Relazioni di ricorrenza

Data una funzione ricorsiva, come calcoliamo l'ordine di grandezza?
Esempio:

$$n! = \begin{cases} 1 & \text{se } n = 0 \\ n \cdot (n-1)! & \text{altrimenti} \end{cases}$$

con tempo computazionale:

$$T(n) = \begin{cases} c & \text{se } n = 0 \\ T(n-1) + d & \text{altrimenti} \end{cases}$$

Dunque un algoritmo ricorsivo ha un tempo computazionale **ricorsivo**

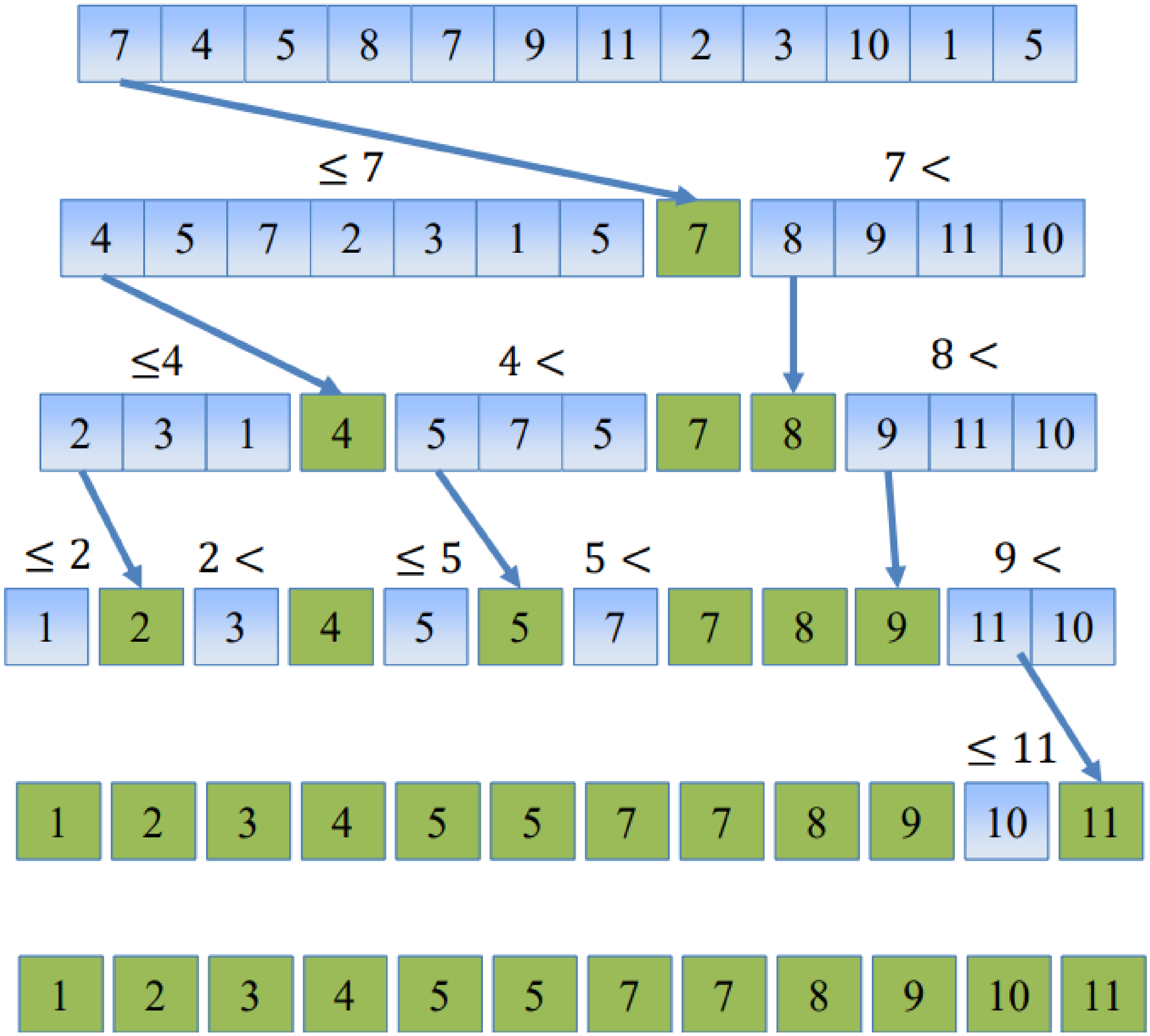
Però qual è esattamente l'ordine di grandezza di $T(n)$?

