ALGORITMI DI ORDINAMENTO R.c.

----- Insertion sort -----
È un sistema di ordinamento attraverso la quale si estrae dal vettore il numero in considerazione; il valore “scala in alto” fino alla posizione successiva a quella di un numero più piccolo di lui.

Passiamo alla visualizzazione grafica per una maggiore comprensione:

1. Generiamo un vettore di 4 elementi:

vet[0] 4
vet[1] 3
vet[2] 1
vet[3] 2

2. L’algoritmo parte prendendo in considerazione il numero 3, ovvero il valore nella posizione 1. La variabile di appoggio assume il valore 3 per evitare che questo venga perso durante il suo posizionamento.

3. Gli elementi che si trovano al di sopra della posizione 1 sono quindi spostati verso il basso finché non si trova la posizione corretta in cui inserire il 3.

4. Il vettore modificato diventa:

vet[0] 3
vet[1] 4
vet[2] 1
vet[3] 2

5. Il ciclo riprende dal punto 2, solo che invece di considerare il valore nella posizione 1, si va a prendere quello nella posizione 2.

6. Il ciclo finisce quando anche l’ultimo valore del vettore viene inserito nella posizione corretta, generando il vettore definitivo:

vet[0] 1
vet[1] 2
vet[2] 3
vet[3] 4

Programma in C

```
for (i=1; i<4; i++) {
   app=vet[i];
   for (j=0; j<=i; j++) {
      if (vet[i]<vet[j]) {
         vet[i]=vet[j];
         vet[j]=app;
      }
   }
}
```

----- Bubble sort -----
È un sistema di ordinamento utilizzato nei vari linguaggi di programmazione per ordinare un vettore in ordine crescente o decrescente. L’algoritmo consiste nel considerare due valori alla volta, inseriti in posizioni consecutive, ponendo al primo posto quello maggiore. Attraverso un ciclo “for”, il programma analizza di volta in volta tutti i valori del vettore, fino a quando non li avrà ordinati completamente.

Il programma, per ipotesi, potrebbe iniziare con analizzare la prima e la seconda posizione del vettore, attraverso un “if” calcola il valore maggiore (o minore) e lo pone al posto di quello minore (nel caso in cui questo sia stato inserito prima di quello maggiore). Si passa poi a considerare la seconda e la terza posizione… e così via fino all’ultima posizione del vettore.

Vediamo ora come procederebbe il computer nel momento in cui deve ordinare un vettore:

1. Generiamo un vettore di lunghezza 10, aventi valori generati tra 1 e 100.

vet[0]	23
vet[1]	56
vet[2]	12
vet[3]	78
vet[4]	29
vet[5]	33
vet[6]	90
vet[7]	12
vet[8]	81
vet[9]	44

2. Creiamo un ciclo “for”, grazie al quale noi diciamo quante volte, al massimo, il programma deve ripetere il bubble sort. In questo caso il valore massimo è 10, perché se per caso il valore che noi vogliamo mettere in cima (che sia il minore o li maggiore) si trova nell’ultima posizione del vettore, il ciclo si ripeterà tante volte quanto è la lunghezza del vettore.

3. Attraverso un ciclo “for” il programma inizia con il considerare il vettore nella posizione 0 e 1.

4. Attraverso un “if”, il programma calcola quale dei due valori è il maggiore (56>23). Nel caso in cui i due valori fossero uguali (del tipo 12=12) il programma non sostituisce il secondo valore con il primo, perché l’operatore è “maggiore” e non “maggiore uguale”.

5. Si passa ora a sostituire il secondo valore con il primo, modificando quindi il vettore originale. Si avrà:

vet[0]	56
vet[1]	23
vet[2]	12
vet[3]	78
vet[4]	29
vet[5]	33
vet[6]	90
vet[7]	12
vet[8]	81
vet[9]	44

6. Il programma ripete poi lo stesso procedimento partendo dal punto 3, avendo però come vettore base, il vettore modificato (Vettore 1). Naturalmente esso non considererà il vettore nella posizione 0 e 1, ma stavolta nella posizione 1 e 2; poi nella posizione 3 e 4 e così via…

7. Alla fine risulterà il vettore ordinato:

vet[0]	12
vet[1]	12
vet[2]	23
vet[3]	29
vet[4]	33
vet[5]	44
vet[6]	56
vet[7]	78
vet[8]	81
vet[9]	90

Programma in C

```
for (i = 0; i < 10; i++) {
   for (j = 0; j < 10; j++) {
     if (vet[j] < vet[j+1]) {
        appoggio = vet[j];
        vet [j] = vet [j+1];
        vet [j+1] = appoggio;
```

```
    }
  }
}
```

La variabile appoggio serve per non perdere il valore di vet[j] perchè questo sarà poi sovrascritto dal valore inserito nella posizione j+1.

