j è vuota allora l'elemento verrà ivi collocato, altrimenti */
/*esso verrà posto nel vettore di appoggio */
        if (table[j]==NULL)
                table[j]=array[i];
        else
        {
                vett_app[n]=array[i];
                n++;
        }
}

/*calcolo il modulo degli elementi presenti nel vettore di appoggio memorizzandolo*/
/* nell'indice k… */
        for(i=0; i<n; i++)
        {
                k = vett_app[i] % tablesize;

/*…e, se la posizione ottenuta è già occupata si scorre la tabella finché non si trova */
/* posizione dove inserire */
        while(table[k] != NULL)
        k++;
        table[k]=vett_app[i];
        }

free (vett_app); /*deallocazione dello spazio usato per vett_app */
}
```

FUNZIONE HASH SEARCH *(effettua la ricerca nella tabella calcolata hash)*

In C:

```
logical hash_search(int *table, int tablesize, int key)
{
        logical active, found;
        int start, position, temp;

        active = vero; /* la ricerca deve continuare */
        found = falso; /* elemento non ancora trovato */

/*calcolo del modulo di key memorizzato in start, che permette di stabilire l'indice  */
/* per il primo accesso alla tabella. Esso viene poi posto in un indice temporaneo    */
        start = key % tablesize;
        position =start;

        if (table[position]==key)
        {
                active = falso; /* interrompere la ricerca */
                found = vero; /* elemento trovato */
                temp = table[position];
        }
        else
        {
                temp = table[position];
                table[position]=key;
        }

while(active) /* la ricerca continua */
{
        /*calcola un nuovo indice di accesso */
        position =(position+1) % tablesize;

        /*se trovo la chiave….*/
        if(table[position]==key)
        {
                active = falso;/*….ferma la ricerca */
                        if(position != start)
                                found =vero;
        }
        else

/*la ricerca si arresta se viene trovata una casella vuota */
        if(table[position]==NULL)
                active = falso;
```

```
                table[start]=temp;
 }
/*la variabile che ritorna il valore al programma chiamante indica se l'elemento è */
/*stato trovato */
        return found;

}
```

RICERCA SEQUENZIALE

Documentazione esterna

- **Scopo:** stabilire se un dato assegnato appartiene ad un insieme, usando il metodo della ricerca sequenziale.

- **Specifiche:**

  int Ricerca_seq (int n, int i, int CHIAVE, int *a)
  int e;

- **Descrizione:** utilizzando un ciclo di tipo while, la funzione esegue una serie di confronti tra l'elemento da ricercare ed i vari componenti dell'insieme, fino a che l'elemento in questione non viene trovato.

- **Lista dei parametri:**

  n: intero. Rappresenta la dimensione dell'insieme in cui effettuare la ricerca.
  i: intero. Indice.
  CHIAVE: intero. Elemento da ricercare.
  ***a:** intero. Puntatore all'array degli elementi.
  e: intero. Variabile che indica quando l'elemento viene trovato. Il valore 0 corrisponde a FALSO; il valore 1 corrisponde a VERO.

- **Complessità computazionale:** l'operazione dominante del programma è il confronto. Nel caso peggiore verranno effettuati n-1 confronti; nel caso migliore 1 confronto.

- **Esempio d'uso:**

```
esempio di programma chiamante
#include <stdio.h>
#include <stdlib.h>

/*prototipo */
        int Ricerca_seq (int n, int i, int CHIAVE, int a[]);

main( )
{

/*variabili */
        int E;
        int n, i, r, *a;
```

```
printf("Quanti sono gli elementi del vettore? ");
scanf("%d", &n);
/*allocazione dinamica e inserimento dati */
        a=(int*) calloc (n, sizeof (int));

        printf("\nScrivi gli elementi del vettore: \n");

        for(i=0;i<n;i++)
        {
                scanf("%d", &a[i]);
        }

        printf("\nScrivi l'elemento da cercare\n ");
        scanf("%d", &CHIAVE );

        Ricerca_seq (n, i, CHIAVE, a); /*chiamata della funzione */

        if(E==1)
        {
                printf("L'elemento è stato trovato!");
        }
        else
        {
                printf("L'elemento non è stato trovato!");
        }
}
```

esempio di esecuzione

```
Quanti sono gli elementi del vettore ?
6
Scrivi gli elementi del vettore:
2 3 4 5 6 7
Scrivere l'elemento da cercare
4
L'elemento è stato trovato!
**************************************
Quanti sono gli elementi del vettore ?
5
Scrivi gli elementi del vettore:
8 11 13 21 30
Scrivere l'elemento da cercare
7
L'elemento non è stato trovato!
```

FUNZIONE RICERCA SEQUENZIALE

In C:

```
int Ricerca_seq (int n, int i, int CHIAVE, int *a)
{
/*indicatore che suggerisce quando l'elemento è stato trovato */
/* 0 = non trovato; 1 = trovato                                 */
        int e;

        e=0;
        i=0;

