

Algoritmi e Strutture Dati

Divide-et-impera

Alberto Montresor

Università di Trento

2023/11/25

This work is licensed under a Creative Commons
Attribution-ShareAlike 4.0 International License.



Sommario

- 1 Introduzione
- 2 Torri di Hanoi
- 3 Quicksort
- 4 Algoritmo di Strassen
- 5 Esercizio

Risoluzione problemi

Dato un problema

- Non ci sono "ricette generali" per risolverlo in modo efficiente
- Tuttavia, è possibile evidenziare quattro fasi
 - Classificazione del problema
 - Caratterizzazione della soluzione
 - Tecnica di progetto
 - Utilizzo di strutture dati
- Queste fasi non sono necessariamente sequenziali

Classificazione dei problemi

Problemi decisionali

- Il dato di ingresso soddisfa una certa proprietà?
- Soluzione: risposta sì/no
- Esempio: Stabilire se un grafo è connesso

Problemi di ricerca

- Spazio di ricerca: insieme di "soluzioni" possibili
- Soluzione ammissibile: soluzione che rispetta certi vincoli
- Esempio: posizione di una sottostringa in una stringa

Classificazione dei problemi

Problemi di ottimizzazione

- Ogni soluzione è associata ad una funzione di costo
- Vogliamo trovare la soluzione di costo minimo
- Esempio: cammino più breve fra due nodi

Definizione matematica del problema

È fondamentale definire bene il problema in modo formale

- Spesso la formulazione è banale...
- ... ma può suggerire una prima idea di soluzione
- Esempio: Data una sequenza di n elementi, una permutazione ordinata è data dal minimo seguito da una permutazione ordinata dei restanti $n - 1$ elementi (Selection Sort)

La definizione matematica può suggerire una possibile tecnica

- Sottostruttura ottima \rightarrow Programmazione dinamica
- Proprietà greedy \rightarrow Tecnica greedy

Tecniche di soluzione problemi

Divide-et-impera

- Un problema viene suddiviso in sotto-problemi indipendenti, che vengono risolti ricorsivamente (top-down)
- Ambito: problemi di decisione, ricerca

Programmazione dinamica

- La soluzione viene costruita (bottom-up) a partire da un insieme di sotto-problemi potenzialmente ripetuti
- Ambito: problemi di ottimizzazione

Memoization (o annotazione)

- Versione top-down della programmazione dinamica

Tecniche di soluzione problemi

Tecnica greedy

- Approccio "ingordo": si fa sempre la scelta localmente ottima

Backtrack

- Procediamo per "tentativi", tornando ogni tanto sui nostri passi

Ricerca locale

- La soluzione ottima viene trovata "migliorando" via via soluzioni esistenti

Algoritmi probabilistici

- Meglio scegliere con giudizio (ma in maniera costosa) o scegliere a caso ("gratuitamente")

Divide-et-impera

Tre fasi

- **Divide**: Dividi il problema in sotto-problemi più piccoli e indipendenti
- **Impera**: Risolvi i sotto-problemi ricorsivamente
- **Combina**: "unisci" le soluzioni dei sottoproblemi

Non esiste una ricetta "unica" per divide-et-impera

- Merge Sort: "divide" banale, "combina" complesso
- Quicksort: "divide" complesso, niente fase di "combina"
- È necessario uno sforzo creativo

Minimo divide-et-impera

```
int minrec(int[] A, int i, int j)
```

```
if i == j then
```

```
    return A[i]
```

```
else
```

```
     $m = \lfloor (i + j) / 2 \rfloor$ 
```

```
    return min(minrec(A, i, m), minrec(A, m + 1, j))
```

Complessità

Minimo divide-et-impera

```

int minrec(int[] A, int i, int j)


---


if i == j then
    |   return A[i]
else
    |   m =  $\lfloor (i + j) / 2 \rfloor$ 
    |   return min(minrec(A, i, m), minrec(A, m + 1, j))
  
```

Complessità

$$T(n) = \begin{cases} 2T(n/2) + 1 & n > 1 \\ 1 & n = 1 \end{cases}$$

Minimo divide-et-impera

```
int minrec(int[] A, int i, int j)
```

```
if i == j then
```

```
    | return A[i]
```

```
else
```

```
    | m =  $\lfloor (i + j) / 2 \rfloor$ 
```

```
    | return min(minrec(A, i, m), minrec(A, m + 1, j))
```

Complessità

$$T(n) = \begin{cases} 2T(n/2) + 1 & n > 1 \\ 1 & n = 1 \end{cases}$$

$T(n) = \Theta(n)$ – Non ne vale la pena!

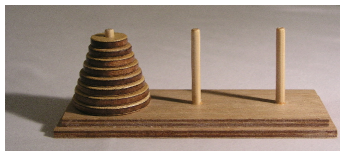
Sommario

- 1 Introduzione
- 2 Torri di Hanoi**
- 3 Quicksort
- 4 Algoritmo di Strassen
- 5 Esercizio

Le torri di Hanoi

Gioco matematico

- tre pioli
- n dischi di dimensioni diverse
- Inizialmente, i dischi sono impilati in ordine decrescente nel piolo di sinistra



https://it.wikipedia.org/wiki/File:Tower_of_Hanoi.jpeg

Scopo del gioco

- Impilare in ordine decrescente i dischi sul piolo di destra
- Senza mai impilare un disco più grande su uno più piccolo
- Muovendo al massimo un disco alla volta
- Utilizzando il piolo centrale come appoggio

Le torri di Hanoi

```
hanoi(int n, int src, int dest, int middle)
```

```
if n == 1 then
```

```
    print src → dest
```

```
else
```

```
    hanoi(n - 1, src, middle, dest)
```

```
    print src → dest
```

```
    hanoi(n - 1, middle, dest, src)
```

Divide-et-impera

- $n - 1$ dischi da *src* a *middle*
- 1 disco da *src* a *dest*
- $n - 1$ dischi da *middle* a *dest*



https://it.wikipedia.org/wiki/Torre_di_Hanoi#/media/File:Tower_of_Hanoi_4.gif

Le torri di Hanoi

```
hanoi(int n, int src, int dest, int middle)
```

```
if n == 1 then
```

```
    print src → dest
```

```
else
```

```
    hanoi(n - 1, src, middle, dest)
```

```
    print src → dest
```

```
    hanoi(n - 1, middle, dest, src)
```

Divide-et-impera

- $n - 1$ dischi da *src* a *middle*
- 1 disco da *src* a *dest*
- $n - 1$ dischi da *middle* a *dest*



https://it.wikipedia.org/wiki/Torre_di_Hanoi#/media/File:Tower_of_Hanoi_4.gif

Le torri di Hanoi

```
hanoi(int n, int src, int dest, int middle)
```

```
if n == 1 then
```

```
    print src → dest
```

```
else
```

