Algoritmi e Strutture Dati

Modelli di Calcolo e Analisi di Algoritmi

Maria Rita Di Berardini, Emanuela Merelli[1]

[1]Dipartimento di Matematica e Informatica
Università di Camerino

6 ottobre 2009

Parte I

Il concetto di algoritmo

Algoritmo - alcune definizioni

Babilon

(Mathematics) step-by-step procedure used to solve a problem (often includes repetition of steps);
(Computers) step-by-step problem-solving procedure used within software applications

NIST

A computable set of steps to achieve a desired result.

Wikipedia

Un procedimento che consente di ottenere un risultato atteso eseguendo, in un determinato ordine, un insieme di passi semplici corrispondenti ad azioni scelte solitamente da un insieme finito.

Il concetto di Algoritmo

Definizione (Informale)

Un algoritmo *è un procedimento formato da una sequenza finita di operazioni elementari (o passi) che ci consentono di risolvere un dato problema computazionale.*

- Ogni problema computazionale è caratterizzato dai dati di cui si dispone all'inizio (dati in ingresso, o **valori in input**) e dai risultati che si vogliono ottenere (**valori in output**)
- Risolve un problema computazionale significa fornire un algoritmo che consente di ottenere i risultati attesi a partire da un certo insieme di dati in input

Il concetto di Algoritmo

Nel senso più ampio del termine, un “algoritmo è anche:

- una ricetta di cucina
- le istruzioni per il montaggio di un modellino
- sezione del libretto delle istruzioni di una lavatrice che spiega come programmare un lavaggio

Di norma la parola viene usata in contesti matematici (fin dalle origini) e soprattutto informatici (più recentemente).

Algoritmo per la determinazione del minimo

Problema (Minimo di un array)

$min(A[1 \cdots n]) = a$ *se e solo se* $a \leq A[i]$ *per ogni* $i = 1, \ldots, n$

Algoritmo

```
MIN(A)
1  a ← A[1]
2  for i ← 2 to n
3  do if a > A[i]
4        then a ← A[i]
5  return a
```

```
MIN(A)
1  return A[1]
```

Algoritmo Euclideo per il calcolo del MCD

Definition (Massimo comun divisore - MCD)

Siano a e b due interi non entrambi pari a zero. Il $MCD(a, b)$ tra a e b è il più grande numero naturale che divide sia a che b.

Proprietà

- *se* $a = b$ *allora* $MCD(a, b) = a$
- *se* $a > b$ *allora* $MCD(a, b) = MCD(a - b, b)$
- *se* $a < b$ *allora* $MCD(a, b) = MCD(a, b - a)$

Algoritmo

1 *finchè* $a \neq b$ *ripeti*

  1.1 *se* $a > b$ *poni* $a = a - b$

  1.2 *altrimenti poni* $b = b - a$

2 *restituisci* a

Un pò di storia

Etimologia

Il termine **algoritmo** (procedimento di calcolo) deriva dal termine latino medievale **algorismus**, che a sua volta è una translitterazione del nome del matematico persiamo Abu Jafar Mohammad ibn-Musa al-Khowarismi, il padre dell'algebra.

Algoritmi nella storia

- Algoritmi di tipo numerico sono studiati da babilonesi e indiani
- Algoritmi in uso ancora oggi sono stati studiati da matematici greci 2000 anni fa (Algoritmo di Euclide per il MCD, algoritmi geometrici, etc.)

Il concetto di Algoritmo

Definizione (Formale)

Un algoritmo è un insieme **finito** *e* **ordinato** *di passi semplici, materialmente* **eseguibili** *da un agente di calcolo e* **non ambigui***, che definiscono un procedimento atto a risolvere un problema (o una classe di problemi) utilizzando dei dati in input e producendo dei dati in output.*

Le quattro proprietà fondamentali di un algoritmo

1. Finitezza: la sequenza di passi di cui si compone un algoritmo deve essere finita.
2. Risolutività: i passi di cui si compone un algoritmo devono portare ad un risultato (la soluzione del problema o una diagnostica sulla mancata soluzione).
3. Effettuabilità: i passi devono essere eseguibili materialmente, deve quindi esistere un agente di calcolo in grado di eseguire ogni passo in un tempo finito.
4. Definitezza: ogni passo deve essere non ambiguo - non solo i passi devono essere espressi chiaramente, ma il passaggio da un passo al successivo deve avvenire in modo esplicitamente previsto dall'algoritmo.