/*finché ci sono ancora elementi eseguire la ricerca */
        while(i<n)
        {
        if(CHIAVE == a[i])
                {
                e=1;
                }
        i++;
        }


        return e;
}
```

ALGORITMI FLOATING-POINT

EPSILON MACCHINA (singola precisione)

Documentazione esterna

- **Scopo:** calcolare l'epsilon macchina di un calcolatore in singola precisione (max 23 cifre decimali).

- **Specifiche:**

  function EPSILON_MACCHINA (int b, float E)
  float eps, E2;

- **Descrizione:** utilizzando un ciclo di tipo while, la funzione calcola il valore di epsilon, dividendo una variabile temporanea per la base, fino a quando, trovato tale valore, lo restituisce al programma chiamante.

- **Lista dei parametri:**

  b: intero. Rappresenta la base su cui si opera.
  E: float. Variabile in cui viene memorizzato il valore di epsilon.
  E2: float. Variabile temporanea.
  eps: float. Variabile che restituirà il valore al programma chiamante.

- **Complessità computazionale:** il programma implica un'esecuzione di divisioni, addizioni e una conversione temporanea di tipo della variabile "base".

- **Esempio d'uso**

esempio di programma chiamante

```
#include <stdio.h>

/*prototipo della funzione EPSILON_MACCHINA */
        float EPSILON_MACCHINA (int b, float E);

main ( )
{
/*dichiarazione di variabili */
        float E, epsilon;
        int base;

        E =1.0;
        base =2;
/*richiamo della funzione EPSILON_MACCHINA */
        epsilon =EPSILON_MACCHINA (base, E);
```

```
/*stampa risultato */
        printf ("\nEpsilon macchina=%.16f\n", epsilon);
}
```

esempio di esecuzione

avendo stabilito nell'esempio di programma chiamante di visualizzare il risultato con 16 cifre decimali avremo:

epsilon =0.0000001192092896

FUNZIONE EPSILON MACCHINA

In C:

```
float EPSILON_MACCHINA (int b, float E)
{

/*dichiarazione di variabili */
        float eps, E2;

        while (E2!=1)
        {
                eps =E;
                E =E/(float)b;
                E2 =E+1;
        }
        /*ritorna il valore dell'epsilon */
        return eps;
}
```

EPSILON MACCHINA (doppia precisione)

Documentazione esterna

- **Scopo:** calcolare l'epsilon macchina di un calcolatore in doppia precisione (max 52 cifre decimali).

- **Specifiche:**

  function EPSILON_MACCHINA (int b, double E)
  double eps, E2;

- **Descrizione:** utilizzando un ciclo di tipo while, la funzione calcola il valore di epsilon, dividendo una variabile temporanea per la base, fino a quando, trovato tale valore, lo restituisce al programma chiamante.

- **Lista dei parametri:**

  b: intero. Rappresenta la base su cui si opera.
  E: double. Variabile in cui viene memorizzato il valore di epsilon.
  E2: double. Variabile temporanea.
  eps: double. Variabile che restituirà il valore al programma chiamante.

- **Complessità computazionale:** il programma implica un'esecuzione di divisioni, addizioni e una conversione temporanea di tipo della variabile "base".

- **Esempio d'uso**

esempio di programma chiamante

```
#include <stdio.h>

/*prototipo della funzione EPSILON_MACCHINA */
        double EPSILON_MACCHINA (int b, double E);

main ( )
{
/*dichiarazione di variabili */
        double E, epsilon;
        int base;

        E =1.0;
        base =2;
/*richiamo della funzione EPSILON_MACCHINA */
        epsilon =EPSILON_MACCHINA (base, E);
```

```
/*stampa risultato */
        printf ("\nEpsilon macchina=%.32f\n", epsilon);
}
```

esempio di esecuzione

avendo stabilito nell'esempio di programma chiamante di visualizzare il risultato con 32 cifre decimali avremo:

epsilon =0.00000000000000022204460492503131

FUNZIONE EPSILON MACCHINA

In C:

```
double EPSILON_MACCHINA (int b, double E)
{

/*dichiarazione di variabili */
        double eps, E2;

        while (E2!=1)
        {
                eps =E;
                E =E/(double)b;
                E2 =E+1;
        }
        /*ritorna il valore dell'epsilon */
        return eps;
}
```

MINIMO RAPPRESENTABILE (in singola precisione)

Documentazione esterna

- **Scopo:** la funzione individua il più piccolo numero rappresentabile del sistema aritmetico, in singola precisione.

- **Specifiche:**

  float minimo_rappresentabile_sp (float x, int b)
  float rmin;

- **Descrizione:** utilizzando un ciclo while, la funzione effettua ripetute divisioni del numero x per la base (che in questo caso è 2), allo scopo di determinare il più piccolo numero rappresentabile.

- **Lista dei parametri:**

  x: float.
  b: float. E' la base di calcolo.
  rmin: float. Variabile temporanea che restituisce il valore al programma chiamante.

- **Complessità computazionale:** dipende dalla precisione del sistema aritmetico.

- **Esempio d'uso:**