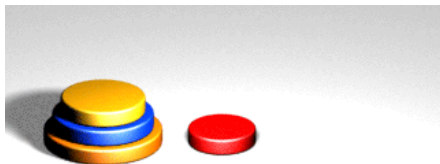
```
    hanoi(n - 1, src, middle, dest)
```

```
    print src → dest
```

```
    hanoi(n - 1, middle, dest, src)
```

Divide-et-impera

- $n - 1$ dischi da *src* a *middle*
- 1 disco da *src* a *dest*
- $n - 1$ dischi da *middle* a *dest*



https://it.wikipedia.org/wiki/Torre_di_Hanoi#/media/File:Tower_of_Hanoi_4.gif

Le torri di Hanoi

```
hanoi(int n, int src, int dest, int middle)
```

```
if n == 1 then
```

```
    print src → dest
```

```
else
```

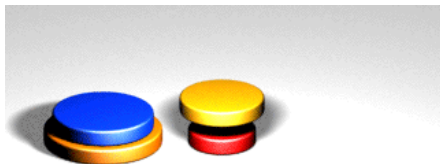
```
    hanoi(n - 1, src, middle, dest)
```

```
    print src → dest
```

```
    hanoi(n - 1, middle, dest, src)
```

Divide-et-impera

- $n - 1$ dischi da *src* a *middle*
- 1 disco da *src* a *dest*
- $n - 1$ dischi da *middle* a *dest*



https://it.wikipedia.org/wiki/Torre_di_Hanoi#/media/File:Tower_of_Hanoi_4.gif

Le torri di Hanoi

```
hanoi(int n, int src, int dest, int middle)
```

```
if n == 1 then
```

```
    print src → dest
```

```
else
```

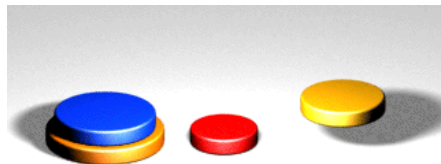
```
    hanoi(n - 1, src, middle, dest)
```

```
    print src → dest
```

```
    hanoi(n - 1, middle, dest, src)
```

Divide-et-impera

- $n - 1$ dischi da *src* a *middle*
- 1 disco da *src* a *dest*
- $n - 1$ dischi da *middle* a *dest*



https://it.wikipedia.org/wiki/Torre_di_Hanoi#/media/File:Tower_of_Hanoi_4.gif

Le torri di Hanoi

```
hanoi(int n, int src, int dest, int middle)
```

```
if n == 1 then
```

```
    print src → dest
```

```
else
```

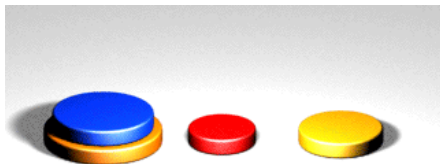
```
    hanoi(n - 1, src, middle, dest)
```

```
    print src → dest
```

```
    hanoi(n - 1, middle, dest, src)
```

Divide-et-impera

- $n - 1$ dischi da *src* a *middle*
- 1 disco da *src* a *dest*
- $n - 1$ dischi da *middle* a *dest*



https://it.wikipedia.org/wiki/Torre_di_Hanoi#/media/File:Tower_of_Hanoi_4.gif

Le torri di Hanoi

```
hanoi(int n, int src, int dest, int middle)
```

```
if n == 1 then
```

```
    print src → dest
```

```
else
```

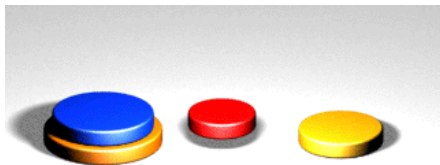
```
    hanoi(n - 1, src, middle, dest)
```

```
    print src → dest
```

```
    hanoi(n - 1, middle, dest, src)
```

Divide-et-impera

- $n - 1$ dischi da *src* a *middle*
- 1 disco da *src* a *dest*
- $n - 1$ dischi da *middle* a *dest*



https://it.wikipedia.org/wiki/Torre_di_Hanoi#/media/File:Tower_of_Hanoi_4.gif

Le torri di Hanoi

```
hanoi(int n, int src, int dest, int middle)
```

```
if n == 1 then
```

```
    print src → dest
```

```
else
```

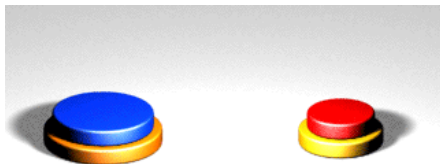
```
    hanoi(n - 1, src, middle, dest)
```

```
    print src → dest
```

```
    hanoi(n - 1, middle, dest, src)
```

Divide-et-impera

- $n - 1$ dischi da *src* a *middle*
- 1 disco da *src* a *dest*
- $n - 1$ dischi da *middle* a *dest*



https://it.wikipedia.org/wiki/Torre_di_Hanoi#/media/File:Tower_of_Hanoi_4.gif

Le torri di Hanoi

```
hanoi(int n, int src, int dest, int middle)
```

```
if n == 1 then
```

```
    print src → dest
```

```
else
```

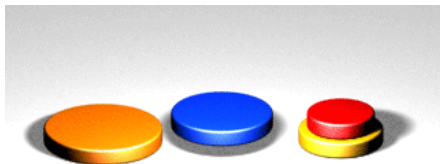
```
    hanoi(n - 1, src, middle, dest)
```

```
    print src → dest
```

```
    hanoi(n - 1, middle, dest, src)
```

Divide-et-impera

- $n - 1$ dischi da *src* a *middle*
- 1 disco da *src* a *dest*
- $n - 1$ dischi da *middle* a *dest*



https://it.wikipedia.org/wiki/Torre_di_Hanoi#/media/File:Tower_of_Hanoi_4.gif

Le torri di Hanoi

```
hanoi(int n, int src, int dest, int middle)
```

```
if n == 1 then
```

```
    print src → dest
```

```
else
```

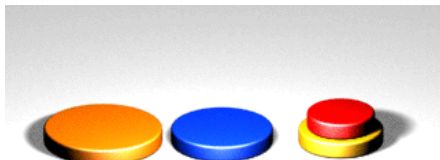
```
    hanoi(n - 1, src, middle, dest)
```

```
    print src → dest
```

```
    hanoi(n - 1, middle, dest, src)
```

Divide-et-impera

- $n - 1$ dischi da *src* a *middle*
- 1 disco da *src* a *dest*
- $n - 1$ dischi da *middle* a *dest*



https://it.wikipedia.org/wiki/Torre_di_Hanoi#/media/File:Tower_of_Hanoi_4.gif

Le torri di Hanoi

```
hanoi(int n, int src, int dest, int middle)
```

```
if n == 1 then
```

```
    print src → dest
```

```
else
```

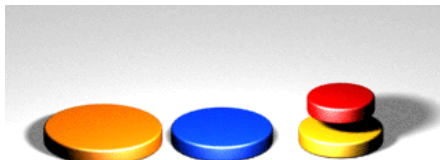
```
    hanoi(n - 1, src, middle, dest)
```

```
    print src → dest
```

```
    hanoi(n - 1, middle, dest, src)
```