- **Metodo dell'iterazione:**
  Applichiamo ripetutamente la relazione di ricorrenza fino a trovare la soluzione.
  $T(n) = T(n-1) + d = T(n-2) + d + d = \ldots = T(n-k) + kd = T(0 + n$
  con $k \leq n$ scegliendo $k = n$
- **Metodo di sostituzione:**
  Ipotizziamo inizialmente una soluzione e, induttiviamente, la verifichiamo
  Soluzione ipotizzata: $T(n) = c + nd$
  Caso base: $c + 0 \cdot d = c = T(0)$
  Passo induttivo: dimostriamo che
  - $T(n) = c + nd \rightarrow T(n+1) + c + (n+1)d$
  - Secondo la relazione di ricorrenza:
    $T(n+1) = T(n) + d =$
  - Utilizzando l'ipotesi induttiva:
    $= c + nd + d = c + (n+1)d$

Quick-Sort

Idea di base dato A[1..n]:

- se $n \leq 1$ il caso è banale e non faccio niente(vettore ordinato)
- scelgo un elemento del vettore q, chiamato **perno**
- riorganizzo il vettore in modo che
  - gli elementi a sinistra di q siano $\leq q$
  - gli elementi a destra di q siano $\geq q$
    - **Attenzione:** gli elementi a sinistra e a destra possono essere anche in disordine, basta soltanto che siano $\leq$(o $\geq$) di q
- ciò implica che q sia al posto giusto
- sia p la posizione di q, abbiamo dunque: $A[1..p-1] \leq A[p] \leq A[p+1..n]$
- reitero il procedimento su $A[1..p-1]$ e $A[p+1..n]$



Funzionamento del Quick-Sort con partizionamento generico

Nota bene! Con A[1..n] si intende dal primo elemento(A[0] all'ultimo elemento (A[n-1])

Partizionamento del Quick-Sort

Partizionamento in $A[1..n]$*(Attenzione alla nota sopra!)*

- Scegliamo come elemento **perno** $A[1]$
- Scegliamo due **indici di partizionamento** i e j tali che:

- Gli elementi in $A[2..i-1]$ sono già esaminati e $A[2..i-1] \leq A[1]$
- Gli elementi in $A[i..j]$ sono elemento ancora da esaminare
- Gli elementi in $A[j+1..n]$ sono già esaminati e $A[1] < A[j+1..n]$

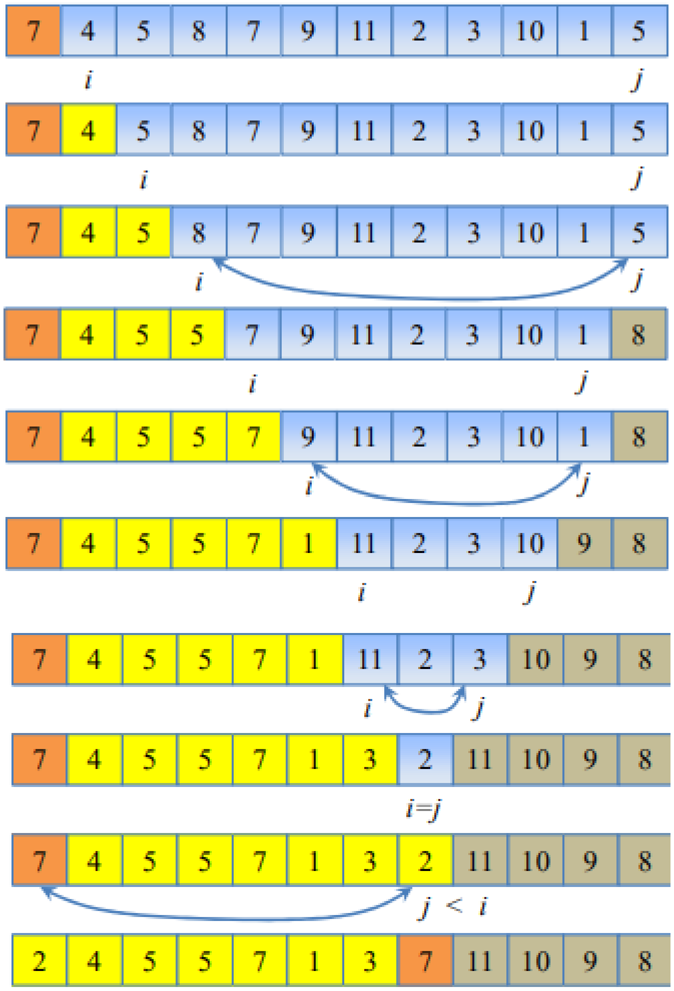
- Scegliamo $i = 2$ e $j = n$

- i e j si spostano, se $i \leq j$, nel seguente modo:

