

# Algoritmi e Strutture Dati

a.a. 2023/24

## Compito del 16/01/2025

Cognome: \_\_\_\_\_

Nome: \_\_\_\_\_

Matricola: \_\_\_\_\_

E-mail: \_\_\_\_\_

### Parte I

(30 minuti; ogni esercizio vale 2 punti)

**Avvertenza:** Si giustifichino tecnicamente tutte le risposte. In caso di discussioni poco formali o approssimative gli esercizi non verranno valutati pienamente.

1. Scrivere l'algoritmo build-Max-Heap e simulare la sua esecuzione sull'array  $\langle -37, -9, 15, 8, 70 \rangle$

$$T(m) = 9T(m/3) + 7m + \log m$$

dove  $m$  rappresenta il numero di archi del grafo. Esistono algoritmi più efficienti di MyMST per risolvere il problema dato?

3. Sia  $G = (V, E)$  un grafo non orientato e siano  $u$  e  $v$  due vertici collegati da un cammino  $p$ . Si stabilisca se le seguenti affermazioni sono vere o false, giustificando tecnicamente la risposta:
  - a. "Se  $G$  è aciclico, allora non esiste un altro cammino tra  $u$  e  $v$ , oltre  $p$ ."
  - b. "Se  $G$  è connesso, allora esistono necessariamente altri cammini tra  $u$  e  $v$ , oltre  $p$ ."
  - c. "Se  $G$  è un albero, allora il numero di cammini tra  $u$  e  $v$  dipende dal numero di archi in  $G$ ."

# Algoritmi e Strutture Dati

a.a. 2023/24

Compito del 16/01/2025

Cognome: \_\_\_\_\_

Nome: \_\_\_\_\_

Matricola: \_\_\_\_\_

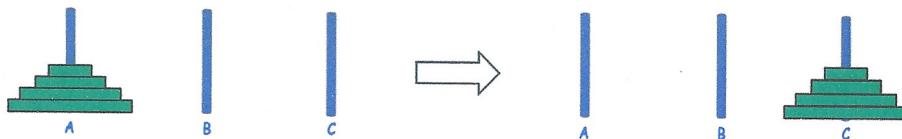
E-mail: \_\_\_\_\_

## Parte II

(2.5 ore; ogni esercizio vale 6 punti)

**Avvertenza:** Si giustifichino tecnicamente tutte le risposte. In caso di discussioni poco formali o approssimative gli esercizi non verranno valutati pienamente.

1. Dato un albero binario i cui nodi  $x$  hanno i campi **left**, **right** e **key**, dove **key** è un numero intero:
  - a. definire l'**altezza** di un nodo  $x$ ;
  - b. scrivere una funzione **efficiente** in C o C++ che ritorna il numero di nodi per i quali la chiave  $x \rightarrow \text{key}$  è minore o uguale dell'altezza del nodo. Il prototipo della funzione è:  
`int lessHeight(PNode r)`
  - c. valutare la complessità della funzione, indicando eventuali relazioni di ricorrenza e mostrando la loro risoluzione;
  - d. specificare il linguaggio di programmazione scelto.
2. Per ordinare l'array  $A[1..n]$ , si ordina in modo ricorsivo il sottoarray  $A[1.. n-1]$  e poi si inserisce  $A[n]$  nel sottoarray ordinato  $A[1.. n-1]$ .
  - a. Scrivere lo pseudocodice per questa variante **ricorsiva** dell'insertion sort.
  - b. Fornire una ricorrenza per il suo tempo di esecuzione nel caso peggiore e risolverla in modo formale.
  - c. Quale è il tempo di esecuzione nel caso migliore? Mostrare un esempio di input che determina il caso migliore.
3. Il problema della *torre di Hanoi* è un classico rompicapo matematico nel quale sono dati tre paletti e un certo numero di dischi di grandezza diversa. Il gioco inizia ponendo i dischi su un paletto in ordine decrescente, in modo da formare un cono. Lo scopo del gioco è portare tutti i dischi su un paletto diverso potendo spostare solo un disco alla volta e potendo mettere un disco solo su un altro più grande, mai su uno più piccolo. Nella configurazione riportata in figura il numero di dischi è 4, il paletto di partenza è A e il paletto di arrivo è C. Il paletto B viene utilizzato come paletto di "appoggio".



