Algoritmi

Definizione

Metodo risolutivo che collega ogni istanza alla soluzione.

Le **istanze** di un problema sono di solito **infinite**

- Ogni istanza ha una **lunghezza** n e una **taglia** $|n|$ (numero di cifre parte intera inferiore)
- Le istanze di una certa lunghezza sono in genere infinite

Definizione: Costo Computazionale

Definiamo **costo computazionale** di un algoritmo una funzione $T(n)$ che restituisce le risorse usate in funzione della lunghezza dell'istanza

Definizione: Algoritmo Ottimo e Complessità Computazionale

- **Algoritmo Ottimo**: Algoritmo con costo computazionale migliore tra i possibili per un problema
- **Complessità Computazionale**: Costo computazionale matematicamente migliore per la risoluzione di un problema, anche definita **lower bound**

Bisogna notare che la **complessità computazionale è riferita ad un problema** e non un algoritmo, difatti è possibile che nonostante si sappia quale sia matematicamente il costo computazionale migliore per la risoluzione di un problema, non è ancora stato scoperto un algoritmo che lo raggiunga

Definizione: Problema Indecidibile

Definiamo problema indecidibile un **algoritmo irrisolvibile**

Esempio: Terminatore di Touring

Creare un algoritmo che, preso un altro algoritmo, dica se quest'ultimo finisce

Strutture Dati

Definizione

Strutture identificate da un particolare metodo di **salvataggio delle informazioni** e dalle **operazioni su di esse**, che sono a loro volta algoritmi.

Tipologie

- **Concrete**: strutture di base che è possibile dare per scontate (es. vettori)
- **Astratte**: strutture complesse create utilizzando diverse strutture concrete ed eventualmente altre astratte

Tipi di Operazioni

- **Statiche**: leggono i dati senza modificarne la struttura o il contenuto
- **Dinamiche**: modificano i dati e necessitano di una computazione per il mantenimento della struttura

Dimensione dei Problemi

Consideriamo la seguente tabella sui **numeri di cifre** di alcune funzioni note

n	$\lvert\log_2(n)\rvert$	$\lvert n\rvert$	$\lvert n^2\rvert$	$\lvert n^5\rvert$	$\lvert 2^n\rvert$	$\lvert n^n\rvert$
1	1	1	1	1	1	1
100.000	2	6	11	26	30.103	500.001
200.000	2	6	11	27	60.206	1.060.206
300.000	2	6	11	28	90.309	1.643.137
400.000	2	6	12	29	120.412	2.240.824
500.000	2	6	12	29	150.515	2.849.486
600.000	2	6	12	29	180.618	3.466.891
700.000	2	6	12	30	210.721	4.091.569
800.000	2	6	12	30	240.824	4.722.472
900.000	2	6	12	30	270.927	5.358.819
1.000.000	2	7	13	31	301.030	6.000.001

Possiamo notare come da $\log a\ n$ e da n a 2^n è presente lo stesso "salto" di grandezza dei valori, in quanto si passa di un livello esponenziale

Sia

- A un algoritmo
- n la lunghezza delle istanze da risolvere
- $f(n)$ il numero di operazioni da eseguire di A in funzione di n
- a il numero di operazioni al secondo eseguite

Possiamo ottenere l'istanza più lunga risolvibile in k secondi come

$$\frac{f(n)}{a} \leq k$$

Da questa formula possiamo notare l'immensa **differenza di costo computazionale** tra le varie funzioni

$$a = 1$$

k	$\log_2(n)$	n	n^2	2^n
1 secondo	2	1	1	0
2 secondi	4	2	1	1
4 secondi	16	4	2	2
1 minuto	$1.15 \cdot 10^{18}$	60	7	6
1 ora	*	3.600	60	12
1 giorno		86.400	293	17
1 secolo		$3.1536 \cdot 10^9$	56.156	32