Perchè parliamo di algoritmi

La teoria degli algoritmi ha iniziato a stabilizzarsi agli inizi del XX secolo.

Le tecniche di progettazione di algoritmi, di analisi di correttezza e di efficienza si sono invece evolute nella seconda metà del XX secolo grazie alla diffusione dei calcolatori elettronici

Ovunque si impieghi un calcolatore, occorrono algoritmi corretti ed efficienti in grado di utilizzarne al massimo le potenzialità. Esempi:

- controllo dei voli aerei
- regolazione reattori nucleari
- reperimento d'informazioni da archivi
- smistamento di comunicazioni telefoniche
- gioco degli scacchi
- controllo della produzione di una catena di montaggio

Come valutiamo gli algoritmi

Risolve correttamente il problema?

- un algoritmo si dice corretto se, per ogni istanza di input, si ferma con l'output corretto
- un algoritmo corretto risolve il problema computazionale dato
- dimostrazione matematica, descrizione informale

Risolve il problema in maniera efficiente (analisi di algoritmi)?

- definizione di **efficienza** in termini di utilizzo di tempo e/o memoria
- alcuni problemi non possono essere risolti in maniera efficiente
- per altri, esistono delle soluzioni ottime: non è possibile fare di meglio

Algoritmi e Programmi

Gli algoritmi vengono tradotti in **programmi**

I programmi si avvalgono di istruzioni e costrutti dei **linguaggi di programmazione** e possono essere eseguiti da un calcolatore elettronico

I programmi sono formulazioni **concrete** di algoritmi **astratti** che si basano su particolari rappresentazioni dei dati, e utilizzano operazioni di manipolazione dei dati, messe a disposizione da uno specifico linguaggio di programmazione

Le proprietà degli algoritmi sono talmente **fondamentali**, **generali** e **robuste**, da essere indipendenti dalle caratteristiche di specifici linguaggi di programmazione o di particolari calcolatori elettronici

Strutture Dati

Il concetto di algoritmo è inscindibile da quello di dato.

Per risolvere un problema, occorre **organizzare** ed **elaborare** dati in input per produrre dati in output.

Un algoritmo può essere visto come un manipolatore di dati: a fronte di dati in ingresso che descrivono il problema, produce dati in uscita come risultato del problema.

È fondamentale che i dati siano ben organizzati e strutturati in modo che il calcolatore li possa elaborare efficientemente.

L'efficienza di un algoritmo dipende in maniera **critica** dal modo in cui sono organizzati i dati su cui l'algoritmo stesso deve operare.

Strutturazione dei dati

Tra i dati possono sussistere delle relazioni logiche che definiscono delle "strutturazioni" dei dati stessi.

Per esempio, è possibile rappresentare dati eterogenei quali il nome, cognome e data di nascita di una persona, definendo un "dato strutturato" (un record) costituito dall' insieme dei due campi di testo (per il nome ed il cognome) ed uno alfa-numerico (per la data di nascita).

Una struttura dati è un insieme di valori logicamente correlati e opportunamente memorizzati, per i quali sono stati definiti dei costruttori, degli operatori di selezione e degli operatori di manipolazione.

Strutture dati e occupazione di memoria

Le strutture dati possono essere classificate in base alla loro occupazione di memoria

Strutture dati statiche: la quantitá di memoria di cui necessitano è determinabile a priori (esempi: array, record).

Strutture dati dinamiche: la quantità di memoria di cui esse necessitano varia a tempo d'esecuzione e può essere diversa da esecuzione a esecuzione (esempi: liste, code, pile, alberi, grafi).

Analizzeremo le principali strutture dati (ed in particolare, pile, code, liste, alberi, grafi) per fornirvi gli strumenti necessari per scegliere di volta in volta la struttura dati che megli si adatta all'algoritmo che state progettando.

"Clever" e "Efficient"

Obiettivo:

Studiare i modi più appropriati di organizzare i dati di un problema al fine di realizzare un algoritmo efficiente

Domanda:

- Che cosa intendiamo per appropriato "clever"?
- Che cosa intendiamo per efficiente "efficient"?