```
esempio di programma chiamante
#include <stdio.h>

/*prototipo */
float minimo_rappresentabile_sp (float, int);
main ( )
{
/*dichiarazione di variabili */
 float e, rmins;
 int base;
 e=1.;
 base=2;
/*chiamata della funzione */
 rmins=minimo_rappresentabile_sp (e, base);
/*stampa risultato */
 printf ("\n Minimo Rappresentabile in Singola Precisione: %E", rmins);
}
```

esempio di esecuzione
Minimo Rappresentabile in Singola Precisione: 1.401298E-45

FUNZIONE MINIMO RAPPRESENTABILE sp
In C:

```
float minimo_rappresentabile_sp (float x, int b)
{
	float rmin;
	while (x != 0.)
	{
		rmin = x;
		x=rmin/(float)b;
	}

	return rmin;
}
```

MINIMO RAPPRESENTABILE (in doppia precisione)

Documentazione esterna

- **Scopo:** la funzione individua il più piccolo numero rappresentabile del sistema aritmetico, in doppia precisione.

- **Specifiche:**

  double minimo_rappresentabile_dp (double x, int b)
  double rmin;

- **Descrizione:** utilizzando un ciclo while, la funzione effettua ripetute divisioni del numero x per la base (che in questo caso è 2), allo scopo di determinare il più piccolo numero rappresentabile.

- **Lista dei parametri:**

  x: double.
  b: double. E' la base di calcolo.
  rmin: double. Variabile temporanea che restituisce il valore al programma chiamante.

- **Complessità computazionale:** dipende dalla precisione del sistema aritmetico.

- **Esempio d'uso:**

esempio di programma chiamante

```
#include <stdio.h>

/*prototipo */
        double minimo_rappresentabile_dp (double, int);

main ( )
{
/*dichiarazione di variabili */
        double e, rmind;
        int base;

e=1.;
base=2;

/*chiamata della funzione */
```

```
        rmind = minimo_rappresentabile_dp (e, base);

/*stampa risultato */
printf ("\n Minimo Rappresentabile in Doppia Precisione: %E", rmind);
}
```

esempio di esecuzione

Minimo Rappresentabile in Doppia precisione: 9.807802E-78

FUNZIONE MINIMO RAPPRESENTABILE dp

In C:

```
double minimo_rappresentabile_dp (double x, int b)
{
	double rmin;
	while (x != 0.)
	{
		rmin = x;
		x=rmin / (double)b;
	}

	return rmin;

}
```

ALGORITMI DI COMPLESSITA' COMPUTAZIONALE

VALUTAZIONE DI UN POLINOMIO IN UN PUNTO
(Algoritmo di Horner)

Documentazione esterna

- **Scopo:** effettuare la valutazione di un polinomio in un punto prefissato, utilizzando il metodo sperimentato da Hostrowski nel 1954 e noto come algoritmo di Horner.

- **Specifiche:**

  float HORNER (int n, float c[ ], float x)
  float ris;
  int i;

- **Descrizione:** effettuando un ciclo for e partendo dal coefficiente di grado più alto, la funzione calcola il valore del polinomio in un punto prefissato.

- **Lista dei parametri:**
  n: intero. E' il grado del polinomio
  i: intero. Indice dell'array dei coefficienti
  c[ ]: float. Array dei coefficienti
  x: float. Punto in cui valutare il polinomio
  ris: float. Variabile che conterrà il valore del polinomio da restituire alla funzione chiamante

- **Riferimenti bibliografici:**

  Almerico Murli
  LEZIONI DI LABORATORIO DI PROGRAMMAZIONE E CALCOLO
  Liguori Editore

- **Complessità computazionale:** la complessità computazionale dell'algoritmo è di **n moltiplicazioni** ed **n addizioni.**

- **Esempio d'uso**

```
esempio di programma chiamante
# include < stdio.h>

/*prototipo della funzione horner */
        float horner (int n, float c[ ], float x);
main ( )
{
/*variabili */
        int i, n;
```

```
        float c[100], x, v;
        printf ("Inserire il grado del polinomio ");
        scanf ("%d", &n);

/* Inserimento dei coefficienti del polinomio in ordine decrescente */
        for( i = n; i>=0;i--)
        {
                printf ("Inserire il coefficente di %d grado ",i);
                scanf ("%f", &c[i]);
        }

/* valore del punto in cui si deve valutare il polinomio */
        printf ("Inserire il valore della x in cui si deve valutare il polinomio ");
        scanf ("%f", &x);

/* chiamata della funzione Horner */
        v = horner (n, c, x);

/* stampa del risultato */
        printf ("Per x = %3.2f il polinomio vale %3.2f\n", x, v);

}
```

esempio di esecuzione

```
Inserire il grado del polinomio: 5
Inserire il coefficiente di 5° grado: 4
Inserire il coefficiente di 4° grado: 7
Inserire il coefficiente di 3° grado: 12
Inserire il coefficiente di 2° grado: 3.0
Inserire il coefficiente di 1° grado: 6.42
Inserire il coefficiente di 0° grado: 9.1
Inserire il valore della x in cui si deve valutare il polinomio: 8
Per x = 8.00 il polinomio vale 166140.45
**********************************************************************
Inserire il grado del polinomio 3
Inserire il coefficiente di 3° grado: 5.6
Inserire il coefficiente di 2° grado: 8.21
Inserire il coefficiente di 1° grado: 7.2
Inserire il coefficiente di 0° grado: 11.0
Inserire il valore della x in cui si deve valutare il polinomio: 5
Per x = 5.00 il polinomio vale 952.25
```

FUNZIONE HORNER

In C:

```
float horner (int n, float c[ ], float x)
{
/*variabili */
        int i;
        float ris;

        ris = c[i];


for ( i=n-1; i>=0; i--)
        ris = ris * x + c[ i ];

return ris;

}
```

RICORSIONE

BINARY SEARCH vers. ricorsiva

Documentazione esterna

- **Scopo:** dato un elemento x ed un insieme di dati ordinati in senso crescente, stabilire se x appartiene all'insieme.

- **Specifiche:**

  int Binarysearch_ric(int Inizio, int Fine, int *Elenco, int Chiave);
  int Indice;

- **Descrizione:** la function utilizza un approccio ricorsivo. Infatti, se l'elemento ricercato è maggiore dell'elemento selezionato dividendo in due parti l'array degli elementi, ci sarà un ritorno alla funzione stessa considerando però la parte destra dell'insieme: viceversa la funzione esaminerà la parte sinistra, fino a che l'elemento non verrà trovato.

- **Lista dei parametri:**

  Inizio: intero. Limite inferiore dell'insieme su cui opera la funzione.
  Fine: intero. Limite superiore dell'insieme su cui opera la funzione
  ***Elenco:** intero. Puntatore all'insieme degli elementi.
  Chiave: intero. Elemento da ricercare.
  Indice: intero. Elemento medio usato per il confronto.

- **Complessità computazionale:** l'operazione dominante eseguita dalla function è il confronto. Nel caso migliore (x trovato immediatamente), viene eseguito un solo confronto. Nel caso peggiore la complessità computazionale equivale a log2 (n).

- **Esempio d'uso:**

```
esempio di programma chiamante
#include <stdio.h>
#include <stdlib.h>

/*prototipo della funzione Binarysearch_ric */
        int Binarysearch_ric(int Inizio, int Fine, int *Elenco, int Chiave);

main ( )
{

/*dichiarazione variabili */
```

```
        int n, k, Posto, Chiave;
        int *Elenco;

printf("Quanti elementi compongono l'elenco ordinato in cui cercare? \n”);
scanf("%d", &n);

/*allocazione di memoria e inserimento elementi dell’elenco */
        Elenco = (int*)calloc(n, sizeof(int));

for (k = 0; k < n; k++)
{
        printf (“Elemento numero %d”, k);
        scanf("%d", &Elenco[k]);
}


        printf("Digitare l'elemento da cercare: \n");
        scanf("%d", &Chiave);

/*chiamata della funzione: viene restituita la posizione dell’elemento. Se esso non è
presente, verrà restituito il valore 0 */
        Posto = Binarysearch_ric(0, n - 1, Elenco, Chiave);

        if (Posto == 0)
                printf("L'elemento non è presente nell'elenco...\n");

        else
                printf("L'elemento %d compare al %d° posto. \n", Chiave, Posto);

}
```

esempio di esecuzione

```
Quanti elementi compongono l'elenco ordinato in cui cercare?
6
Elemento numero 0:   5
Elemento numero 1:   7
Elemento numero 2:   9
Elemento numero 3: 11
Elemento numero 4: 22
Elemento numero 5: 26
Digitare l'elemento da cercare:
26
L'elemento 26 compare al 6 posto.
```

FUNZIONE BINARYSEARCH_RIC

In C:

```
int Binarysearch_ric (int Inizio, int Fine, int Elenco[ ], int Chiave)
{

        int Indice;

        Indice = (Inizio + Fine) / 2;

        if (Elenco[Indice] = Chiave)

                return Indice + 1;

        else if (Inizio = Fine)

                return 0;

        else

        if (Elenco[Indice] < Chiave)
                return Binarysearch_ric(Indice + 1, Fine, Elenco, Chiave);

        else
                return Binarysearch_ric(Inizio, Indice - 1, Elenco, Chiave);


}
```

BUBBLE SORT vers. ricorsiva

Documentazione esterna

- **Scopo:** ordinare un insieme di n elementi con il metodo degli interscambi ricorrendo ad un approccio di tipo ricorsivo.

- **Specifiche:**

  void BUBBLESORT_Ric (int *m, int n);
  int k;
  int i;

- **Descrizione:** utilizzando un approccio di tipo ricorsivo e l'ausilio di una funzione che si occupa di effettuare lo scambio delle variabili, l'algoritmo controlla se due elementi contigui dell'array sono ordinati: se lo sono, la funzione prosegue richiamando se stessa per ordinare gli elementi successivi, altrimenti viene richiamata la funzione swap per effettuare lo scambio di variabili.

- **Lista dei parametri:**

  ***m:** intero. Puntatore all'array degli elementi.
  n: intero. Dimensione dell'array.
  k: intero. Indice della prima componente scambiata. Viene inizialmente posto a 0.
  i: intero. Indice di lavoro.

- **Complessità computazionale:** la complessità di tempo di questo algoritmo dipende dal valore dei dati forniti, i quali influenzano il numero di confronti e scambi da effettuare. Infatti, nel caso migliore(elementi già ordinati) si effettueranno solo n-1 confronti e nessuno scambio. Se invece l'array è completamente disordinato, allora bisognerà effettuare n-1 passaggi ed un numero di confronti e scambi pari a n(n-1)/2.

- **Funzioni ausiliarie:**

  void swap (int *m, int i, int j)
  int temp;

- **Esempio d'uso:**

esempio di programma chiamante
#include <stdio.h>
#include <stdlib.h>

