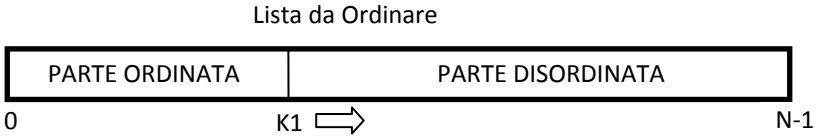
ALGORITMI DI BASE (3° Parte):
ORDINAMENTO "SELECTION-SORT", RICERCA BINARIA

Ordinamento di Dati tramite l'Algoritmo di Ordinamento chiamato "SELECTION-SORT"

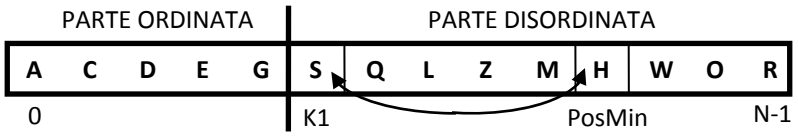
L'*Ordinamento dei Dati*, in informatica come nella vita quotidiana, è una operazione fondamentale per rendere più efficiente l'accesso ai dati e velocizzare le *Operazioni di Ricerca* dei dati stessi. Per questo motivo esistono molti *Algoritmi di Ordinamento*, ciascuno basato su una diversa tecnica e ciascuno dotato di diversi pregi e difetti.

L'Algoritmo di Ordinamento denominato SELECTION-SORT divide la Lista di Dati in due parti: la *Parte Ordinata* e la *Parte Disordinata*. All'inizio la Parte Disordinata costituisce l'intera Lista ma, al procedere dell'elaborazione, la Parte Ordinata *cresce* e la Parte Disordinata *decresce* fino a rendere ordinata l'intera Lista:

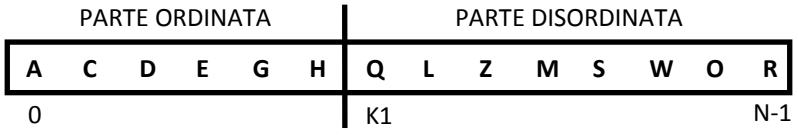


Un Ciclo Esterno, con indice K1, fa si che, ad ogni passo, la Parte Ordinata aumenti di un elemento e, di conseguenza, quella Disordinata si riduca di un elemento: l'indice K1 parte dalla posizione 0 e indica sempre *il primo elemento della Parte Disordinata*.

Ad ogni passo, per "espandere" la Parte Ordinata di un elemento, l'algoritmo prevede di Trovare il Minimo nella Parte Disordinata e, una volta determinata la sua posizione (posizione PosMin), prevede di Scambiare il Minimo con il Primo Elemento della Parte Disordinata (posizione K1):



A questo punto la Parte Ordinata è "cresciuta" di un elemento e il Ciclo Esterno può passare al prossimo passo *incrementando K1*:



Naturalmente il Ciclo Esterno varierà K1 da 0 a N-2 (ossia fino al *penultimo dato*), poiché l'ultimo dato rimasto sarà ovviamente il massimo e sarà già nella corretta posizione. Il Ciclo Esterno può quindi configurarsi come segue:

```
for ( int K1 = 0;  K1 <= N-2;  K1++ )
{   << determina PosMin ossia la Posizione del Minimo nella Parte Disordinata >>
    << scambia il Minimo (posizione PosMin) con il Primo Dato della Parte Disordinata (posizione K1) >>
}
```

L'operazione che determina *PosMin* è un normale algoritmo di *Ricerca del Minimo* che opera sulla Parte Disordinata (ossia da *K1* a *N-1*). Come è noto, si realizza con un ulteriore ciclo (Ciclo Interno) con indice K2 che *scandisce tutta la Parte Disordinata* dal suo secondo elemento (da *K1+1*) all'ultimo (a *N-1*). PosMin viene inizializzato a K1, ipotizzando inizialmente che il minimo sia il primo elemento della Parte Disordinata.

La seconda operazione è un semplice *Scambio di Dati* fra gli elementi in posizione *PosMin* e *K1*, da effettuare con l'ausilio di una Variabile di Appoggio *Temp*. Quindi il codice completo, per *l'algoritmo SELECTION-SORT*, è il seguente:

```
for ( int K1 = 0;  K1 <= N-2;  K1++ )          // ad ogni passo cresce la Parte Ordinata, decresce la Disordinata
{  // determina PosMin ossia la Posizione del Minimo nella Parte Disordinata …
   PosMin = K1;                                 // si parte ipotizzando che il minimo sia in prima posizione
   for ( int K2 = K1+1;  K2 <= N-1;  K2++ )     // il ciclo interno scandisce dal 2° all'ultimo elemento
       if ( Lista[K2] < Lista[PosMin] )         // se l'elemento in esame è minore del minimo trovato finora …
           PosMin = K2;                         // … allora è l'elemento in esame (posiz. K2) il nuovo minimo!

   // scambia il Minimo (posizione PosMin) con il Primo Dato della Parte Disordinata (posizione K1) …
   Temp = Lista[K1];                            //  salva, in Temp,  il dato in posizione K1
   Lista[K1] = Lista[PosMin];                   // ora puoi modificare l'elemento in K1 e assegnarvi il minimo
   Lista[PosMin] = Temp;                        // infine poni, nella posizione dove era il minimo, il dato che era in K1
}
```

Si noti come i due cicli K1 e K2 siano "annidati", per cui il numero di passi complessivo è sempre dell'ordine di N^2, *indipendentemente dallo stato iniziale di ordinamento dei dati* (anche se i dati fossero già ordinati, questo algoritmo compirebbe comunque circa ((N - 1) * (N / 2)) confronti ed (N - 1) scambi!).