Parte II

Analisi di Algoritmi

Modello di Calcolo

Abbiamo bisogno di stabilire quale sarà la tecnologia di riferimento utilizzata per eseguire gli algoritmi quando saranno realizzati come programmi

Assumiamo di utilizzare

Mono-Processore + RAM (Random Access Memory)

assenza di concorrenza e parallelismo

Analisi di Algoritmi

Analizzare un algoritmo vuol dire determinare le risorse necessarie all'algoritmo in termini di

spazio di memoria

(quantità di memoria utilizzata durante l'esecuzione)

e

tempo computazionale

(tempo di esecuzione)

Analisi di Algoritmi

L'analisi della complessità di un algoritmo in termini di tempo di esecuzione consente di:

- stimare il tempo impiegato
- stimare il più grande input gestibile in termini ragionevoli
- confrontare l'efficienza di due algoritmi diversi
- ottimizzare le parti "critiche"

Definiamo una funzione T : **dimensione** $\rightarrow$ **tempo** impiegato

Dimensione dell'input

Per molti problemi (ex. l'ordinamento) la misura più naturale è il numero di elementi (**criteri di costo uniforme**)

Per altri (ex. moltiplicazione di numeri interi) la misura migliore è il numero totale di bit necessari per la rappresentazione dell'input (**criteri di costo logaritmico**)

In realtà, poichè ciascun elemento è rappresentato da un numero costante di bit, le due misure coincidono a meno di una costante moltiplicativa

In altri casi ancora, è più appropriato descrivere la dimensione con due numeri; ex: se l'input è una matrice bidimensionale la dimensione dell'input è $\#righe \times \#colonne$

Il tempo

Tempo = "*wall-clock time*", il tempo effettivamente impiegato per eseguire un algoritmo

Dipende da troppi fattori (non sempre prevedibili)

1. bravura del programmatore
2. linguaggio di programmazione utilizzato
3. processore, memoria (cache, primaria, secondaria)
4. sistema operativo, processi attualmente in esecuzione

Dobbiamo usare un **modello astratto**: introduciamo un concetto di tempo legato al **numero di operazioni elementari** o di passi eseguiti per il calcolo dell'output corrispondente

Tempo di esecuzione

Numero di operazioni elementari o “passi” eseguiti per il calcolo dell’output

$$\text{passo} \cong \text{una linea di pseudocodice}$$

Hp: ogni passo riferito ad una linea i, è eseguito in un tempo costante c_i

Parte III

Il problema dell'ordinamento

Il problema dell'ordinamento

Definizione (ordinamento)

Dato un insieme di n numeri $\langle a_1, a_2, \ldots, a_n \rangle$, *trovare un'opportuna permutazione permutazione degli indici in* $1, \cdots, n$ *tale che* $a_{\pi(1)} \leq a_{\pi(2)} \leq \ldots \leq a_{\pi(n)}$

Input: $\langle a_1, a_2, \ldots, a_n \rangle$

Output: $\langle a_{\pi(1)}, a_{\pi(2)}, \ldots, a_{\pi(n)} \rangle$
dove π è una permutazione degli indici $1, \ldots, n$

$\pi : \{1, ..., n\} \rightarrow \{1, ..., n\}$ tale che
se $\pi(i) = j$ allora j è la posizione corretta dell'elemento originariamente in posizione i

Un istanza del problema dell'ordinamento

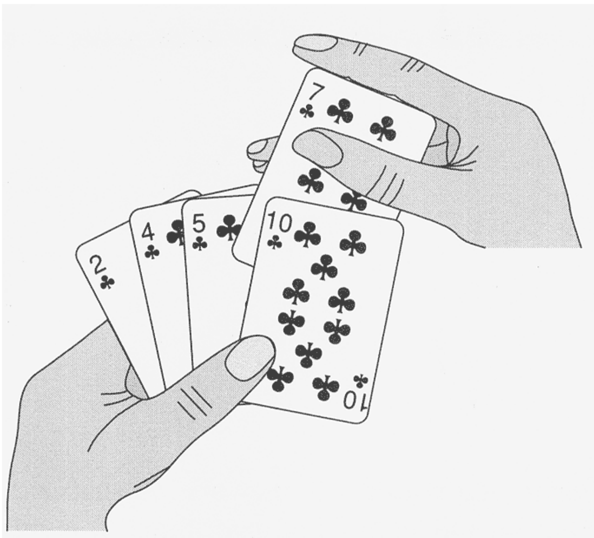
Input: (31, 41, 59, 26, 41, 58)