Il seguente algoritmo rappresenta una soluzione ricorsiva al problema:

```

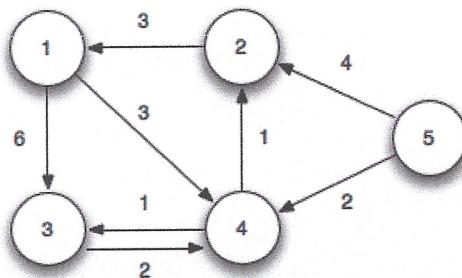
procedure hanoi (n: integer; partenza, appoggio, arrivo: palo);
begin
  if (n = 1) then muoviDisco(partenza, arrivo)
  else begin
    hanoi(n-1, partenza, arrivo, appoggio);
    muoviDisco(partenza, arrivo);
    hanoi(n-1, appoggio, partenza, arrivo);
  end
end

```

dove muoviDisco(partenza, arrivo) è una semplice procedura che sposta il disco situato in cima al palo di partenza su quello di arrivo e ha complessità costante.

Qual è la complessità dell'algoritmo? Giustificare formalmente la risposta.

4. Si scriva l'algoritmo di Dijkstra, si derivi la sua complessità, si dimostri la sua correttezza e si simuli la sua esecuzione sul seguente grafo utilizzando il vertice 1 come sorgente:



In particolare:

- si indichi l'ordine con cui vengono estratti i vertici
- si riempia la tabella seguente con i valori dei vettori  $d$  e  $\pi$ , iterazione per iterazione:

	vertice 1		vertice 2		vertice 3		vertice 4		vertice 5	
	$d[1]$	$\pi[1]$	$d[2]$	$\pi[2]$	$d[3]$	$\pi[3]$	$d[4]$	$\pi[4]$	$d[5]$	$\pi[5]$
dopo inizializzazione	0	NIL	$\infty$	NIL	$\infty$	NIL	$\infty$	NIL	$\infty$	NIL
1 iterazione 1	0	NIL	$\infty$	NIL	6	1	3	1	$\infty$	NIL
4 iterazione 2	0	NIL	4	4	4	4	3	1	$\infty$	NIL
2 iterazione 3	0	NIL	4	4	4	4	3	1	$\infty$	NIL
3 iterazione 4	0	NIL	4	4	4	4	3	1	00	NIL
5 iterazione 5	0	NIL	4	4	4	4	3	1	00	NIL

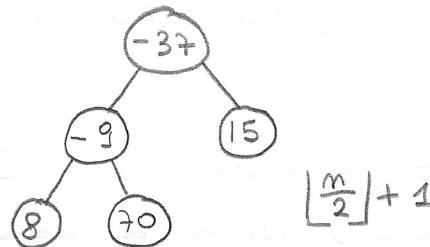
## PART I

## 1) build max heap (Array A)

$A.\text{heapsize} = A.\text{length}$

For  $i = \lfloor A.\text{heapsize}/2 \rfloor$  DOWN TO 1  
 $\text{max-heapify}(A, i) \rightsquigarrow O(h)$

-37	-9	15	8	70
1	2	3	4	5



$$\lfloor \frac{m}{2} \rfloor + 1$$

$i=2$	-37	-9	15	8	70
	1	2	3	4	5
	↑				

$i=1$	-37	70	15	8	-9
	1	2	3	4	5
	↑				

$i=0$	70	-37	15	8	-9
	1	2	3	4	5

$$\Rightarrow [70 \ 8 \ 15 \ -37 \ -9]$$

$$T(m) = O(m)$$

SE SCRIVEVO LA MAX-HEAPIFY MA NON BUILDMAXHEAP DAVA LO STESSO 2 PUNTI SE C'ERA TEMPO SCRIVERLO

$$2) T(m) = 9T\left(\frac{m}{3}\right) + 7m + \log m$$

$$a = 9 \quad b = 3 \quad d = \log_3 9 = 2 \quad g(m) = m^2 \quad f(m) = 7m + \log m \quad \underset{=}^{|\ } \text{DOMINA } 7m$$