Ricerca della Posizione di un Dato in una Lista
tramite l'Algoritmo di Ricerca chiamato "RICERCA BINARIA"

L'operazione di Ricerca consiste nel Determinare la Posizione di un Dato all'interno di una Lista e, quindi, nel verificarne anche l'esistenza. Il valore da individuare è presente nella variabile DatoDaCercare e il risultato, ossia la sua posizione nella lista, viene memorizzato nella variabile Pos: nel caso in cui il dato non sia presente nella lista, in Pos, si pone il valore particolare "-1".

L'algoritmo della Ricerca Binaria è applicabile solo se la lista contiene dei Dati Ordinati. La ricerca è effettuata su una Zona di Ricerca, inizialmente estesa a tutta la lista. Il Confronto fra il Dato da Cercare e l'Elemento Centrale della zona stessa, consente di dimezzare progressivamente la zona di ricerca, fino all'individuazione del dato o, in caso il dato non sia presente, alla scomparsa della zona.

La Zona di Ricerca è gestita con le variabili Inizio e Fine e, inizialmente essa si estende dalla posizione 0 alla posizione N-1. La variabile Centro serve a determinare la posizione dell'elemento centrale della zona. La variabile Pos viene inizializzata al valore -1 e, solo se il dato viene effettivamente trovato, viene modificata assegnandovi la posizione così individuata: quindi, se il dato non viene trovato, in Pos resta il valore iniziale -1.

```
int Pos = -1;                                   // La variabile Pos viene inizializzata a -1
int Inizio = 0;                                 // la zona di ricerca inizialmente si estende dalla prima posizione …
int Fine = N-1;                                 // … fino all'ultima posizione (N-1) della Lista …
int Centro;                                     // Centro serve per calcolare la posizione dell'elemento centrale
while ( ( Pos == -1 )  &&  ( Inizio <= Fine ) ) // mentre "non l'hai trovato" e "esiste ancora una zona di ricerca"
{
   Centro = ( Inizio + Fine ) / 2;              // calcola la posizione dell'elemento centrale della zona di ricerca
   if ( DatoDaCercare == Lista[Centro] )        // se il dato da cercare è proprio nell'elemento centrale …
       Pos = Centro;                            // … allora memorizza la sua posizione in Pos
   else
       if ( DatoDaCercare < Lista[Centro] )     // se il dato da cercare è MINORE dell'elemento centrale …
           Fine = Centro - 1;                   // … allora la zona di ricerca diventa:  Inizio … Centro-1
       else                                     //  il dato è sicuramente MAGGIORE dell'elemento centrale …
           Inizio = Centro + 1;                 // … quindi la zona di ricerca diventa:  Centro+1 … Fine
}
```

La Ricerca Binaria è estremamente efficiente e rapida: il numero di confronti che effettua è al massimo $\log_2 N$ contro un numero di confronti massimo pari a N nel caso di Ricerca Sequenziale.

Ad esempio, per una lista composta da 65536 elementi (si noti che $65536 = 2^{16}$), la Ricerca Sequenziale impiega al massimo **65536 confronti** mentre la Ricerca Binaria solo **16 confronti**.

Lo svantaggio della Ricerca Binaria è che essa è applicabile solo se i dati sono ordinati.

Ricerca della Posizione di Inserimento di un Nuovo Dato in una Lista da mantenere Ordinata

E' possibile utilizzare lo stesso algoritmo allo scopo di **Determinare la Posizione in cui Inserire un Nuovo Dato per mantenere la Lista ordinata**.

E' sufficiente eseguire il ciclo di dimezzamento **fino alla Scomparsa della Zona di Ricerca**.
A quel punto, **la Posizione di Inserimento è data dal valore contenuto nella variabile Inizio**.

```
int Inizio = 0;                          // la zona di ricerca inizialmente si estende dalla prima posizione …
int Fine = N-1;                          // … fino all'ultima posizione (N-1) della Lista …
int Centro;                              // Centro serve per calcolare la posizione dell'elemento centrale
while ( Inizio <= Fine )                 //  mentre "esiste ancora una zona di ricerca" …
{
   Centro = ( Inizio + Fine ) / 2;       // calcola la posizione dell'elemento centrale della zona di ricerca
   if ( DatoDaInserire < Lista[Centro] ) // se il dato da cercare è  <  dell'elemento centrale …
        Fine = Centro - 1;               //  … allora la zona di ricerca diventa:  Inizio … Centro-1
   else                                  //  il dato è sicuramente  >=  dell'elemento centrale …
        Inizio = Centro + 1;             // … quindi la zona di ricerca diventa:  Centro+1 … Fine
}                                        //  … a ciclo terminato, la Posizione di Inserimento è nella variabile Inizio
```

L'algoritmo opera correttamente anche nei casi particolari:

- Se il **Dato da Inserire è Minore di Tutti** quelli presenti nella lista, al termine dell'elaborazione, il valore della *variabile Inizio* sarà **0 (zero)**, cioè la **prima posizione**.
- Se il **Dato da Inserire è Maggiore di Tutti** quelli presenti nella lista, al termine dell'elaborazione, il valore della *variabile Inizio* sarà N, cioè la **prima posizione libera disponibile** oltre tutte quelle già occupate dagli altri dati.
- Se il **Dato da Inserire è già presente**, una o più volte, nella lista, al termine dell'elaborazione, il valore della *variabile Inizio* indicherà la **posizione subito successiva all'ultima occorrenza** del dato stesso nella lista.