Output: (26, 31, 41, 41, 58, 59)

La scelta del migliore algoritmo dipende:

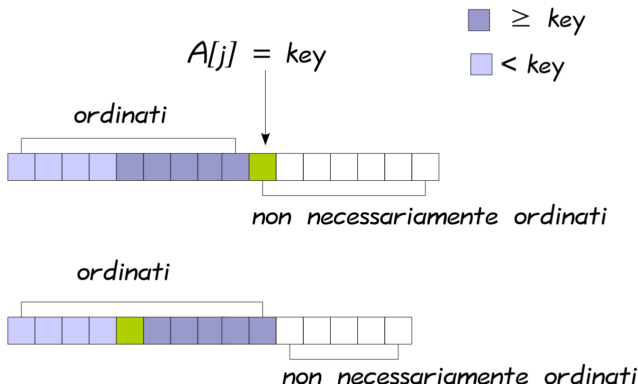
- dal numero di elementi da ordinare
- da quanto gli elementi siano già ordinati
- dispositivo di memoria (metodo d'accesso)

Idea per ordinare

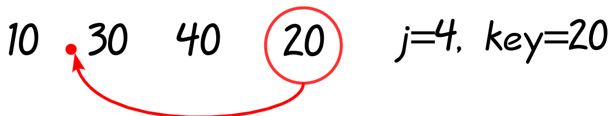


Idea per ordinare

Ad ogni passo si ha una sottosequenza ordinata in cui inserisco un nuovo elemento dell'input:



Idea per ordinare



10 20 30 40

InsertionSort

```
InsertionSort(A)
1   for j ← 2 to n
2   do key ← A[j]
3       i ← j − 1
4       while (i > 0 e A[i] > key)
5       do A[i + 1] ← A[i]
6           i ← i − 1
7       A[i + 1] ← key
```

Istanza $A = \{5, 2, 4, 6, 1, 3\}$

j = 2	5	2	4	6	1	3
j = 3	2	5	4	6	1	3
j = 4	2	4	5	6	1	3
j = 5	2	4	5	6	1	3
j = 6	1	2	4	5	6	3
j = 7	1	2	3	4	5	6

InsertionSort - analisi del costo computazionale

Il tempo impiegato dall'algoritmo INSERTIONSORT dipende:

- dalla dimensione dell'input, il numero n degli elementi
- dallo ordinamento implicito nella sequenza

L'INSERTIONSORT risulta efficiente nel caso in cui il numero degli elementi da ordinare è piccolo, e gli elementi sono già ordinati

InsertionSort - analisi del costo computazionale

```
INSERTIONSORT(A)
1  for j ← 2 to n
2  do key ← A[j]
3       i ← j − 1
4       while (i > 0 e A[i] > key)
5       do A[i + 1] ← A[i]
6            i ← i − 1
7       A[i + 1] ← key
```

La funzione $T(n)$ che definisce il costo di INSERTIONSORT è:

$$T(n) = \sum_{i=1}^{7} c_i \cdot n_i$$

dove, per $i = 1, \cdots, 7$:

1. c_i è il tempo costante necessario per eseguire il passo i;
2. n_i è il numero di volte che tale passo viene eseguito.

InsertionSort - analisi del costo computazionale

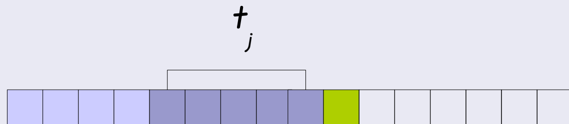
```
INSERTIONSORT(A)
1   for j ← 2 to n
2   do key ← A[j]
3       i ← j − 1
4       while (i > 0 e A[i] > key)
5       do A[i + 1] ← A[i]
6           i ← i − 1
7       A[i + 1] ← key
```

i	c_i	n_i
1	c_1	n
2	c_2	$n-1$
3	c_3	$n-1$
4	c_4	??
5	c_5	??
6	c_6	??
7	c_7	n-1

InsertionSort - analisi del costo computazionale

Sia $j = 2, \cdots, n$. Denotiamo con t_j il numero di volte in cui, una volta fissato j, valutiamo la guardia del ciclo while di riga 4

$$t_j = (\text{numero di elementi maggiori di } A[j]) + 1$$



Inoltre, se la guardia del ciclo while di riga 4 viene valutata t_j volte, allora gli assegnamenti di riga 5 e 6 vengono eseguiti $t_j - 1$ volte