** numero con circa 1084 cifre*

O con un caso più realistico, una istruzione per microsecondo

$$a = 10^{11}$$

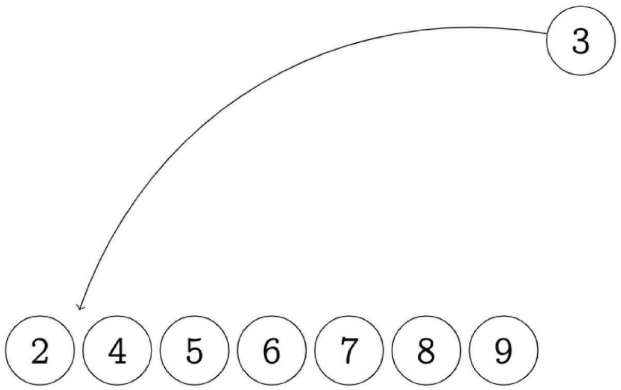
k	$\log_2(n)$	n	n^2	2^n
1 secondo	2	1	1	0
2 secondi	4	2	1	1
4 secondi	16	4	2	2
1 minuto	$1.15 \cdot 10^{18}$	60	7	6
1 ora	*	3.600	60	12
1 giorno		86.400	293	17
1 secolo		$3.1536 \cdot 10^9$	56.156	32

** numero con circa 30'000'000'000 di cifre*

Insertion Sort

Idea

L'idea è quella di, per ogni elemento, controllare dal fondo ogni elemento fino a trovarne uno minore, per poi inserirlo dopo di esso

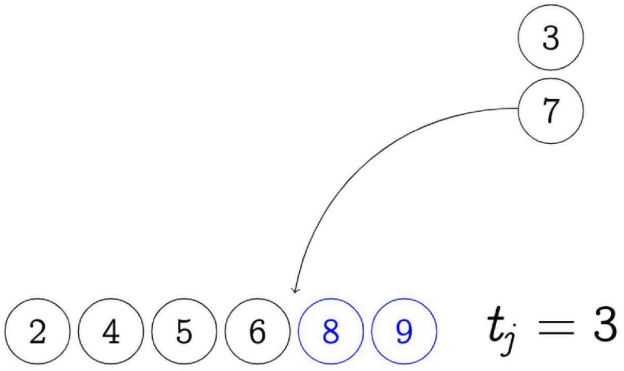


INSERTION-SORT(A)

```
1  for j = 2 to A.length
2          key = A[j]
3          // Insert A[j] into the sorted sequence A[1..j-1]
4          i = j - 1
5          while i > 0 and A[i] > key
6                  A[i + 1] = A[i]
7                  i = i - 1
8          A[i + 1] = key
```

Parametri: *A=vettore*

Chiamiamo t_j il numero di volte che la **guardia del ciclo while** (condizione - riga 5) viene valutata.
Il valore di t_j dipende da j e corrisponde al numero di elementi del vettore ogni volta spostati, più uno (condizione risulta false e esce dal ciclo)



Costo Computazionale

Calcoliamo il costo computazionale dell'algoritmo

Istruzione	Costo	Ripetizioni
`for j = 2 to A.length`	c_1	n
`key = A[j]`	c_2	$n-1$
`i = j - 1`	c_4	$n-1$
`while i > 0 and A[i] > key`	c_5	$\sum_{j=2}^{n} t_j$
`A[i + 1] = A[i]`	c_6	$\sum_{j=2}^{n}(t_j-1)$
`i = i - 1`	c_7	$\sum_{j=2}^{n}(t_j-1)$
`A[i + 1] = key`	c_8	$n-1$

La formula sarà quindi

$$
\begin{aligned}
T(n) = {} & c_1 \cdot n + \\
& c_2 \cdot (n-1) + \\
& c_4 \cdot (n-1) + \\
& c_5 \cdot \sum_{j=2}^{n} t_j + \\
& c_6 \cdot \sum_{j=2}^{n} (t_j - 1) + \\
& c_7 \cdot \sum_{j=2}^{n} (t_j - 1) + \\
& c_8 \cdot (n-1)
\end{aligned}
$$

A seconda del valore di t_j, abbiamo risultati diversi

- **Caso ottimo**: non sposto mai nessuno $\implies t_j = 1$
- **Caso pessimo**: li sposto **tutti** ogni volta $\implies t_j = j$
- **Caso medio**: ne sposto mediamente **metà** ogni volta $\implies t_j = \frac{j}{2}$