SIAMO NEL 1° CASO DEL TEOREMA MASTER

$$f(m) \leq g(m) \quad f(m) = O(m^{d-\varepsilon})$$

$$7m = O(m^{2-\varepsilon}) \Rightarrow \varepsilon = 1$$

$$T(m) = \Theta(m^2)$$

Gli algoritmi più noti per trovare un MST in un grafo pesato:

- ALGORITMO DI KRUSCAL :  $T(KRUSCAL) = m \log m$

- ALGORITMO DI PRIM :  $T(PRIM) = m \log n$

Dato che la complessità di My MST è  $\Theta(m^2)$ , che è superiore a quella degli algoritmi di Kruskal e Prim, possiamo concludere che esistono algoritmi più efficienti di My MST per risolvere il problema dell'albero di copertura minima in un grafo pesato.

3) a) VERO

Un grafo aciclico è un grafo che non contiene cicli. In un grafo aciclico, tra due vertici  $u$  e  $v$  collegati da un cammino  $p$ , non può esistere un altro cammino diverso da  $p$  che colleghi  $u$  e  $v$ . Se esistesse un altro cammino  $p'$  tra  $u$  e  $v$ , allora l'unione di  $p$  e  $p'$  formerebbe un ciclo, contraddicendo l'ipotesi che il grafo sia aciclico.

b) FALSO

Un grafo连通的 garantisce che esiste almeno un cammino tra qualsiasi coppia di vertici  $u$  e  $v$ . Tuttavia, non è necessariamente vero che esistono altri cammini oltre a  $p$ . Ad esempio, in un albero (che è un grafo连通的 e aciclico), tra due vertici  $u$  e  $v$  esiste esattamente un unico cammino. Pertanto, l'esistenza di altri cammini non è garantita solo dalla connessione del grafo.

c) FALSO

In un albero, che è un grafo連通的 e aciclico, tra due vertici  $u$  e  $v$  esiste esattamente un unico cammino. Questo cammino è determinato unicamente dalla struttura dell'albero e non dipende dal numero totale di archi in  $G$ . Pertanto, il numero di cammini tra  $u$  e  $v$  in un albero è sempre 1, indipendentemente dal numero di archi nel grafo.

## PART II

- 1) a) L'altezza di un nodo  $x$  è la lunghezza del cammino più lungo da  $x$  a una foglia  
 b) int lessHeight (PNode r) {

```
int h;
return lessHeightAux(r, h);
}
```

```
int lessHeightAux (PNode r, int & h) {
```

```
int rissx, risdx, hsx, hdx;
if (r == nullptr) {
```

```
h = -1;
```

```
return 0;
```

```
}
```

```
rissx = lessHeightAux (r->left, hsx);
```

```
risdx = lessHeightAux (r->right, hdx);
```

```
h = (hsx <= hdx ? hdx : hsx) + 1;
```

```
if (r->key <= h)
```

```
return 1 + rissx + risdx;
```

```
else
```

```
return rissx + risdx;
```

```
}
```

$$T(m) = \begin{cases} c & m=0 \\ T(k) + T(m-k-1) + d & m>0 \end{cases}$$

$k$  è il numero di nodi  
nel sottoalbero sinistro

IDEA:  $T(n) = \alpha n + b \rightarrow$  È LINEARE, PERCHE' OGNI NODO VIENE VISTO UNA SOLA VOLTA

Voglio dimostrare che  $\forall m \ T(m) = \alpha m + b$

Per induzione completa

CASO BASE

$$(m=0) \quad T(m) = c \quad \text{ma allora} \quad T(0) = \alpha 0 + b = b \\ \text{DEFINIZIONE} \qquad \qquad \qquad \text{Dunque } b = c$$