```
/*prototipi */
void BUBBLESORT_Ric (int *m, int n );
void swap (int *m, int i, int j);

main( )
{
/*dichiarazione di variabili */
        int *m;
        int n, i;

        printf ("\nQuanti elementi vuoi inserire? ");
        scanf ("%d", &n);
/* Allocazione dinamica dell'array e inserimento dati */
        m=(int *) calloc (n, sizeof(int));

for (i=0;i<n;i++)
{
        printf ("Inserire il %d elemento: ",i+1);
        scanf ("%d", &m[i]);
}

/* Richiamo della procedura di ordinamento ricorsiva */
        BUBBLESORT_Ric (m, n);
/* Stampa risultato */
printf ("Array ordinato con il BUBBLE SORT ricorsivo: \n");

for (i=0;i<n;i++)
        printf ("%d \n", m[i]);

}
```

esempio di esecuzione

```
Quanti elementi vuoi inserire? 5
Inserire il 1 elemento: 234
Inserire il 2 elemento: 112
Inserire il 3 elemento: 2
Inserire il 4 elemento: 0
Inserire il 5 elemento: 73
Array ordinato con il BUBBLE SORT ricorsivo
0
2
73
112
234
```

FUNZIONE BUBBLESORT_Ricorsiva

In C:

```
void BUBBLESORT_Ric (int *m, int n )
{

/*  indice della prima componente  scambiata */
        int k=0;
/* indice di lavoro */
        int i;

for( i=0; i<n-1; i++ )
{
        if( m[i] > m[i+1] )

        {
                swap( m,i,i+1 );
                k = i;
        }


}

        if( k > 0 )
                BUBBLESORT_Ric (m, k+1);

}
```

FUNZIONE SWAP (scambio)

In C:

```
void swap (int *m, int i, int j)
{
        int temp;
/* Scambio delle due variabili */
        temp = m[i];
        m[i] = m[j];
        m[j] = temp;
}
```

INSERTION SORT vers. ricorsiva

Documentazione esterna

- **Scopo:** il presente programma si propone di ordinare in modo non decrescente un insieme di n elementi disposti in modo casuale utilizzando il metodo di inserzione.

- **Specifiche:**

  void INSERTION_SORT_ric (int *M, int k, int n)
  int j, v, i;

- **Descrizione:** Il vettore m di n componenti e' ordinato da 0 a k. Se k=n-1, tutto il vettore e' ordinato. Se k<(n-1), sia j (0<=j<=k) la posizione del primo elemento maggiore di v = M[k+1]. Il vettore ordinato si ottiene spostando gli elementi da j a k di una posizione a destra e copiando v in M[j]. Il resto del vettore si ordina richiamando la funzione con (M, k+1, n).
  La determinazione del valore di j e gli spostamenti si possono fare in una sola scansione della parte di vettore 0,...,k.
  Per non perdere valori, gli spostamenti si devono fare procedendo da k a j.

- **Lista dei parametri:**

  ***M:** puntatore al vettore di elementi da ordinare.
  k: intero. Indice del vettore da ordinare.
  n: intero. E' il numero di elementi del vettore.
  j, v, i: interi. Sono indici di lavoro.

- **Raccomandazioni d'uso:** il programma è consigliato soltanto per ordinare piccoli insiemi di dati.

- **Complessità computazionale:** le operazioni dominanti dell'algoritmo sono confronti e inserimenti. Nel caso peggiore, quando cioè bisogna riordinare successioni ordinate in senso opposto, l'andamento riscontrato della complessità asintotica è O(n^2) [quadratico]. Invece, nel caso migliore, cioè quando l'insieme è già ordinato, la complessità risulta O(n) [lineare].

- **Esempio d'uso:**

esempio di programma chiamante

```
#include <stdio.h>
#include <stdlib.h>

    /*prototipo */
    void INSERTION_SORT_ric (int *M, int k, int n);
```

```
main( )
{
 /* dichiarazione di variabili */
        int *M;
        int n, i ;

        printf ("\nQuanti elementi vuoi inserire ? ");
        scanf ("%d", &n);

/* Allocazione dinamica e inserimento dati */
        M = (int *)calloc(n, sizeof(int));

for (i=0; i<n; i++)
{
        printf ("Inserire il %d elemento: ",i+1);
        scanf ("%d", &M[i]);
}