Date le seguenti **sommatorie note** calcoliamo

$$\sum_{j=1}^{n} j = \frac{n(n+1)}{2}$$
$$\sum_{j=2}^{n} j = \frac{n(n+1)}{2} - 1$$
$$\sum_{j=2}^{n} (j-1) = \frac{n(n-1)}{2}$$

Esempio: Caso Ottimo

$$t_j = 1$$

$$c_1 n + c_2(n-1) + c_4(n-1) + c_5 \sum_{j=2}^{n} 1 + c_6 \sum_{j=2}^{n} (1-1) + \sum_{j=2}^{n} (1-1) + c_8(n-1)$$

Semplificando e raccogliendo le costanti

$$T(n) = an + b$$

Esempio: Caso Pessimo

$$t_j = j$$

$$c_1 n + c_2(n-1) + c_4(n-1) + c_5 \sum_{j=2}^{n} j + c_6 \sum_{j=2}^{n} (j-1) + \sum_{j=2}^{n} (j-1) + c_8(n-1)$$

Semplificando e raccogliendo le costanti

$$T(n) = an^2 + bn + c$$

Esempio: Caso Medio

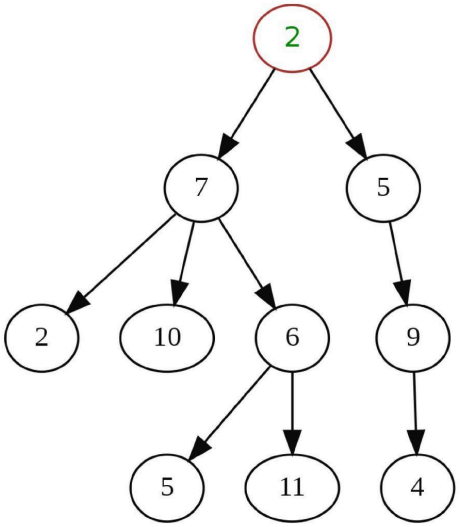
$$t_j = \frac{j}{2}$$

$$c_1 n + c_2(n-1) + c_4(n-1) + c_5 \sum_{j=2}^{n} \frac{j}{2} + c_6 \sum_{j=2}^{n} \left(\frac{j}{2} - 1\right) + \sum_{\frac{j}{2}=2}^{n} \left(\frac{j}{2} - 1\right) + c_8(n-1)$$

Semplificando e raccogliendo le costanti

$$T(n) = an^2 + bn + c$$

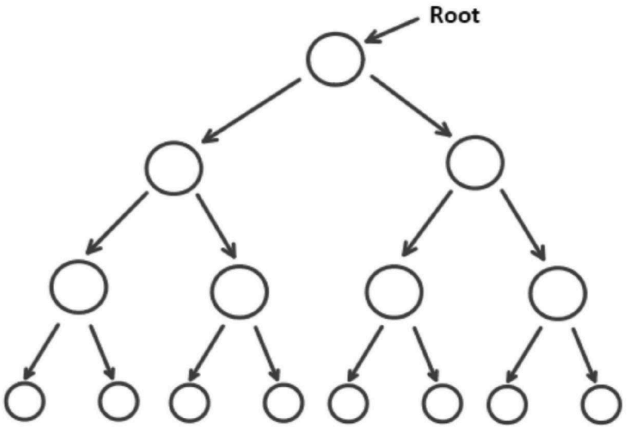
Alberi



Proprietà

- Ogni nodo ha **un solo padre e diversi figli**, ad eccezione del primo che non ne ha, chiamato **radice (root)**
- Due nodi possono essere collegati da **un solo percorso**, ovvero **non sono presenti cicli**
- I nodi senza figli vengono chiamati **foglie**

Alberi Binari



Proprietà

- Ogni nodo a **massimo due figli**
- Viene definito **completo** se il numero di nodi è una potenza di 2

Piano (i)	**Numero di Nodi** (LIV_i)	**Totale Nodi** (TOT_i)
1	1	1
2	2	3
3	4	7
4	8	15
h	2^{h-1}	$2^h - 1$