Divide-et-impera

- $n - 1$ dischi da *src* a *middle*
- 1 disco da *src* a *dest*
- $n - 1$ dischi da *middle* a *dest*



https://it.wikipedia.org/wiki/Torre_di_Hanoi#/media/File:Tower_of_Hanoi_4.gif

Le torri di Hanoi

```
hanoi(int n, int src, int dest, int middle)
```

```
if n == 1 then
```

```
    print src → dest
```

```
else
```

```
    hanoi(n - 1, src, middle, dest)
```

```
    print src → dest
```

```
    hanoi(n - 1, middle, dest, src)
```

Divide-et-impera

- $n - 1$ dischi da *src* a *middle*
- 1 disco da *src* a *dest*
- $n - 1$ dischi da *middle* a *dest*



https://it.wikipedia.org/wiki/Torre_di_Hanoi#/media/File:Tower_of_Hanoi_4.gif

Le torri di Hanoi

```
hanoi(int n, int src, int dest, int middle)
```

```
if n == 1 then
```

```
    print src → dest
```

```
else
```

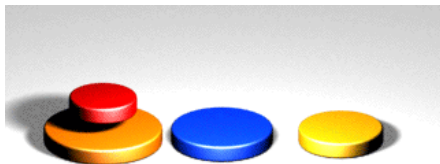
```
    hanoi(n - 1, src, middle, dest)
```

```
    print src → dest
```

```
    hanoi(n - 1, middle, dest, src)
```

Divide-et-impera

- $n - 1$ dischi da *src* a *middle*
- 1 disco da *src* a *dest*
- $n - 1$ dischi da *middle* a *dest*



https://it.wikipedia.org/wiki/Torre_di_Hanoi#/media/File:Tower_of_Hanoi_4.gif

Le torri di Hanoi

```
hanoi(int n, int src, int dest, int middle)
```

```
if n == 1 then
```

```
    print src → dest
```

```
else
```

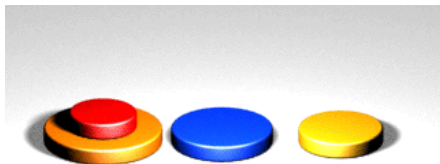
```
    hanoi(n - 1, src, middle, dest)
```

```
    print src → dest
```

```
    hanoi(n - 1, middle, dest, src)
```

Divide-et-impera

- $n - 1$ dischi da *src* a *middle*
- 1 disco da *src* a *dest*
- $n - 1$ dischi da *middle* a *dest*



https://it.wikipedia.org/wiki/Torre_di_Hanoi#/media/File:Tower_of_Hanoi_4.gif

Le torri di Hanoi

```
hanoi(int n, int src, int dest, int middle)
```

```
if n == 1 then
```

```
    print src → dest
```

```
else
```

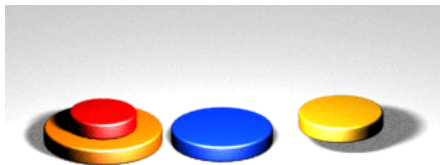
```
    hanoi(n - 1, src, middle, dest)
```

```
    print src → dest
```

```
    hanoi(n - 1, middle, dest, src)
```

Divide-et-impera

- $n - 1$ dischi da *src* a *middle*
- 1 disco da *src* a *dest*
- $n - 1$ dischi da *middle* a *dest*



https://it.wikipedia.org/wiki/Torre_di_Hanoi#/media/File:Tower_of_Hanoi_4.gif

Le torri di Hanoi

```
hanoi(int n, int src, int dest, int middle)
```

```
if n == 1 then
```

```
    print src → dest
```

```
else
```

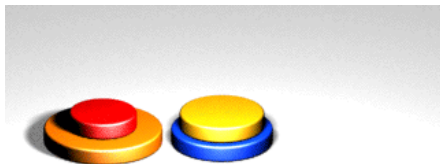
```
    hanoi(n - 1, src, middle, dest)
```

```
    print src → dest
```

```
    hanoi(n - 1, middle, dest, src)
```

Divide-et-impera

- $n - 1$ dischi da *src* a *middle*
- 1 disco da *src* a *dest*
- $n - 1$ dischi da *middle* a *dest*



https://it.wikipedia.org/wiki/Torre_di_Hanoi#/media/File:Tower_of_Hanoi_4.gif

Le torri di Hanoi

```
hanoi(int n, int src, int dest, int middle)
```

```
if n == 1 then
```

```
    print src → dest
```

```
else
```

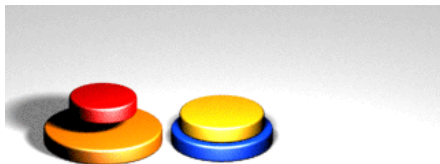
```
    hanoi(n - 1, src, middle, dest)
```

```
    print src → dest
```

```
    hanoi(n - 1, middle, dest, src)
```

Divide-et-impera

- $n - 1$ dischi da *src* a *middle*
- 1 disco da *src* a *dest*
- $n - 1$ dischi da *middle* a *dest*



https://it.wikipedia.org/wiki/Torre_di_Hanoi#/media/File:Tower_of_Hanoi_4.gif

Le torri di Hanoi

```
hanoi(int n, int src, int dest, int middle)
```

```
if n == 1 then
```

```
    print src → dest
```

```
else
```

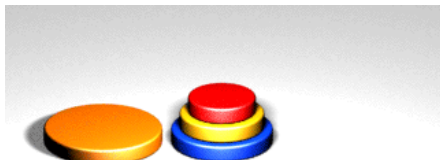
```
    hanoi(n - 1, src, middle, dest)
```

```
    print src → dest
```

```
    hanoi(n - 1, middle, dest, src)
```

Divide-et-impera

- $n - 1$ dischi da *src* a *middle*
- 1 disco da *src* a *dest*
- $n - 1$ dischi da *middle* a *dest*



https://it.wikipedia.org/wiki/Torre_di_Hanoi#/media/File:Tower_of_Hanoi_4.gif

Le torri di Hanoi

```
hanoi(int n, int src, int dest, int middle)
```

```
if n == 1 then
```

```
    print src → dest
```

```
else
```

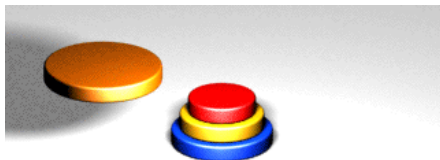
```
    hanoi(n - 1, src, middle, dest)
```

```
    print src → dest
```

```
    hanoi(n - 1, middle, dest, src)
```

Divide-et-impera

- $n - 1$ dischi da *src* a *middle*
- 1 disco da *src* a *dest*
- $n - 1$ dischi da *middle* a *dest*



https://it.wikipedia.org/wiki/Torre_di_Hanoi#/media/File:Tower_of_Hanoi_4.gif

Le torri di Hanoi

```
hanoi(int n, int src, int dest, int middle)
```

```
if n == 1 then
```

```
    print src → dest
```

```
else
```