CASO INDUTTIVO

$$(m > 0) \text{ Assumo che } \forall m < n \quad T(m) = \alpha m + b$$

Voglio dimostrare che vale per  $n$  cioè  $T(n) = \alpha n + b$

$$T(n) = T(k) + T(n-k-1) + d$$

PER DEFINIZIONE

$$= \cancel{\alpha k + b} + \cancel{\alpha(n-k-1)} + b + d = \alpha n - \alpha + 2b + d$$

PER IPOTESI INDUTTIVA

$$k < n$$

$$n-k-1 < n$$

$$\cancel{\alpha n - \alpha + 2b + d} = \cancel{\alpha n + b} \quad \text{VERO QUANDO } \alpha = b + d = c + d$$

$$\text{Dunque } T(n) = (c+d)n + c = \Theta(n)$$

ESAME 16/01/25

③

2) /\*PRE: A[...n]\*/

insertionSort (Array A,int dim) {

if (dim&gt;1) {

insertionsort (A,dim-1);

val = A[n];

j = m-1;

while (j&gt;0 AND A[j]&gt;val) {

A[j+1] = A[j];

j = j-1;

}

A[j+1] = val;

}

}

ORDINA QUESTI ELEMENTI

1	-7	3	0
1	2	3	4

$$T(n) = \begin{cases} c & n \leq 1 \\ T(n-1) + O(n) & n > 1 \end{cases}$$

CASO PEGGIORE

$$T(n) = T(n-1) + d n$$

$$= T(n-2) + d(n-1) + d n$$

$$= T(n-3) + d(n-2) + d(n-1) + d n$$

⋮

$$= T(1) + \sum_{i=1}^n d i = c + d \sum_{i=1}^n i = c + \frac{d(n(n+1))}{2} = \Theta(n^2)$$

$$\downarrow \\ m-k=1 \rightarrow k=m-1$$

IL CASO PIÙ GUORE È QUANDO NON SI ENTRA MAI NEL WHILE

Input ordinato in senso non decrescente

$$T(n) = O(n)$$

Perche' non entra mai nel ciclo while

$$T(n) = T(n-1) + d \quad \text{RICORRENZA SE NON ENTRA MAI NEL CICLO WHILE}$$

3) CASO BASE: Quando  $m=1$ , l'algoritmo esegue una sola operazione, ovvero la chiamata a muovi Disco (partenza, arrivo), che ha complessità costante  $O(1)$

CASO RICORSIVO: Per  $m > 1$ , l'algoritmo esegue:

- una chiamata ricorsiva  $\text{hanoi}(m-1, \text{partenza}, \text{arrivo}, \text{appoggio})$

- una chiamata a muovi Disco (partenza, arrivo)

- un'altra chiamata ricorsiva  $\text{hanoi}(m-1, \text{appoggio}, \text{partenza}, \text{arrivo})$

La relazione di ricorrenza che descrive la complessità temporale  $T(m)$  dell'algoritmo è quindi:

$$T(m) = 2T(m-1) + O(1)$$

Per risolvere questa relazione di ricorrenza, possiamo utilizzare il metodo dell'albero di ricorsione o il metodo di sostituzione.

$$T(m) \\ 2T(m-1) + \frac{c}{O(1)}$$

$$4T(m-2) + 2c$$

$$\dots \\ 2^k T(m-k) + \sum_{k=0}^{m-1} 2^k c$$

Il processo continua fino a quando  $m-k=1$ , ovvero  $k=m-1$ . Quindi l'ultimo livello sarà:

$$\text{LIVELLO } m-1 : 2^{m-1} T(1)$$

$$\text{La somma della serie geometrica è: } \sum_{k=0}^{m-1} 2^k = 2^m - 1$$

$$\text{Quindi, la complessità totale è: } T(m) = O(2^m)$$

#### CONCLUSIONE

La complessità temporale dell'algoritmo ricorsivo per risolvere il problema della Torre di Hanoi è  $O(2^m)$ . Questo significa che il numero di mosse necessarie per risolvere il problema cresce esponenzialmente con il numero di dischi  $m$ .