$$n_4 = \sum_{j=2}^{n} t_j \text{ e } n_5 = n_6 = \sum_{j=2}^{n} (t_j - 1)$$

InsertionSort - analisi del costo computazionale

i	c_i	n_i
1	c_1	n
2	c_2	$n-1$
3	c_3	$n-1$
4	c_4	$\sum_{j=2}^{n} t_j$
5	c_5	$\sum_{j=2}^{n} (t_j - 1)$
6	c_6	$\sum_{j=2}^{n} (t_j - 1)$
7	c_7	$n-1$

$$
\begin{aligned}
T(n) \quad = \quad & c_1 \cdot n & + \\
& (c_2 + c_3 + c_7) \cdot (n-1) & + \\
& c_4 \cdot \sum_{j=2}^{n} t_j & + \\
& (c_5 + c_6) \cdot \sum_{j=2}^{n} (t_j - 1) &
\end{aligned}
$$

InsertionSort - analisi del costo computazionale

Il valore di t_j dipende dall'ordinamento parziale del vettore in input:

- Caso ottimo (vettore già ordinato) $t_j = 1$
- Caso peggiore (ordinato in maniera decrescente) $t_j = j$
- Caso medio $t_j = j/2$

InsertionSort - caso ottimo

Nel **caso ottimo** $t_j = 1$, quindi

$$
\begin{aligned}
T(n) = \quad & c_1 n + (c_2 + c_3 + c_7)(n-1) + c_4 \sum_{j=2}^{n} t_j + (c_5 + c_6) \sum_{j=2}^{n} (t_j - 1) \\
= \quad & c_1 n + (c_2 + c_3 + c_7)(n-1) + c_4 \sum_{j=2}^{n} 1 \\
= \quad & c_1 n + (c_2 + c_3 + c_7)(n-1) + c_4 (n-1) \\
= \quad & (c_1 + c_2 + c_3 + c_4 + c_7) n - (c_2 + c_3 + c_4 + c_7)
\end{aligned}
$$

$T(n) = an + b$ dove a e b sono delle costanti

$T(n)$ è una funzione lineare in n

Insertion-Sort - caso peggiore

Nel **caso peggiore** $t_j = j$, quindi

$$
\begin{aligned}
T(n) = \quad & c_1 n + (c_2 + c_3 + c_7)(n-1) + c_4 \sum_{j=2}^{n} t_j + (c_5 + c_6) \sum_{j=2}^{n} (t_j - 1) \\
= \quad & c_1 n + (c_2 + c_3 + c_7)(n-1) + c_4 \sum_{j=2}^{n} j + (c_5 + c_6) \sum_{j=2}^{n} (j - 1)
\end{aligned}
$$

Insertion-Sort - caso peggiore

Nel **caso peggiore** $t_j = j$, quindi

$$\begin{aligned} T(n) = \quad & c_1 n + (c_2 + c_3 + c_7)(n-1) + c_4 \sum_{j=2}^{n} t_j + (c_5 + c_6) \sum_{j=2}^{n} (t_j - 1) \\ = \quad & c_1 n + (c_2 + c_3 + c_7)(n-1) + c_4 \sum_{j=2}^{n} j + (c_5 + c_6) \sum_{j=2}^{n} (j - 1) \end{aligned}$$

Sommatorie notevoli

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

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

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

InsertionSort - caso peggiore

$$
\begin{aligned}
T(n) = \;& c_1 n + (c_2 + c_3 + c_7)(n-1) + c_4 \sum_{j=2}^{n} j + (c_5 + c_6) \sum_{j=2}^{n} (j-1) \\
= \;& c_1 n + (c_2 + c_3 + c_7)(n-1) + c_4\left(\frac{n^2}{2} + \frac{n}{2} - 1\right) + (c_5 + c_6)\left(\frac{n^2}{2} - \frac{n}{2}\right) \\
= \;& \frac{c_4 + c_5 + c_6}{2} n^2 + \left(c_1 + c_2 + c_3 + \frac{c_4 - c_5 - c_6}{2} + c_7\right) n - (c_2 + c_3 + c_4 + c_7)
\end{aligned}
$$