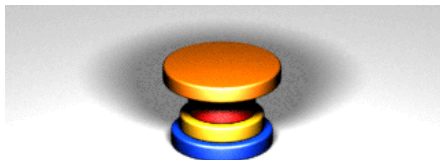
```
    hanoi(n - 1, src, middle, dest)
```

```
    print src → dest
```

```
    hanoi(n - 1, middle, dest, src)
```

Divide-et-impera

- $n - 1$ dischi da *src* a *middle*
- 1 disco da *src* a *dest*
- $n - 1$ dischi da *middle* a *dest*



https://it.wikipedia.org/wiki/Torre_di_Hanoi#/media/File:Tower_of_Hanoi_4.gif

Le torri di Hanoi

```
hanoi(int n, int src, int dest, int middle)
```

```
if n == 1 then
```

```
    print src → dest
```

```
else
```

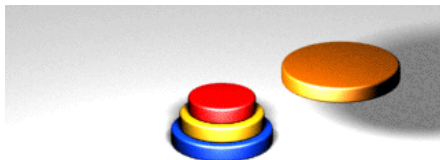
```
    hanoi(n - 1, src, middle, dest)
```

```
    print src → dest
```

```
    hanoi(n - 1, middle, dest, src)
```

Divide-et-impera

- $n - 1$ dischi da *src* a *middle*
- 1 disco da *src* a *dest*
- $n - 1$ dischi da *middle* a *dest*



https://it.wikipedia.org/wiki/Torre_di_Hanoi#/media/File:Tower_of_Hanoi_4.gif

Le torri di Hanoi

```
hanoi(int n, int src, int dest, int middle)
```

```
if n == 1 then
```

```
    print src → dest
```

```
else
```

```
    hanoi(n - 1, src, middle, dest)
```

```
    print src → dest
```

```
    hanoi(n - 1, middle, dest, src)
```

Divide-et-impera

- $n - 1$ dischi da *src* a *middle*
- 1 disco da *src* a *dest*
- $n - 1$ dischi da *middle* a *dest*



https://it.wikipedia.org/wiki/Torre_di_Hanoi#/media/File:Tower_of_Hanoi_4.gif

Le torri di Hanoi

```
hanoi(int n, int src, int dest, int middle)
```

```
if n == 1 then
```

```
    print src → dest
```

```
else
```

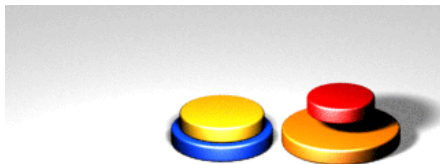
```
    hanoi(n - 1, src, middle, dest)
```

```
    print src → dest
```

```
    hanoi(n - 1, middle, dest, src)
```

Divide-et-impera

- $n - 1$ dischi da *src* a *middle*
- 1 disco da *src* a *dest*
- $n - 1$ dischi da *middle* a *dest*



https://it.wikipedia.org/wiki/Torre_di_Hanoi#/media/File:Tower_of_Hanoi_4.gif

Le torri di Hanoi

```
hanoi(int n, int src, int dest, int middle)
```

```
if n == 1 then
```

```
    print src → dest
```

```
else
```

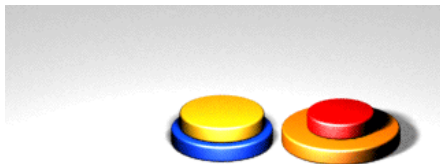
```
    hanoi(n - 1, src, middle, dest)
```

```
    print src → dest
```

```
    hanoi(n - 1, middle, dest, src)
```

Divide-et-impera

- $n - 1$ dischi da *src* a *middle*
- 1 disco da *src* a *dest*
- $n - 1$ dischi da *middle* a *dest*



https://it.wikipedia.org/wiki/Torre_di_Hanoi#/media/File:Tower_of_Hanoi_4.gif

Le torri di Hanoi

```
hanoi(int n, int src, int dest, int middle)
```

```
if n == 1 then
```

```
    print src → dest
```

```
else
```

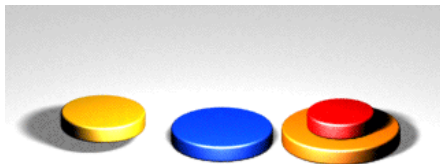
```
    hanoi(n - 1, src, middle, dest)
```

```
    print src → dest
```

```
    hanoi(n - 1, middle, dest, src)
```

Divide-et-impera

- $n - 1$ dischi da *src* a *middle*
- 1 disco da *src* a *dest*
- $n - 1$ dischi da *middle* a *dest*



https://it.wikipedia.org/wiki/Torre_di_Hanoi#/media/File:Tower_of_Hanoi_4.gif

Le torri di Hanoi

```
hanoi(int n, int src, int dest, int middle)
```

```
if n == 1 then
```

```
    print src → dest
```

```
else
```

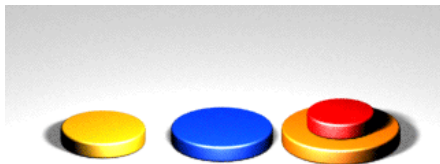
```
    hanoi(n - 1, src, middle, dest)
```

```
    print src → dest
```

```
    hanoi(n - 1, middle, dest, src)
```

Divide-et-impera

- $n - 1$ dischi da *src* a *middle*
- 1 disco da *src* a *dest*
- $n - 1$ dischi da *middle* a *dest*



https://it.wikipedia.org/wiki/Torre_di_Hanoi#/media/File:Tower_of_Hanoi_4.gif

Le torri di Hanoi

```
hanoi(int n, int src, int dest, int middle)
```

```
if n == 1 then
```

```
    print src → dest
```

```
else
```

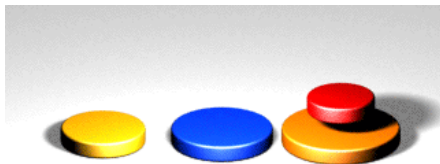
```
    hanoi(n - 1, src, middle, dest)
```

```
    print src → dest
```

```
    hanoi(n - 1, middle, dest, src)
```

Divide-et-impera

- $n - 1$ dischi da *src* a *middle*
- 1 disco da *src* a *dest*
- $n - 1$ dischi da *middle* a *dest*



https://it.wikipedia.org/wiki/Torre_di_Hanoi#/media/File:Tower_of_Hanoi_4.gif

Le torri di Hanoi

```
hanoi(int n, int src, int dest, int middle)
```

```
if n == 1 then
```

```
    print src → dest
```

```
else
```

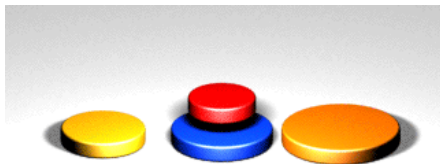
```
    hanoi(n - 1, src, middle, dest)
```

```
    print src → dest
```

```
    hanoi(n - 1, middle, dest, src)
```