4) Dijkstra ( $G, \omega, s$ )INIT-SS ( $G, s$ ) $Q \leftarrow V[G]$  $S \leftarrow \emptyset$ while  $Q \neq \emptyset$  $u \leftarrow \text{EXTRACTMIN}(Q)$  $S \leftarrow S \cup \{u\}$ for each  $v \in \text{Adj}[u]$ RELAX ( $u, v, \omega(u, v)$ )return ( $d, G_T$ )INIT-SS ( $G, s$ )for each  $u \in V[G]$  $d[u] = +\infty$  $\pi[u] = \text{NIL}$  $d[s] = 0$ RELAX ( $u, v, \omega(u, v)$ )if  $d[v] > d[u] + \omega(u, v)$  then $d[v] = d[u] + \omega(u, v)$  $\pi[v] = u$ 

T(DISKSTRA) :

• HEAP  $O(m \log m)$ - GRAFO SPARSO:  $m \approx n = O(m \log m)$ - GRAFO DENSO:  $m \approx n^2 = O(n^2 \log n)$ • ARRAY  $O(m \cdot m)$ - GRAFO SPARSO:  $m \approx n = O(m^2)$ - GRAFO DENSO:  $m \approx n^2 = O(n^2)$ 

CORRETEZZA

Sia  $G = (V, E)$  un grafo orientato pesato sugli archi, cioè con  $\omega: E \rightarrow \mathbb{R}$  tale che  $\forall (u, v) \in E: \omega(u, v) \geq 0$ . Allora, alla fine dell'algoritmo di Dijkstra, si ha:

1)  $\forall v \in V: d[v] = \delta(s, v)$ 2)  $G_T$  è un albero di cammini minimi.

Dimostreremo la correttezza del primo punto attraverso la dimostrazione di un'affermazione più forte ma equivalente, cioè che per ogni  $u \in V$ , al momento dell'esecuzione di  $u$ , risulta  $d[u] = \delta(s, u)$ .

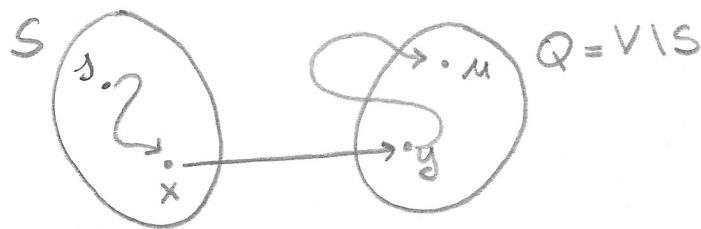
La dimostrazione avviene per assurdo e si avvale di 9 osservazioni.

## DIMOSTRAZIONE

Supponiamo per assurdo che esista un vertice  $u \in V$  tale che al momento della sua estrazione  $d[u] \neq S(s, u)$ , e che  $u$  sia il primo vertice per cui questo accade.

### OSSERVAZIONI

- 1)  $u$  non può essere la sorgente: dopo la INIT-SS,  $d[s] = 0 = S(s, s)$ , e  $S(s, s)$  non può valere  $-\infty$  perché per ipotesi non ci sono archi con pesi negativi, quindi neanche cicli negativi.
- 2) Al momento dell'estrazione di  $u$ ,  $S \neq \emptyset$ , perché in  $S$  ci sarà almeno la sorgente  $s$ .
- 3)  $u$  non è irraggiungibile dalla sorgente: se così fosse,  $S(s, u) = +\infty = d[u]$ , che sarebbe corretto e non violerebbe la proprietà che abbiamo voluto violare per assurdo; quindi  $S(s, u) \neq +\infty$
- 4) Ci poniamo nell'istante in cui  $s$  è già stato estratto ( $s \in S$ ) ma  $u$  no ( $u \notin Q = V \setminus S$ ). Per il punto precedente, esiste un cammino minimo  $p$  tra  $s$  e  $u$ : sia  $(x, y)$  un arco appartenente a  $p$  che attraversa il taglio, cioè tale che  $x \in S$  e  $y \in Q$



