CALCOLO DELL'ORDINE DI COMPLESSITÀ

```java
22    /**
23     * Restituisce il minimo del vettore a. Variante col ciclo while.
24     */
25    public static int min2(int[] a) {
26        int min = a[0];
27        int i = 1;
28        while (i < a.length) {
29            if (a[i] < min)
30                min = a[i];
31            i++;
32        }
33        return min;
34    }
```

Costo: $(C_2 + C_3 + C_4 + C_5)\, n + (C_0 + C_1 - C_3 - C_4 - C_5 + C_6)$ ← Assegniamo un costo c_i ad ogni operazione

Costo: $4n$ ← Mettiamo tutti i costi uguali ad 1

$\Downarrow$

Costo asintotico: $\Theta(n)$

```java
22    /**
23     * Restituisce il minimo del vettore a. Variante col ciclo while.
24     */
25    public static int min2(int[] a) {
26        int min = a[0];
27        int i = 1;
28        while (i < a.length) {
29            if (a[i] < min)
30                min = a[i];
31            i++;
32        }
33        return min;
34    }
```

n = dimensione dei dati di input

26: $\Theta(1)$

27: $\Theta(1)$

28: $\Theta(n)$ (non importa stabilire se le eseguiamo n volte, $n-1$ volte o simili; si tratta sempre di $\Theta(n)$)

29: $\Theta(n)$

30: $\Theta(n)$

31: $\Theta(n)$

33: $\Theta(1)$

TOT = massimo di tutti gli ordini di grandezza $= \Theta(n)$

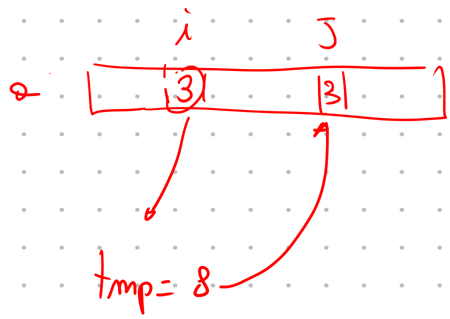
$\Theta(1) + \Theta(1) = \Theta(1)$

$\Theta(1) + \Theta(n) = \Theta(n)$

$\Theta(n) + \Theta(n) = \Theta(n)$

```java
/**
 * Scambia le posizioni i e j nel vettore a.
 */
public static void swap(int[] a, int i, int j) {
    int tmp = a[i];
    a[i] = a[j];
    a[j] = tmp;
}

/**
 * Restituisce la posizone in cui si trova il minimo
 * image.pngdell'array a, ma limitatamente alle
 * posizioni dalla j-esima in poi.
 */
public static int minPos(int[] a, int j) {
    int minPos = j;
    for (int i = j + 1; i < a.length; i++) {
        if (a[i] < a[minPos])
            minPos = i;
    }
    return minPos;
}
```



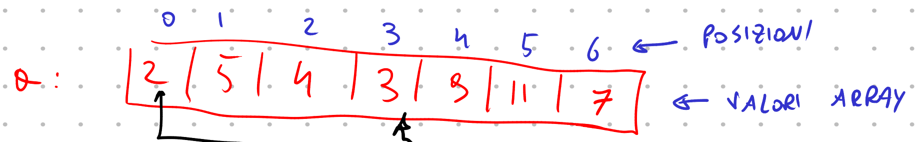
la posizione dove si trova l'elemento minimo tra tutti quelli provati fin'ora.

```java
/**
 * Ordina il vettore a usando l'algoritmo di ordinamento per selezione.
 */
public static void selectionSort(int[] a) {
    for (int i = 0; i < a.length - 1; i++) {
        int j = minPos(a, i);
        swap(a, i, j);
    }
}
```

ogni volta che il corpo del ciclo for parte, le posizioni da a[0] ad a[i-1] sono a posto, e devo sistemare quelle da a[i] in poi

l'ultimo valore di i è a.length-2 che è la penultima posizione dell'array.

restituisce la posizione tra gli elementi da sistemare (a[i], a[i+1], ...) trova il minimo e restituisce la sua posizione



minpos (a, 0) = 0

zero

minpos (a, 1) = 3

minpos (a, 4) = 6

```java
48      /**
49       * Scambia le posizioni i e j nel vettore a.
50       */
51      public static void swap(int[] a, int i, int j) {
52          int tmp = a[i];
53          a[i] = a[j];
54          a[j] = tmp;
55      }
56
57      /**
58       * Restituisce la posizone in cui si trova il minimo
59       * image.pngdell'array a, ma limitatamente alle
60       * posizioni dalla j-esima in poi.
61       */
62      public static int minPos(int[] a, int j) {
63          int minPos = j;
64          for (int i = j + 1; i < a.length; i++) {
65              if (a[i] < a[minPos])
66                  minPos = i;
67          }
68          return minPos;
69      }
70
```