/* Richiamo della procedura di ordinamento ricorsiva */
        INSERTION_SORT_ric (M,0,n);
/* Stampa del risultato */
        printf ("Array ordinato con l'Insertion Sort ricorsivo: \n");

                for (i=0;i<n;i++)
                        printf ("%d \n", M[i]);

}
```

esempio di esecuzione

```
Quanti elementi vuoi inserire? 8
elemento numero 1 4
elemento numero 2 1
elemento numero 3 7
elemento numero 4 23
elemento numero 5 6
elemento numero 6 78
elemento numero 7 0
elemento numero 8 11
Array ordinato con l'Insertion Sort ricorsivo:
0 1 4 6 7 11 23 78
```

FUNZIONE RICORSIVA DI INSERTION SORT

In C:

```
void INSERTION_SORT_ric (int *M, int k, int n)
{

/* dichiarazione variabili locali */
        int j,v,i; /* indici di lavoro */

        if( k < n-1 )
        {
                j = 0;
                v = M [k+1];

                while( j <= k && m[j] <= v )
                j++;
                        for(i=k; i>=j;i--)
                        {
                                M[i+1] = M[i];      /* crea lo spazio */
                                M[j]=v;             /* inserisce */
                        }

/*chiamata ricorsiva che ordina la parte restante */
        INSERTION_SORT_ric (M, k+1, n);

        }

}
```

MERGE (fusione di due sequenze ordinate): vers. ricorsiva

Documentazione esterna

- **Scopo:** ordinare un insieme di n elementi utilizzando il metodo denominato merge.

- **Specifiche:**

  void merge_sort(int *a, int inizio, int fine);
  int medio;

- **Descrizione:** la funzione merge sort suddivide l'insieme in 2 sequenze, ordina separatamente ciascuna di esse con una chiamata ricorsiva, e dopo, chiamando la funzione merge, le due sequenze ordinate vengono fuse in un unico insieme finale ordinato.

- **Lista dei parametri:**

  ***a:** puntatore all'array degli elementi.
  inizio: primo elemento dell'array.
  fine: ultimo elemento dell'array.
  medio: variabile locale usata per indicare il punto di confine tra le due sequenze separate da ordinare.

- **Funzioni ausiliarie:**

  void merge (int *a, int inizio1, int inizio2, int fine);
  int *ris;
  int i, j, k;

- **Complessità computazionale:** l'operazione dominante eseguita è il confronto, ma il numero dipende dal valore dei dati di input.

- **Esempio d'uso:**

```
esempio di programma chiamante
#include <stdio.h>
#include <stdlib.h>

/*prototipi */
        void merge_sort(int *a, int inizio, int fine);
        void merge (int *a, int inizio1, int inizio2, int fine);

main( )
```

```
{
/*dichiarazione di variabili */
        int *a;
        int n, i, j;

        printf ("\nQuanti elementi vuoi inserire? ");
        scanf ("%d", &n);

/*allocazione dinamica e inserimento dati */
        a = (int *) calloc (n, sizeof(int));

        for (i=0;i<n;i++)
        {
                printf("Inserire il %d elemento: ",i+1);
                scanf("%d", &a[i]);
        }

        i=0;
        j=n-1;

/*chiamata della funzione merge sort */
        merge_sort(a,i,j);


/*stampa risultato */
        printf ("\nArray ordinato con il metodo del Merge Sort:\n");

        for (i=0;i<n;i++)
                printf("%d \n", a[i]);

}
```

esempio di esecuzione

```
Quanti elementi vuoi inserire? 5
Inserire il 1 elemento: 82
Inserire il 2 elemento: 11
Inserire il 3 elemento: 23
Inserire il 4 elemento: 4
Inserire il 5 elemento: 9

Array ordinato con il metodo del Merge Sort:
4
9
11
23
82
```

FUNZIONE MERGE SORT

In C:

```
void merge_sort(int *a, int inizio, int fine)
{

/*variabili locali */
int medio;

        if (inizio < fine)
        {
                medio = (inizio + fine)/2;
                merge_sort(a, inizio, medio); /*chiamata ricorsiva 1 */
                merge_sort(a,medio+1,fine); /*chiamata ricorsiva 2 */
                merge (a,inizio,medio+1,fine); /*chiamata della funzione merge */

        }

}
```

FUNZIONE MERGE

In C:

```
void merge(int *a, int inizio1,int inizio2,int fine)
{
/*variabili locali */
        int *ris;
        int i, j, k;

        ris = (int *)calloc((inizio1+fine),sizeof(int));
        i=inizio1;
        j=inizio2;
        k=inizio1;

        while ((i <= inizio2-1) && (j <= fine))
        {
                if (a[i] < a[j])
                {
                        ris[k] = a[i];
                        i++;
                }

                else
                {
                        ris[k] = a[j];
```

```
                j++;
            }

            k++;
        }

        while (i <= inizio2-1)
        {
            ris[k] = a[i];
            i++;
            k++;
        }

        while (j <= fine)
        {
            ris[k] = a[j];
            j++;
            k++;
        }

        for (i=inizio1;i<=fine;i++)
            a[i]=ris[i];

        free (ris);
}
```

QUICK SORT vers. ricorsiva

Documentazione esterna

- **Scopo:** progettare ed implementare una versione ricorsiva dell'algoritmo di ordinamento rapido.

- **Specifiche:**

  void quick_sort_ric (int *a, int u, int v);
  int p;

- **Descrizione:** utilizzando un approccio di tipo ricorsivo, la funzione, dopo aver verificato che l'estremo inferiore è minore di quello superiore, procede con la suddivisione dell'insieme in due partizioni: sinistra e destra. Fatto ciò lavora su entrambe le stesse, effettuando delle chiamate ricorsive alla funzione.

- **Lista dei parametri:**

  ***a**: intero. Puntatore all'array degli elementi.
  u: intero. Estremo inferiore
  v: intero. Estremo superiore
  p: intero. Carica la funzione partizione

- **Funzioni ausiliarie:**

  int partition (int *a, int u, int v);
  void scambia(int *p, int *q);

- **Complessità computazionale:** Nel caso peggiore, cioè quando la dimensione dell'insieme è molto grande, la profondità della ricorsione può arrivare fino ad *n-1*.

- **Esempio d'uso:**

```
esempio di programma chiamante
#include <stdio.h>
#include <stdlib.h>

/*prototipi */
        void quick_sort_ric (int *a, int u, int v);
        int partition (int *a, int u, int v);
        void scambia(int *p, int *q);
```

```
main( )
{
/*dichiarazione di variabili */
        int *a;
        int n, i;

        printf ("\nQuanti elementi vuoi inserire? ");
        scanf ("%d", &n);

/* Allocazione dinamica e inserimento dati */
        a=(int *)calloc(n, sizeof(int));

        for (i=0;i<n;i++)
        {
                printf ("Inserire il %d elemento: ",i+1);
                scanf ("%d",  &a[i]);
        }

/* Richiamo della procedura ricorsiva */
        quick_sort_ric (a,0,n-1);

/*Stampa risultato */
        printf ("Array ordinato con il quick sort ricorsivo: \n");
                for (i=0;i<n;i++)
                        printf ("%d \n", a[i]);

}
```

esempio di esecuzione

```
Quanti elementi vuoi inserire? 5
Inserire il 1 elemento: 2
Inserire il 2 elemento: 1
Inserire il 3 elemento: 11
Inserire il 4 elemento: 73
Inserire il 5 elemento: 18
Array ordinato con il quick sort ricorsivo:
1
2
11
73
18
```

FUNZIONE QUICK SORT ricorsivo

In C:

```
void quick_sort_ric (int *a, int u, int v)
{
/*variabili locali */
        int p;


        if (u < v)
        {
                p = partition (a,u,v);

        if (u < p)
                quick_sort (a,u,p-1);

        else

        if (p < v)
                quick_sort (a,p+1,v);

        }

}
```

FUNZIONE PARTITION

In C:

```
int partition(int *a, int u, int v)
{

/*variabili locali */
        int i, j;

        i=u+1;
        j = v;

        while (i <= j)
        {

                while (a[i] < a[u])
                i++;

                        while (a[u] < a[j])
                        j--;
```

```
                        if (i < j)
                        {
                                scambia(&a[i], &a[j]);
                                i++;
                                j--;
                        }
        }

        if (a[u]>a[j])
                scambia(&a[u],&a[j]);
        else
                j = u;

return(j);

}
```

FUNZIONE SCAMBIA

In C:

```
void scambia(int *p, int *q)
{

        int temp;

        temp = *p;
        *p = *q;
        *q = temp;

}
```

SELECTION SORT vers. ricorsiva

Documentazione esterna

- **Scopo:** SELECTION SORT ordina (generalmente in modo crescente) un insieme di n elementi disposti casualmente utilizzando il metodo di selezione ed un approccio di tipo ricorsivo.

- **Specifiche:**

  void Selection_sort_ric (int *b, int i, int n)
  int j, small, temp;

- **Descrizione:** la funzione, prima seleziona l'elemento minimo e lo colloca in prima posizione, poi esegue un confronto con l'elemento successivo: se quest'ultimo è minore viene eseguito lo scambio e poi, effettuando una chiamata ricorsiva, si ordina la parte restante dell'insieme.

- **Lista dei parametri:**

  ***b:** puntatore all'array degli elementi.
  n: dimensione dell'insieme.
  small: posizione dell'elemento minimo.
  temp: variabile temporanea.
  i, j: indici.

- **Complessità computazionale:** le operazioni dominanti dell'algoritmo sono CONFRONTI e SCAMBI. Gli scambi eseguiti sono n-1. I confronti sono invece pari a n(n-1)/2.

- **Esempio d'uso:**

```
esempio di programma chiamante
#include <stdio.h>
#include <stdlib.h>

/*prototipo */
        void Selection_sort_ric (int *b, int i, int n);

main ( )
{
/*dichiarazione di variabili */
        int *b;
        int i, n;
```

```
        printf ("\Quanti elementi vuoi inserire? ");
        scanf ("%d", &n);