  - se $A[i] \leq A[1]$ incremento i
  - se $A[i] > A[1] \wedge A[j] > A[1]$ decremento j
  - se $A[i] > A[1] \wedge A[j] \leq A[1]$ scambio $A[i]$ e $A[j]$ e incremento i e decremento j

- Quando per la prima volta $i > j$, scambio $A[1]$ con $A[j]$

In questo modo,arriverò a un punto dove gli elementi prima di i sono minori del perno e gli elementi dopo di j sono maggiori del perno. Quando i "supera" j scambio il perno con $A[j]$ così da render vera la prima fase



Esempio di funzionamento di partizionamento nel quicksort

Correttezza del partizionamento

```
PARTITION(A[1..n])
i ← 2, j ← n
while i ≤ j do
    if A[i] ≤ A[1] then
        i ← i + 1
    else
        if A[j] > A[1] then
            j ← j − 1
        else
            scambia A[i] con A[j]
            i ← i + 1, j ← j − 1
        end if
    end if
end while
scambia A[1] con A[j]
return j
```

Invariante di ciclo: $A[2..\, i-1] \leq A[1] \wedge A[1] < A[j+1..\, n]$

Ovvero tutti gli elementi prima di i sono minori dell'elemento perno e tutti gli elementi dopo j sono maggiori dell'elemento perno

Inizializzazione:
Con $i = 2$ e $j = n$ i due sottovettori sono vuoti. Dunque è vero

Mantenimento:
Il ciclo viene eseguito finchè $i \leq j$ ed effettua **soltanto** una tra le seguenti operazioni:

- se $A[i] \leq A[1]$ incremento i altrimenti
- se $A[1] < A[j]$ decremento j altrimenti
- scambio $A[i]$ e $A[j]$, incremento i, decremento j

ed è evidente(visto che esegue soltanto una di quelle operazioni) che il ciclo viene **mantenuto**

Correttezza del Quick Sort

```
QUICK-SORT(A[1..n])
if n > 1 then
    p ← PARTITION(A[1..n])      ▷ partizionamento porta il perno nella posizione p
    if p > 2 then               ▷ se prima del perno ci sono almeno 2 elementi
        QUICK-SORT(A[1..p − 1])
    end if
    if p < n − 1 then           ▷ se dopo il perno ci sono almeno 2 elementi
        QUICK-SORT(A[p + 1..n])
    end if
end if
```

Essendo un algoritmo **ricorsivo**, per dimostrare la sua correttezza bisogna usare una induzione completa, assumendo che il partizionamento funzioni correttamente.

Caso base: con $n \leq 1$ il vettore in input è **ordinato**
Passo induttivo:
HP: Corretto con dimensione $< n$ ⇒ Corretto con dimensione $= n$

- Dopo aver effettuato la partizione:
  $A[1..p-1] \leq A[p] < A[p+1..n]$
- Dall'ipotesi induttiva:
  - Se $A[1..p-1]$ ha più di 1 elemento, sarà ordinato correttamente alla prima chiamata ricorsiva di Quick-Sort perchè ha meno di n elementi
  - Se $A[p+1..n]$ ha più di 1 elemento, sarà ordinato correttamente alla prima chiamata ricorsiva di Quick-Sort perchè ha meno di n elementi
  $A[1..n]$ è dunque ordinato

Complessità del Quick-Sort

- **Partizionamento:**
  Il partizionamento scansiona una volta il vettore su cui opera.
  Nel caso peggiore, $T_p(n) \in O(n)$ infatti $T_p(n) = an$

- Per quanto riguarda il **Quick-Sort**:
  Dopo aver partizionato i vettori, le due chiamate ricorsive lavorano su $p-1$ elementi e $n-p$ elementi

- Le due situazioni estreme sono:

  - $A[1..p-1]$ e $A[p+1..n]$ hanno circa lo stesso numero di elementi
  - $A[1..p-1]$ ha $n-1$ elementi e $A[p+1..n]$ è vuoto(o viceversa)

La seconda casistica dà luogo alla seguente relazione di ricorrenza:

$$T(n) = \begin{cases} c & n = 1 \\ T(n-1) + T_P(n) + b & n > 1 \end{cases} = \begin{cases} c & n = 1 \\ T(n-1) + an + b & n > 1 \end{cases}$$