Divide-et-impera

- $n - 1$ dischi da *src* a *middle*
- 1 disco da *src* a *dest*
- $n - 1$ dischi da *middle* a *dest*



https://it.wikipedia.org/wiki/Torre_di_Hanoi#/media/File:Tower_of_Hanoi_4.gif

Le torri di Hanoi

```
hanoi(int n, int src, int dest, int middle)
```

```
if n == 1 then
```

```
    print src → dest
```

```
else
```

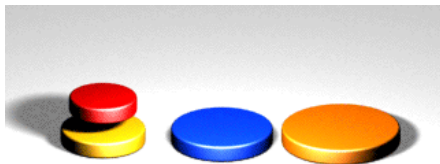
```
    hanoi(n - 1, src, middle, dest)
```

```
    print src → dest
```

```
    hanoi(n - 1, middle, dest, src)
```

Divide-et-impera

- $n - 1$ dischi da *src* a *middle*
- 1 disco da *src* a *dest*
- $n - 1$ dischi da *middle* a *dest*



https://it.wikipedia.org/wiki/Torre_di_Hanoi#/media/File:Tower_of_Hanoi_4.gif

Le torri di Hanoi

```
hanoi(int n, int src, int dest, int middle)
```

```
if n == 1 then
```

```
    print src → dest
```

```
else
```

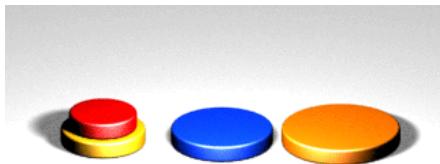
```
    hanoi(n - 1, src, middle, dest)
```

```
    print src → dest
```

```
    hanoi(n - 1, middle, dest, src)
```

Divide-et-impera

- $n - 1$ dischi da *src* a *middle*
- 1 disco da *src* a *dest*
- $n - 1$ dischi da *middle* a *dest*



https://it.wikipedia.org/wiki/Torre_di_Hanoi#/media/File:Tower_of_Hanoi_4.gif

Le torri di Hanoi

```
hanoi(int n, int src, int dest, int middle)
```

```
if n == 1 then
```

```
    print src → dest
```

```
else
```

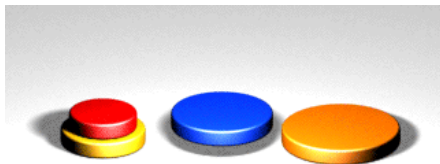
```
    hanoi(n - 1, src, middle, dest)
```

```
    print src → dest
```

```
    hanoi(n - 1, middle, dest, src)
```

Divide-et-impera

- $n - 1$ dischi da *src* a *middle*
- 1 disco da *src* a *dest*
- $n - 1$ dischi da *middle* a *dest*



https://it.wikipedia.org/wiki/Torre_di_Hanoi#/media/File:Tower_of_Hanoi_4.gif

Le torri di Hanoi

```
hanoi(int n, int src, int dest, int middle)
```

```
if n == 1 then
```

```
    print src → dest
```

```
else
```

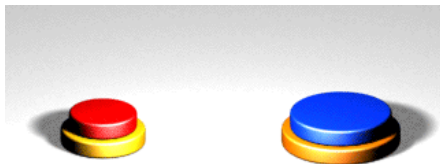
```
    hanoi(n - 1, src, middle, dest)
```

```
    print src → dest
```

```
    hanoi(n - 1, middle, dest, src)
```

Divide-et-impera

- $n - 1$ dischi da *src* a *middle*
- 1 disco da *src* a *dest*
- $n - 1$ dischi da *middle* a *dest*



https://it.wikipedia.org/wiki/Torre_di_Hanoi#/media/File:Tower_of_Hanoi_4.gif

Le torri di Hanoi

```
hanoi(int n, int src, int dest, int middle)
```

```
if n == 1 then
```

```
    print src → dest
```

```
else
```

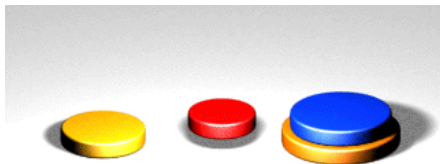
```
    hanoi(n - 1, src, middle, dest)
```

```
    print src → dest
```

```
    hanoi(n - 1, middle, dest, src)
```

Divide-et-impera

- $n - 1$ dischi da *src* a *middle*
- 1 disco da *src* a *dest*
- $n - 1$ dischi da *middle* a *dest*



https://it.wikipedia.org/wiki/Torre_di_Hanoi#/media/File:Tower_of_Hanoi_4.gif

Le torri di Hanoi

```
hanoi(int n, int src, int dest, int middle)
```

```
if n == 1 then
```

```
    print src → dest
```

```
else
```

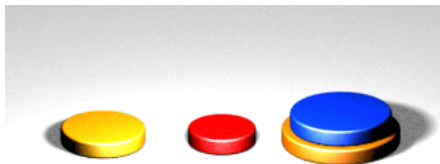
```
    hanoi(n - 1, src, middle, dest)
```

```
    print src → dest
```

```
    hanoi(n - 1, middle, dest, src)
```

Divide-et-impera

- $n - 1$ dischi da *src* a *middle*
- 1 disco da *src* a *dest*
- $n - 1$ dischi da *middle* a *dest*



https://it.wikipedia.org/wiki/Torre_di_Hanoi#/media/File:Tower_of_Hanoi_4.gif

Le torri di Hanoi

```
hanoi(int n, int src, int dest, int middle)
```

```
if n == 1 then
```

```
    print src → dest
```

```
else
```

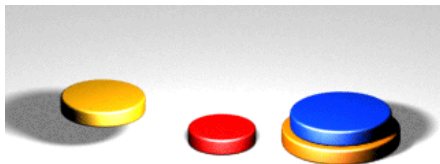
```
    hanoi(n - 1, src, middle, dest)
```

```
    print src → dest
```

```
    hanoi(n - 1, middle, dest, src)
```

Divide-et-impera

- $n - 1$ dischi da *src* a *middle*
- 1 disco da *src* a *dest*
- $n - 1$ dischi da *middle* a *dest*



https://it.wikipedia.org/wiki/Torre_di_Hanoi#/media/File:Tower_of_Hanoi_4.gif

Le torri di Hanoi

```
hanoi(int n, int src, int dest, int middle)
```

```
if n == 1 then
```

```
    print src → dest
```

```
else
```

```
    hanoi(n - 1, src, middle, dest)
```

```
    print src → dest
```

```
    hanoi(n - 1, middle, dest, src)
```