$\Theta(1)$

CASO OTTIMO $j = a.length$

63: $\Theta(1)$

68: $\Theta(1)$

64: $\Theta(1)$

65, 66: 0

TOT: $\Theta(1)$

$a.length = n$

CASO PESSIMO: $j = 0$

63: $\Theta(1)$

64: $\Theta(n)$

65: $\Theta(n)$

66: $\Theta(n)$

68: $\Theta(1)$

$\Theta(n)$ max

```java
71      /**
72       * Ordina il vettore a usando l'algoritmo di ordinamento per selezione.
73       */
74      public static void selectionSort(int[] a) {
75          for (int i = 0; i < a.length - 1; i++) {
76              int j = minPos(a, i);
77              swap(a, i, j);
78          }
79      }
80
```

75: $\Theta(n)$

76: $\Theta(n) \cdot \Theta(n) = \Theta(n^2)$

77: $\Theta(n) \cdot \Theta(1) = \Theta(n)$

TOT: $\Theta(n^2)$

Significativamente più lento degli altri

In realtà, con una analisi più dettagliata, si vede che la complessità è proprio $\Theta(n^2)$

minPos è $\Theta(n)$ nel caso pessimo

$\Theta(1)$ nel caso ottimo

$\Downarrow$

in entrambi i casi è $O(n)$

```java
81      /**
82       * Restituisce la posizione di x nel vettore a, o -1 se
83       * volte, può essere restituita una posizione qualunque
84       * maniera crescente.
85       */
86      public static int binarySearch(int[] a, int x) {
87          int min = 0;
88          int max = a.length - 1;
89          while (min <= max) {
90              int pos = (min + max) / 2;
91              if (a[pos] == x)
92                  return pos;
93              else if (a[pos] > x)
94                  max = pos - 1;
95              else
96                  min = pos + 1;
97          }
98          return -1;
99      }
```

min = 7 Pos

a

0	1	2	3	4	5	6	7	8
1	3	3	5	8	11	11	22	90

$Pos_1 = \frac{min + max}{2}$

max = 8

x = 22

$Pos_3 = \frac{7+8}{2} = \frac{15}{2} = 7$

$Pos_2 = \frac{5+8}{2} = \frac{13}{2} = 6$

```java
81  /**
82   * Restituisce la posizione di x nel vettore a, o -1 se
83   * volte, può essere restituita una posizione qualunque
84   * maniera crescente.
85   */
86  public static int binarySearch(int[] a, int x) {
87      int min = 0;
88      int max = a.length - 1;
89      while (min <= max) {
90          int pos = (min + max) / 2;
91          if (a[pos] == x)
92              return pos;
93          else if (a[pos] > x)
94              max = pos - 1;
95          else
96              min = pos + 1;
97      }
98      return -1;
99  }
```

87, 88, 98: $\Theta(1)$

Sono tutte operazioni elementari, per cui la complessità è

$\Theta(\text{numero di iterazioni del while})$

$\uparrow$

ma quante sono?