- 5) Per ipotesi vale che  $d[x] = S(s, x)$ : infatti,  $u$  è il primo vertice per cui questa proprietà non vale.
- 6) Poiché siamo su un cammino minimo, possiamo applicare la proprietà della convergenza: al momento dell'estrazione di  $x$ , applichiamo la RELAX su  $y$  e otteniamo che  $d[y] = S(s, x) + w(x, y) = S(s, y)$

## CONTINUA DEMOSTRAZIONE DIJKSTRA

- 7) Dal momento che stiamo per estrarre il nodo  $u$ , siccome l'algoritmo di Dijkstra estrae il vertice avente campo d più piccolo, dovrà valere che  $d[u] \leq d[y]$
- 8) Non può accadere che  $S(y, y) > S(s, u)$ : in virtù del fatto che ci troviamo in un cammino minimo e che i pesi sono tutti maggiori o uguali a zero, dunque  $S(s, y) \leq S(s, u)$ .
- 9) Per la proprietà del limite inferiore, vale sicuramente che  $S(s, u) \leq d[u]$ .

Ricostriamo l'assurdo sulla base delle precedenti osservazioni:

$$\begin{aligned} S(s, u) &\leq d[u] && \text{PER OSS. 9} \\ &\leq d[y] && \text{PER OSS. 7} \\ &= S(s, y) && \text{PER OSS. 6} \\ &\leq S(s, u) && \text{PER OSS. 8} \end{aligned}$$

Avvalendoci delle osservazioni, siamo riusciti a "rinchiudere" il valore di  $d[u]$  tra  $S(s, u)$  e  $S(s, u)$ :  $S(s, u) \leq d[u] \leq S(s, u) \Rightarrow d[u] = S(s, u)$

Che è assurdo in quanto andiamo ad invalidare l'ipotesi che  $d[u] \neq S(s, u)$ .

build max heap (array A)

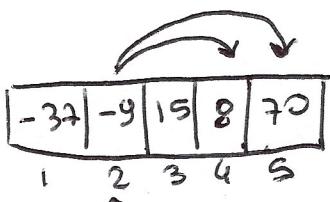
PART I ES 1

1	2	3	4	5
-37	-9	15	8	70

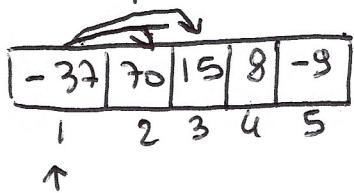
$$A.\text{heapsize} = A.\text{length}$$

for  $i = \lfloor \frac{n}{2} \rfloor$  down to 1

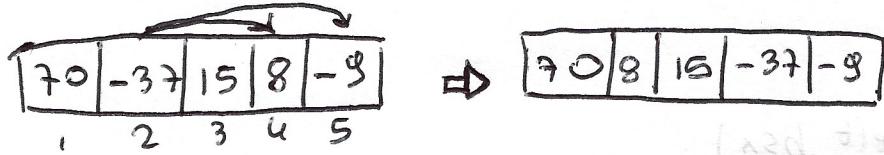
max-heapify ( $A, i$ )  $\rightsquigarrow O(h)$



$i=2$



$i=1$



$$T(n) = O(n)$$

SE SCRIVONO LA MAX-HEAPIFY MA NON ~~BUILD MAX HEAP~~ BUILD MAX HEAP DAVOLLO STESO 2 PUNTI  
SE C'ERA TEMPO SCRIVERLO

$$O(n) + b + \left( -\frac{1}{2}n + n \right) T + \left( \frac{1}{2}n \right) T \quad \left\{ \begin{matrix} O(n) \\ b \\ -\frac{1}{2}n + n \\ \frac{1}{2}n \end{matrix} \right\} (n) T$$

### PART III ES 1

L'altezza di un nodo  $x$  è la lunghezza del cammino più lungo da  $x$  a una foglia.