----- Quick sort -----
È il sistema più diffuso riguardo agli ordinamento dei vettori. L'algoritmo inizia prendendo come pivot (valore di riferimento) un numero qualsiasi del vettore (preferibilmente quello centrale per convenzione).

Si va a sostituire il pivot nella posizione che ha come valore il numero degli elementi minori del pivot.

Attraverso due indici l'algoritmo confronta il primo numero, partendo dall'alto, maggiore del pivot (indice i) con il primo numero, partendo dal basso, minore del pivot (indice j). Se il valore dell'indice "i" è maggiore del valore dell'indice "j" essi vengono scambiati.

Il ciclo si ripete fino a quando tutti gli elementi sopra al pivot risultano <= al pivot, e tutti gli elementi sotto il pivot risultano >= al pivot.

Finita la divisione il vettore viene separato in due sotto-vettori, il primo composto dai numeri minori del vettore e il secondo formato, invece, da quelli maggiori. Il pivot viene quindi lasciato a parte per questa fase di ordinamento.

I sotto vettori ordinano i propri elementi attraverso il modo appena descritto, ovvero scelgono un valore come pivot e vengono posti, al sopra del pivot, i valori minori o uguali al pivot, e al di sotto del pivot, i valori maggiori o uguali al pivot.
Il processo si ripete fino a quando il vettore non viene completamente ordinato.

Passiamo alla rappresentazione grafica:

```
vet[0]   20
vet[1]   4
vet[2]   12
vet[3]   14
vet[4]   2
vet[5]   13
vet[6]   15
```

L'algoritmo prende come pivot il valore 14 nella posizione 3.

1. Si calcola il numero degli elementi minore del pivot, ovvero 4; quindi si va a sostituire il pivot nella posizione 4:

```
vet[0]   20
vet[1]   4
vet[2]   12
vet[3]   2
vet[4]   14
vet[5]   13
vet[6]   15
```

2. Ora si va alla ricerca del primo valore (indice i) maggiore di 14 e allo stesso tempo si ricerca il primo valore maggiore, partendo dal basso, di 14.

```
(i)-->   vet[0]   20
   vet[1]   4
   vet[2]   12
   vet[3]   2
   vet[4]   14
(j)-->   vet[5]   13
   vet[6]   15

   vet[0]   13
```

vet[1] 4
vet[2] 12
vet[3] 2
vet[4] 14
vet[5] 20
vet[6] 15

3. Si scompone ora il vettore principale in due sotto-vettori:

vet[0] 13
vet[1] 4
vet[2] 12
vet[3] 2
---
vet[5] 20
vet[6] 15

4. Per ogni vettore si lavora in modo separato, pre ognuno si sceglie un pivot e si ripete il procedimento dal punto 1, fino a quando il vettore non è ordinato.

vet[0] 13
vet[1] 4
vet[2] 12
vet[3] 2
---
vet[5] 20
vet[6] 15

5. Il 4 nel primo sotto-vettore e il 20 nel secondo sotto-vettore sono i nuovi pivot. Si procedere ora separatamente per l'ordinamento di ogni sotto-vettore.

vet[0] 13
vet[1] 4
vet[2] 12
vet[3] 2
---
vet[5] 15
vet[6] 20

6. Il secondo sotto-vettore è stato ordinato completamente; dedichiamoci ora al primo: abbiamo preso come pivot il valore 4 che è gia inserito nella posizione corretta. Passiamo ora al confronto tra il valore dell'indice "i" e il valore dell'indice "j".

(i)--> vet[0] 13
vet[1] 4
vet[2] 12
(j)--> vet[3] 2

vet[0] 2
vet[1] 4
vet[2] 12
vet[3] 13

7. Il vettore è stato ora completamente ordinato grazie al QuickSort:

vet[0] 2
vet[1] 4
vet[2] 12
vet[3] 13
vet[4] 14
vet[5] 15
vet[6] 20

Programma in Java

```
public static void QuickSort(int [] vector, int p, int r) {
```

```
   if(p < r) {
      int pivot_i = Partition(vector,p,r);
      QuickSort(vector,p,pivot_i);
      QuickSort(vector,(pivot_i+1),r);
   }
   return;
}

private static int Partition(int[] vector, int p, int r)
{
   int pivot = vector[p];
   int i = (p-1);
   int j = (r+1);
   while(true) {
      do {
         j = j - 1;
      } while(vector[j]>pivot);
      do {
         i = i + 1;
      } while(vector[i] < pivot);
      if(i < j) {
         int temp = vector[i];
         vector[i] = vector[j];
         vector[j] = temp;
      } else {
         return(j);
      }
   }
}
```