/*allocazione dinamica e inserimento dati */
        b = (int*) calloc (n, sizeof(int));

        for (i=0;i<n;i++)
        {
                printf ("Inserire il %d° elemento: ",i+1);
                scanf ("%d", &b[i]);
        }

/*richiamo della funzione */
        Selection_sort_ric (b, n);

/*stampa risultato */
        printf ("Array ordinato per selezione ricorsiva: ");

        for (i=0;i<n;i++)
                printf ("%d \n", b[i]);

}
```

esempio di esecuzione

```
Quanti elementi vuoi inserire? 6
Inserire il 1 elemento: 11
Inserire il 2 elemento:   2
Inserire il 3 elemento:   4
Inserire il 4 elemento: 23
Inserire il 5 elemento: 18
Inserire il 6 elemento: 90
Array ordinato per selezione ricorsiva:
2
4
11
18
23
90
```

FUNZIONE SELECTION SORT_Ricorsiva

In C:

```
void Selection_sort_ric (int *b, int i, int n)
{

        int j, small, temp; /*variabili locali */

if (i < n-1)
{ /* passo base: i = n-1. In questo caso la funzione
                    ritorna senza modificare A */
   small = i;
for (j = i+1; j < n; j++)
        if (b[j] < b[small])
                small = j;
                temp = b[small];
                b[small] = b[i];
                b[i] = temp;
/*ordina il resto per selezione ricorsiva */
        Selection_sort_ric (b, i+1, n);

        }

}
```

GESTIONE DI STRINGHE

MANIPOLAZIONE DI STRINGHE

Documentazione esterna

- **Scopo:** mostrare l'utilizzo di funzioni per la manipolazione di stringhe.

- **Specifiche:**

  void manipolazione_stringhe (char *s1, char *s2);

- **Descrizione:** l'algoritmo utilizza una serie di funzioni di libreria, incluse nell'header file *string.h* , atte a manipolare le due stringhe ricevute in input.

- **Lista dei parametri:**

  ***s1, *s2:** char. Puntatori alle due stringhe da manipolare.

- **Esempio d'uso:**

```
esempio di programma chiamante
#include <stdio.h>
#include <string.h>
#define  dim 30

/*prototipo */
void manipolazione_stringhe (char *s1, char *s2);

main ( )
{
/*variabili */
        char *s1,*s2;
        int a, b;

/*allocazione dinamica */
        s1=(char*) calloc (dim, sizeof(char));
        s2=(char*) calloc (dim, sizeof(char));

        printf ("\nInserire la prima stringa (max 30 caratteri): ");
        gets (s1);

        printf ("\nInserire la seconda stringa (max 30 caratteri): ");
        gets (s2);

/*chiamata della funzione */
        manipolazione_stringhe (s1,s2);
```

}

esempio di esecuzione

```
Inserire la prima stringa (max 30 caratteri): Cartello
Inserire la seconda stringa (max 30 caratteri): giallo
La prima stringa è lunga 8 caratteri
La seconda stringa è lunga 6 caratteri
La prima stringa precede la seconda
Trasformazione in lettere MAIUSCOLE:
Prima stringa maiuscola: CARTELLO
Seconda stringa maiuscola: GIALLO
Trasformazione in lettere minuscole:
Prima stringa minuscola: cartello
Seconda stringa minuscola: giallo
Prima stringa al rovescio: olletrac
Seconda stringa al rovescio: ollaig
Stringhe concatenate: cartellogiallo
```

FUNZIONE MANIPOLAZIONE STRINGHE

In C:

```
void manipolazione_stringhe (char *s1, char *s2)
{

/*strlen: restituisce la lunghezza della stringa*/
        printf ("La prima stringa é lunga %d caratteri \n", strlen(s1));
        printf ("La seconda stringa é lunga %d caratteri \n", strlen(s2));

/*strcmp: confronto di stringhe */
        if (strcmp(s1,s2) < 0)
                printf ("La prima stringa precede la seconda");

        else
        if (strcmp(s1,s2) > 0)
                printf ("La seconda stringa precede la prima");

        else
                printf ("Le due stringhe sono uguali");


/*strupr: trasforma la stringa in maiuscolo*/
        printf ("\nTrasformazione in lettere MAIUSCOLE: \n");
        printf ("Prima stringa maiuscola: %s\ n", strupr(s1));
        printf ("Seconda stringa maiuscola: %s \n", strupr(s2));

/*strlwr: trasforma la stringa in minuscolo */
        printf ("Trasformazione in lettere minuscole: \n");
        printf ("Prima stringa minuscola: %s \n", strlwr(s1));
        printf ("Seconda stringa minuscola: %s \n", strlwr(s2));

/*strrev: rovescia la stringa */
        printf ("Prima stringa al rovescio: %s \n", strrev(s1));
        strrev (s1);
        printf ("Seconda stringa al rovescio: %s \n", strrev(s2));
        strrev (s2);

/*strcat: concatenazione di stringhe */
        printf ("Stringhe concatenate: %s", strcat(s1,s2));
}
```