Divide-et-impera

- $n - 1$ dischi da *src* a *middle*
- 1 disco da *src* a *dest*
- $n - 1$ dischi da *middle* a *dest*



https://it.wikipedia.org/wiki/Torre_di_Hanoi#/media/File:Tower_of_Hanoi_4.gif

Le torri di Hanoi

```
hanoi(int n, int src, int dest, int middle)
```

```
if n == 1 then
```

```
    print src → dest
```

```
else
```

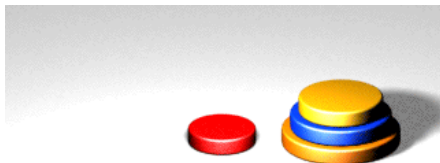
```
    hanoi(n - 1, src, middle, dest)
```

```
    print src → dest
```

```
    hanoi(n - 1, middle, dest, src)
```

Divide-et-impera

- $n - 1$ dischi da *src* a *middle*
- 1 disco da *src* a *dest*
- $n - 1$ dischi da *middle* a *dest*



https://it.wikipedia.org/wiki/Torre_di_Hanoi#/media/File:Tower_of_Hanoi_4.gif

Le torri di Hanoi

```
hanoi(int n, int src, int dest, int middle)
```

```
if n == 1 then
```

```
    print src → dest
```

```
else
```

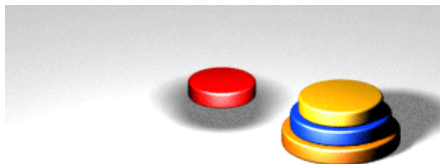
```
    hanoi(n - 1, src, middle, dest)
```

```
    print src → dest
```

```
    hanoi(n - 1, middle, dest, src)
```

Divide-et-impera

- $n - 1$ dischi da *src* a *middle*
- 1 disco da *src* a *dest*
- $n - 1$ dischi da *middle* a *dest*



https://it.wikipedia.org/wiki/Torre_di_Hanoi#/media/File:Tower_of_Hanoi_4.gif

Le torri di Hanoi

```
hanoi(int n, int src, int dest, int middle)
```

```
if n == 1 then
```

```
    print src → dest
```

```
else
```

```
    hanoi(n - 1, src, middle, dest)
```

```
    print src → dest
```

```
    hanoi(n - 1, middle, dest, src)
```

Divide-et-impera

- $n - 1$ dischi da *src* a *middle*
- 1 disco da *src* a *dest*
- $n - 1$ dischi da *middle* a *dest*



https://it.wikipedia.org/wiki/Torre_di_Hanoi#/media/File:Tower_of_Hanoi_4.gif

Le torri di Hanoi

```
hanoi(int n, int src, int dest, int middle)
```

```
if n = 1 then
```

```
    | print src → dest
```

```
else
```

```
    | hanoi(n - 1, src, middle, dest)
```

```
    | print src → dest
```

```
    | hanoi(n - 1, middle, dest, src)
```

Costo computazionale

?

Le torri di Hanoi

```
hanoi(int n, int src, int dest, int middle)
```

```
if n = 1 then
```

```
    print src → dest
```

```
else
```

```
    hanoi(n - 1, src, middle, dest)
```

```
    print src → dest
```

```
    hanoi(n - 1, middle, dest, src)
```

Costo computazionale

- Ricorrenza: $T(n) = 2T(n - 1) + 1$
- Costo computazionale? $O(2^n)$
- Questa soluzione è ottima (si può dimostrare)

Sommario

- 1 Introduzione
- 2 Torri di Hanoi
- 3 Quicksort**
- 4 Algoritmo di Strassen
- 5 Esercizio

Quicksort (Hoare, 1961)

Algoritmo di ordinamento basato su divide-et-impera

- Caso medio: $O(n \log n)$
- Caso pessimo: $O(n^2)$

Caso medio vs caso pessimo

- Il fattore costante di Quicksort è migliore di Merge Sort
- "In-memory": non utilizza memoria aggiuntiva
- Tecniche "euristiche" per evitare il caso pessimo
- Quindi spesso è preferito ad altri algoritmi

R. Sedgwick, "*Implementing Quicksort Programs*". Communications of the ACM, 21(10):847-857, 1978. <http://portal.acm.org/citation.cfm?id=359631>

Quicksort

Input

- Vettore $A[1 \dots n]$,
- Indici $start, end$ tali che $1 \leq start \leq end \leq n$

Divide

- Sceglie un valore $p \in A[start \dots end]$ detto **perno** (**pivot**)
- Sposta gli elementi del vettore $A[start \dots end]$ in modo che:

<i>start</i>				<i>j</i>	<i>end</i>						
13	14	15	12	20	27	29	30	21	25	28	
$A[start \dots j-1]$				$A[j]$	$A[j+1 \dots end]$						
$A[i] < pivot$				$A[j] = pivot$	$A[i] \geq pivot$						

- L'indice j del perno va calcolato opportunamente

Quicksort

Impera

Ordina i due sottovettori $A[start \dots j - 1]$ e $A[j + 1 \dots end]$ richiamando ricorsivamente Quicksort

Combina

Non fa nulla: infatti,

- il primo sottovettore,
- $A[j]$,
- il secondo sottovettore

formano già un vettore ordinato

Quicksort – pivot()

```
int pivot(ITEM[] A, int start, int end)
```

```
ITEM pivot = A[start]
```

```
int j = start
```

```
for i = start + 1 to end do
```

```
    if A[i] < pivot then
```

```
        j = j + 1
```

```
        swap(A, i, j)
```

```
A[start] = A[j]
```

```
A[j] = pivot
```

```
return j
```

```
swap(ITEM[] A, int i, int j)
```

```
ITEM temp = A[i]
```

```
A[i] = A[j]
```

```
A[j] = temp
```

Funzionamento pivot()

<i>pivot</i>	<i>start</i>	<i>end</i>									
	20	14	28	29	15	27	12	30	21	25	13

<i>pivot</i>	<i>start</i>	<i>end</i>									
20	20	14	28	29	15	27	12	30	21	25	13

<i>pivot</i>	<i>i</i>										
20	20	14	28	29	15	27	12	30	21	25	13

<i>pivot</i>	<i>j</i>	<i>i</i>									
20	20	14	28	29	15	27	12	30	21	25	13

<i>pivot</i>	<i>j</i>	<i>i</i>									
20	20	14	28	29	15	27	12	30	21	25	13

<i>pivot</i>	<i>j</i>	<i>i</i>									
20	20	14	28	29	15	27	12	30	21	25	13

<i>pivot</i>	<i>j</i>	<i>i</i>									
20	20	14	15	29	28	27	12	30	21	25	13

j $pivot = A[start]$

$$j = lo$$

$$i = lo+1$$

$$A[i] < pivot \rightarrow j=j+1, swap(A, i, j)$$

$$i = i+1$$

$$A[i] \geq pivot \rightarrow i=i+1$$

$$A[i] \geq pivot \rightarrow i=i+1$$

$$A[i] < pivot \rightarrow j=j+1, swap(A, i, j)$$

$$i = i+1$$

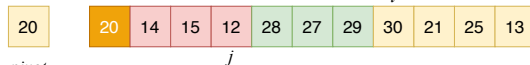
Funzionamento pivot()