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

dove a, b e c sono delle costanti

$T(n)$ è una funzione quadratica in n

Analisi nel caso medio

- L'analisi del tempo di esecuzione nel **caso medio** analizza come si comporta l'algoritmo in "media"
- In alcune circostanze non è necessario che un algoritmo operi con una quantità limitata di risorsa per ogni possibile input, ma è sufficiente che sia "generalmente efficiente"
- Per ogni n, il costo dell'algoritmo viene calcolato come la somma dei costi per ogni possibile input di dimensione n tenendo conto della probabilità che ciascun singolo caso si verifichi

$$T_{medio}(n) = \sum_{\{I:|I|=n\}} T(I)Pr(I)$$

Analisi nel caso medio

- Bisogna determinare la distribuzione di probabilità di ciascun input
- **Problema**: non sempre si dispone delle informazioni necessarie per determinare la distribuzione di probabilità *Pr*
- **Una possibile soluzione**: fare delle assunzioni plausibili sulle caratteristiche del problema
- Ad esempio, nel caso del InsertionSort è plausibile assumere che, mediamente, solo la metà degli elementi della porzione di vettore $A[1...j-1]$ sia maggiore di *key*, e quindi che $t_j = j/2$

Insertion Sort: analisi del costo computazionale nel caso medio

Nel **caso medio** $t_j = j/2$, quindi

$$\begin{aligned} T(n) = \quad & c_1 n + (c_2 + c_3 + c_7)(n-1) + c_4 \sum_{j=2}^{n} t_j + (c_5 + c_6) \sum_{j=2}^{n} (t_j - 1) \\ = \quad & c_1 n + (c_2 + c_3 + c_7)(n-1) + c_4 \sum_{j=2}^{n} j/2 \\ & + (c_5 + c_6) \sum_{j=2}^{n} (j/2 - 1) \end{aligned}$$

$$\sum_{j=2}^{n} j/2 = \frac{1}{2} \sum_{j=2}^{n} j = \frac{1}{2} (\sum_{j=1}^{n} j - 1) = \frac{1}{2} (\frac{n^2}{2} + \frac{n}{2} - 1) = \frac{n^2}{4} + \frac{n}{4} - \frac{1}{2}$$

$$\sum_{j=2}^{n} (j/2 - 1) = \sum_{j=2}^{n} \frac{j-2}{2} = \frac{1}{2} \sum_{j=2}^{n} j - 2 = \frac{1}{2} \sum_{j=0}^{n-2} j = \frac{1}{2} \sum_{j=1}^{n-2} j =$$

$$\frac{1}{2} \frac{(n-2)(n-1)}{2} = \frac{1}{2} \frac{n^2 - 3n + 2}{2} = \frac{n^2}{4} - \frac{3n}{4} + \frac{1}{2}$$

Insertion Sort: analisi del costo computazionale nel caso medio

$$
\begin{aligned}
T(n) = \quad & c_1 n + (c_2 + c_3 + c_7)(n-1) + c_4 \sum_{j=2}^{n} j/2 \\
& + (c_5 + c_6) \sum_{j=2}^{n} (j/2 - 1) \\
= \quad & c_1 n + (c_2 + c_3 + c_7)(n-1) + c_4 \left(\frac{n^2}{4} + \frac{n}{4} - \frac{1}{2}\right) + \\
& (c_5 + c_6)\left(\frac{n^2}{4} - \frac{3n}{4} + \frac{1}{2}\right) \\
= \quad & \frac{c_4 + c_5 + c_6}{4} n^2 + \left(c_1 + c_2 + c_3 + \frac{c_4 - 3c_5 - 3c_6}{2} + c_7\right) n + \\
& \left(\frac{c_5 + c_6}{2} - \left(c_2 + c_3 + \frac{c_4}{2} + c_7\right)\right)
\end{aligned}
$$

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

dove a, b e c sono delle costanti

$T(n)$ è una funzione quadratica in n

Analisi del caso peggiore vs. analisi del caso medio

Il tempo di esecuzione nel caso ottimo non è molto significativo

Di solito prenderemo in considerazione il tempo di esecuzione nel caso peggiore. Perchè??

- il tempo di esecuzione nel caso peggiore rappresenta una limitazione superiore sui tempi di esecuzione per qualsiasi input, non possiamo fare di peggio
- per alcuni algoritmi il caso peggiore si verifica abbastanza spesso (ex. ricerca di un elemento in una base di dati)
- il caso medio è in generale altrettanto cattivo quanto il caso peggiore (vedi Insertion Sort)