`int lessHeight (PNode r)`

`int h;`

`return lessHeight Aux(r, h);`

}

`int lessHeightAux (PNode r, int & h){`

`int risultosx, risdx, hsx, hdx;`

`if (r == nullptr) {`

`h = -1;`

`return 0;`

}

`rissx = lessHeightAux (r->left, hsx);`

`risdx = lessHeightAux (r->right, hdx);`

`h = (hsx <= hdx ? hdx : hsx) + 1;`

`if (r->key <= h)`

`return 1 + rissx + risdx;`

`else`

`return rissx + risdx;`

}

$$T(n) \begin{cases} C & n=0 \\ T(k) + T(n-k-1) + d & n>0 \end{cases}$$

$k$  è il numero di nodi nel sottoalbero sinistro

(2)

S23 E72019

$$T(n) \begin{cases} c & n=0 \\ T(k) + T(n-k-1) + d & n>0 \end{cases}$$

Idea:  $T(n) = \alpha n + b \rightarrow$  È LINEARE, PERCHÉ' COME NOI VIGE UNA CVA SOCAVOLTA

Voglio dimostrare che  $T(n) = \alpha n + b$

Per induzione completa

CASO BASE

$$(n=0) T(0) = c \text{ ma allora } T(0) = \alpha 0 + b = b$$

$\uparrow$   
Definizione

$$\text{Dunque } b = c$$

CASO INDUTTIVO

$$(n>0) \text{ Assumo che } T(m) = \alpha m + b \quad \Delta \leq m$$

Voglio dimostrare che vale per  $n$  cioè  $T(n) = \alpha n + b$

$$T(n) = T(k) + T(n-k-1) + d$$

$\uparrow$   
per definizione

$$= \alpha k + b + \alpha(n-k-1) + b + d = \alpha n - \alpha + 2b + d$$

$$\alpha k + ((n-k)-1)b + (n-k)d = \alpha n - \alpha + (n-k)b + (n-k)d =$$

PER IPOTESI INDUTTIVA

$k \leq m$

$$m-k-1 < m \Rightarrow S = \underbrace{\alpha k + b}_{\alpha k + ((n-k)-1)b + (n-k)d} + \dots + \underbrace{\alpha b + d}_{\alpha n - \alpha + (n-k)b + (n-k)d} = \alpha n - \alpha + (n-k)b + (n-k)d =$$

$$\alpha n - \alpha + nb + d = \alpha n + b \quad \text{VERO QUANDO } \alpha = b + d = c + d$$

$$\text{Dunque } T(n) = (c+d)n + c = \Theta(n)$$

(n) O = (n)T

dove  $c, d, n$  sono costanti positivedimostrare che  $c, d, n$  sono costanti positive  $b + ((n-k)-1)b + (n-k)d = (n)T$

## PART II ES 2

/\* PRE: A[1...n] \*/

insertionsort(Aray A,int dim){

if (dim>1)

insertionsort(A, dim-1);

val = A[n]

j = m-1

while j>0 AND A[j]>val

A[j+1] = A[j];

j=j-1;

A[j+1] = val

ORDINA QUASI ELETTRONI

1	-7	3	0
1	2	3	4

$$T(n) = \begin{cases} C & n \leq 1 \\ T(n-1) + O(n) & n > 1 \end{cases}$$

CASO PEGGIORIO

$$T(n) = T(n-1) + d \cdot n$$

$$= T(n-2) + d(n-1) + d \cdot n$$

$$= T(n-3) + d(n-2) + d(n-1) + d \cdot n$$

:

$$= T(1) + \sum_{i=1}^n d \cdot i = c + d \sum_{i=1}^n i = c + \frac{d(n(n+1))}{2} = \Theta(n^2)$$

$$n-k=1 \Rightarrow k=n-1$$

IL CASO MIGLIORIO E' QUANDO NON SI ENTRA MAI NEL WHILE  $n(b+c) = (n)b + n(c)$   
Input ordinato in senso non decrescente.

$$T(n) = O(n)$$

Perche' non entra mai nel ciclo while

$$T(n) = T(n-1) + d \quad \text{RICORRENZA SE NON ENTRA MAI NEL CYCLE WHILE}$$