pivot



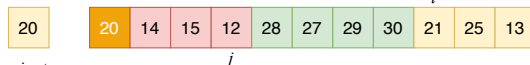
$A[i] \geq \text{pivot} \rightarrow i=i+1$

pivot



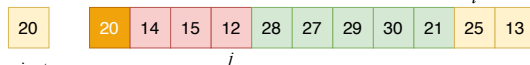
$A[i] < \text{pivot} \rightarrow j=j+1, \text{swap}(A, i, j)$
 $i = i+1$

pivot



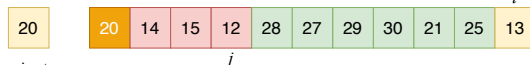
$A[i] \geq \text{pivot} \rightarrow i=i+1$

pivot



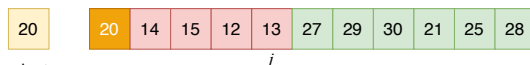
$A[i] \geq \text{pivot} \rightarrow i=i+1$

pivot



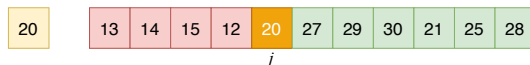
$A[i] \geq \text{pivot} \rightarrow i=i+1$

pivot



$A[i] < \text{pivot} \rightarrow j=j+1, \text{swap}(A, i, j)$
 $i = i+1$

pivot



$A[\text{start}] = A[j]$
 $A[j] = \text{pivot}$

Quicksort – Procedura principale

```
QuickSort(ITEM[] A, int start, int end)
```

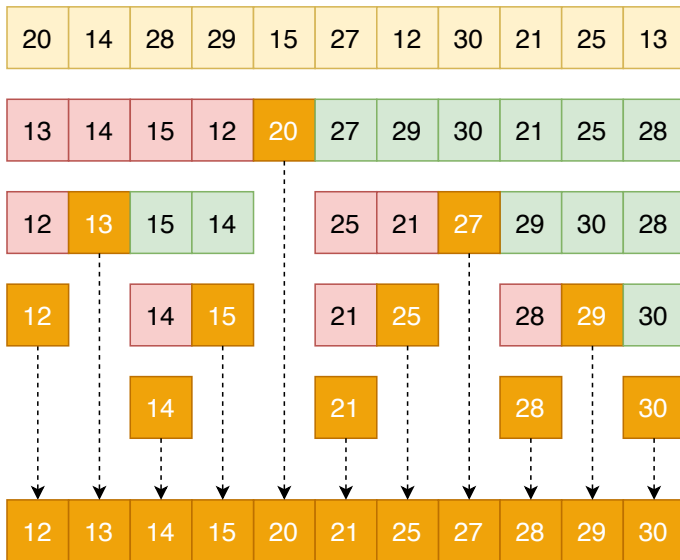
```
if start < end then
```

```
    int j = pivot(A, start, end)
```

```
    QuickSort(A, start, j - 1)
```

```
    QuickSort(A, j + 1, end)
```

Svolgimento ricorsione



Quicksort: Complessità computazionale

Costo di `pivot()`?

Quicksort: Complessità computazionale

Costo di pivot()?

- $\Theta(n)$

Costo Quicksort: caso pessimo?

Quicksort: Complessità computazionale

Costo di `pivot()`?

- $\Theta(n)$

Costo Quicksort: caso pessimo?

- Il vettore di dimensione n viene diviso in due sottovettori di dimensione 0 e $n - 1$
- $T(n) = T(n - 1) + T(0) + \Theta(n) = T(n - 1) + \Theta(n) = \Theta(n^2)$

Costo Quicksort: caso ottimo?

Quicksort: Complessità computazionale

Costo di `pivot()`?

- $\Theta(n)$

Costo Quicksort: caso pessimo?

- Il vettore di dimensione n viene diviso in due sottovettori di dimensione 0 e $n - 1$
- $T(n) = T(n - 1) + T(0) + \Theta(n) = T(n - 1) + \Theta(n) = \Theta(n^2)$

Costo Quicksort: caso ottimo?

- Dato un vettore di dimensione n , viene sempre diviso in due sottoproblemi di dimensione $n/2$
- $T(n) = 2T(n/2) + \Theta(n) = \Theta(n \log n)$

Quicksort: Complessità computazionale

Partizionamenti parzialmente bilanciati

- Il partizionamento nel caso medio di Quicksort è molto più vicino al caso ottimo che al caso peggiore
- Esempio: Partizionamento 9-a-1:

$$T(n) = T(n/10) + T(9n/10) + cn = \Theta(n \log n)$$

- Esempio: Partizionamento 99-a-1:

$$T(n) = T(n/100) + T(99n/100) + cn = \Theta(n \log n)$$

Note

- In questi esempi, il partizionamento ha proporzionalità limitata
- I fattori moltiplicativi possono essere importanti

Quicksort: Complessità computazionale

Caso medio

- Il costo dipende dall'ordine degli elementi, non dai loro valori
- Dobbiamo considerare tutte le possibili permutazioni
- Difficile dal punto di vista analitico

Caso medio: un'intuizione

- Alcuni partizionamenti saranno parzialmente bilanciati
- Altri saranno pessimi
- In media, questi si alterneranno nella sequenza di partizionamenti
- I partizionamenti parzialmente bilanciati “dominano” quelli pessimi

Quicksort: Selezione pivot euristica

Tecnica euristica: selezionare il valore mediano fra il primo elemento, l'ultimo elemento e il valore nella posizione centrale

```

int pivot(ITEM[] A, int start, int end)


---


int m =  $\lfloor (start + end) / 2 \rfloor$ 
if A[start] > A[end] then % Sposta il massimo in ultima posizione
    swap(A, start, end)
if A[m] > A[end] then % Sposta il massimo in ultima posizione
    swap(A, m, end)
if A[m] > A[start] then % Sposta il mediano in prima posizione
    swap(A, m, start)
ITEM pivot = A[start]
[...]
```

<https://xkcd.com/1185/>

```
define JobInterviewQuicksort(list):  
    Ok so you choose a pivot  
    Then divide the list in half  
    for each half:  
        check to see if it's sorted  
        no, wait, it doesn't matter  
    compare each element to the pivot  
        the bigger ones go in a new list  
        the equal ones go into, uh  
        the second list from before  
    hang on, let me name the lists  
        this is list A  
        the new one is list B  
    put the big ones into list B  
    now take the second list  
        call it list, uh, A2  
    which one was the pivot in?  
    scratch all that  
    it just recursively calls itself  
    until both lists are empty  
        right?  
    not empty, but you know what I mean  
    am I allowed to use the standard libraries?
```


Sommario

- 1 Introduzione
- 2 Torri di Hanoi
- 3 Quicksort
- 4 Algoritmo di Strassen**
- 5 Esercizio