- n
- $n/4$ ✓
- n/n ???
- n^2

Caso che analizziamo

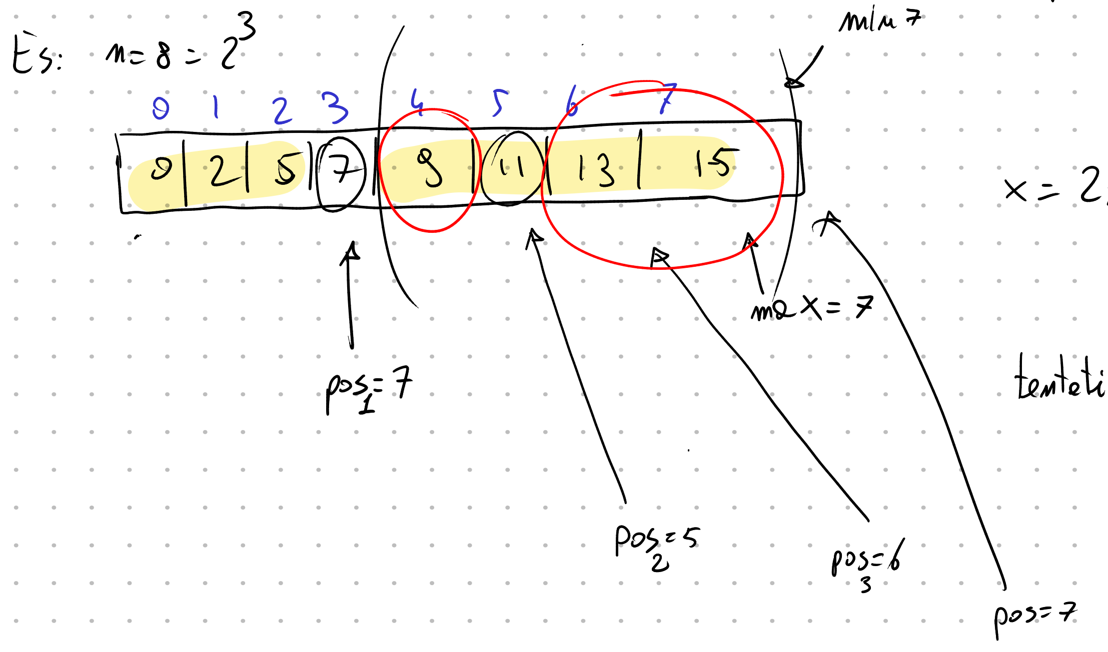
lunghezza di $a = n$ è una potenza di 2

$\Updownarrow$

$n = 2^k$ per qualche k

x non appartiene ad a (caso pessimo)

Es: $n = 8 = 2^3$



$x = 27$

tentativi: ~~1~~ ~~2~~ ~~3~~ 4

Se la parte dell'array che "osservo" (cioè quella compresa tra min e max) è lunga 2^k, l'elemento pos centrale divide l'array in una parte lung $\frac{2^k}{2} = 2^{k-1}$ ed un'altra lunga $\frac{2^k}{2} - 1 = 2^{k-1} - 1$. Se sono sfortunato (caso pessimo) l'elemento x devo cercarlo nella parte più lunga, quella lunga $\frac{2^k}{2} = 2^{k-1}$

Nel caso di prima, partendo da 8, osservo array lunghi $\underbrace{\overset{=2^3}{8}, 4, 2, 1}_{4 \text{ tentativi}}, \ldots$

Se parto da un array lungo 32?

$$\underbrace{\overset{=2^5}{32} \quad 16 \quad 8 \quad 4 \quad 2 \quad 1}_{6 \text{ tentativi}}$$

Dunque: quante volte posso dimezzare un vettore lungo $\overset{n}{\overset{\shortparallel}{2^k}}$ prima di ottenere un vettore lungo 1

Risposta $k+1$. Dunque il numero di iterazioni del while è $\Theta(k)$. Ma come esprimerlo in funzione di n?

$$2^k = n \leadsto \log_2 2^k = \log_2 n \leadsto k = \log_2 n$$

La complessità della ricerca binaria, è $\log_2 n$.

Se non è una potenza di 2, $n = 23$, allora $\overset{16}{\overset{\shortparallel}{2^4}} < 23 < \overset{32}{\overset{\shortparallel}{2^5}}$,
per eseguire l'algoritmo ci metto un numero di passi compreso tra 4 e 5,
$\lfloor \log_2 23 \rfloor$ e $\lceil \log_2 23 \rceil$. L'ordine di grandezza è sempre $\log_2 n$.

$n \simeq 4$ miliardi $\quad \log_2 n = 32$

Selection sort ha complessità $\Theta(n^2)$. Posso fare di meglio?

Per alcuni problemi si può stabilire una limitazione INFERIORE alla complessità di tutti gli algoritmi che lo risolvono.

Alcune limitazioni sono ovvie: se opero su un array lungo n e per dare il risultato devo necessariamente guardare tutti gli elementi dell'array, allora qualunque algoritmo che risolve il problema ha complessità $\Omega(n)$.

n o superiore.

Es: determinare il minimo di un array lungo n che non è ordinato.

Per essere sicuro di trovare il minimo devo per forza guardare tutti gli elementi, quindi la complessità del problema è $\Omega(n)$

Es: ordinamento.

Come sopra, devo guardare almeno una volta tutti gli elementi

La complessità minima dell'ordinamento è $\Omega(n)$

GAP

Il selection sort è $\Theta(n^2)$

QUICK SORT $\Theta(n \lg n)$ CASO MEDIO

MERGE SORT $\Theta(n \lg n)$ CASO PESSIMO

COUNTING SORT $\Theta(n)$ con ipotesi aggiuntive molto stringenti e enorme occupazione di memoria.