Moltiplicazione matrici

$$C = A \times B$$

$$c_{i,j} = \sum_{k=1}^{n_k} a_{i,k} \cdot b_{k,j}$$

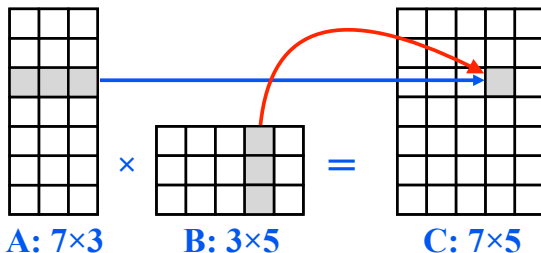
$$\begin{aligned} T(n) &= \Theta(n_i \cdot n_k \cdot n_j) \\ &= \Theta(n^3) \end{aligned}$$

```

matrixProduct(float[][] A, B, C, int n_i, n_k, n_j)


---


for i = 1 to n_i do                                     % Righe
    for j = 1 to n_j do                                 % Colonne
        C[i, j] = 0
        for k = 1 to n_k do
            C[i, j] = C[i, j] + A[i, k] · B[k, j]
        
```



Approccio divide-et-impera

Suddividiamo le matrici $n \times n$ in quattro matrici $n/2 \times n/2$

$$A = \begin{bmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{bmatrix} \quad B = \begin{bmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \end{bmatrix}$$

Calcolo prodotto matrice

$$C = \begin{bmatrix} A_{1,1} \times B_{1,1} + A_{1,2} \times B_{2,1} & A_{1,1} \times B_{1,2} + A_{1,2} \times B_{2,2} \\ A_{2,1} \times B_{1,1} + A_{2,2} \times B_{2,1} & A_{2,1} \times B_{1,2} + A_{2,2} \times B_{2,2} \end{bmatrix}$$

Equazione di ricorrenza

$$T(n) = \begin{cases} 1 & n = 1 \\ 8T(n/2) + n^2 & n > 1 \end{cases}$$

Algoritmo di Strassen

Calcolo elementi intermedi

$$X_1 = (A_{11} + A_{22}) \times (B_{11} + B_{22})$$

$$X_2 = (A_{21} + A_{22}) \times B_{11}$$

$$X_3 = A_{11} \times (B_{12} - B_{22})$$

$$X_4 = A_{22} \times (B_{21} - B_{11})$$

$$X_5 = (A_{11} + A_{12}) \times B_{22}$$

$$X_6 = (A_{21} - A_{11}) \times (B_{11} + B_{12})$$

$$X_7 = (A_{12} - A_{22}) \times (B_{21} + B_{22}).$$

Equazione di ricorrenza

$$T(n) = \begin{cases} 1 & n = 1 \\ 7T(n/2) + n^2 & n > 1 \end{cases}$$

$$T(n) = \Theta(n^{\log_2 7}) \approx \Theta(n^{2.81})$$

Calcolo matrice finale

$$C = \begin{bmatrix} X_1 + X_4 - X_5 + X_7 & X_3 + X_5 \\ X_2 + X_4 & X_1 + X_3 - X_2 + X_6 \end{bmatrix}$$

Moltiplicazione matrici – Panoramica storica

Algoritmo di Strassen (1969)

- $\Theta(n^{2.81})$
- Il primo ad “scoprire” che era possibile moltiplicare due matrici in meno di n^3 moltiplicazioni scalari

Coppersmith and Winograd (1990)

- $O(n^{2.37})$
- Attuale algoritmo migliore
- Fattori moltiplicativi molto alti

Limite inferiore

- $\Omega(n^2)$

Conclusioni

Quando applicare divide-et-impera

- I passi “divide” e “combina” devono essere semplici
- Ovviamente, i costi devono essere migliori del corrispondente algoritmo iterativo
 - Esempio ok: sorting
 - Esempio non ok: ricerca del minimo

Ulteriori vantaggi

- Facile parallelizzazione
- utilizzo ottimale della cache (“cache oblivious”)

Sommario

- 1 Introduzione
- 2 Torri di Hanoi
- 3 Quicksort
- 4 Algoritmo di Strassen
- 5 Esercizio**

Gap

Gap

In un vettore V contenente $n \geq 2$ interi, un **gap** è un indice i , $1 < i \leq n$, tale che $V[i-1] < V[i]$.

- Dimostrare che se $n \geq 2$ e $V[1] < V[n]$, allora V contiene almeno un gap
- Progettare un algoritmo che, dato un vettore V contenente $n \geq 2$ interi e tale che $V[1] < V[n]$, restituisca la posizione di un gap nel vettore.

Gap

Per assurdo:

- Supponiamo che non ci sia un gap nel vettore
- Allora $V[1] \geq V[2] \geq V[3] \geq \dots \geq V[n-1] \geq V[n]$
- Assurdo, perché $V[1] < V[n]$.

Gap – Dimostrazione per induzione

Proviamo a riformulare la proprietà tenendo conto di due indici:

- Sia V un vettore di dimensione n
- Siano i, j due indici tali che $1 \leq i < j \leq n$ e $V[i] < V[j]$

In altre parole, ci sono più di due elementi nel sottovettore $V[i \dots j]$ e il primo elemento $V[i]$ è più piccolo dell'ultimo elemento $V[j]$.

Gap – Dimostrazione per induzione

Vogliamo provare per induzione sulla dimensione n del sottovettore che il sottovettore contiene un gap.

- **Caso base:** $n = j - i + 1 = 2$, i.e. $j = i + 1$:
 $V[i] < V[j]$ implica che $V[i] < V[i + 1]$, che è un gap.
- **Ipotesi induttiva:** dato un qualunque (sotto)vettore $V[h \dots k]$ di dimensione $n' < n$, tale che $V[h] < V[k]$, allora $V[h \dots k]$ contiene un gap
- **Passo induttivo:** consideriamo un qualunque elemento m tale che $i < m < j$. Almeno uno dei due casi seguenti è vero:
 - Se $V[m] < V[j]$, allora esiste un gap in $V[m \dots j]$, per ipotesi induttiva
 - Se $V[i] < V[m]$, allora esiste un gap in $V[i \dots m]$, per ipotesi induttiva

Gap

```
int gap(int[] V, int n)
```

```
return gapRec(V, 1, n)
```

```
int gapRec(int[] V, int i, int j)
```

```
if  $j == i + 1$  then
```

```
    return j
```

```
int  $m = \lfloor (i + j) / 2 \rfloor$ 
```

```
if  $V[m] < V[j]$  then
```

```
    return gapRec(V, m, j)
```

```
else
```

```
    return gapRec(V, i, m)
```

Performance evaluation

n	Iterativa (ms)	Ricorsiva (μ s)
10^3	60.00	2.05
10^4	610.00	2.78
10^5	6 110.00	3.36
10^6	62 440.00	4.01
10^7	621 690.00	4.87
10^8	6 205 720.00	5.47