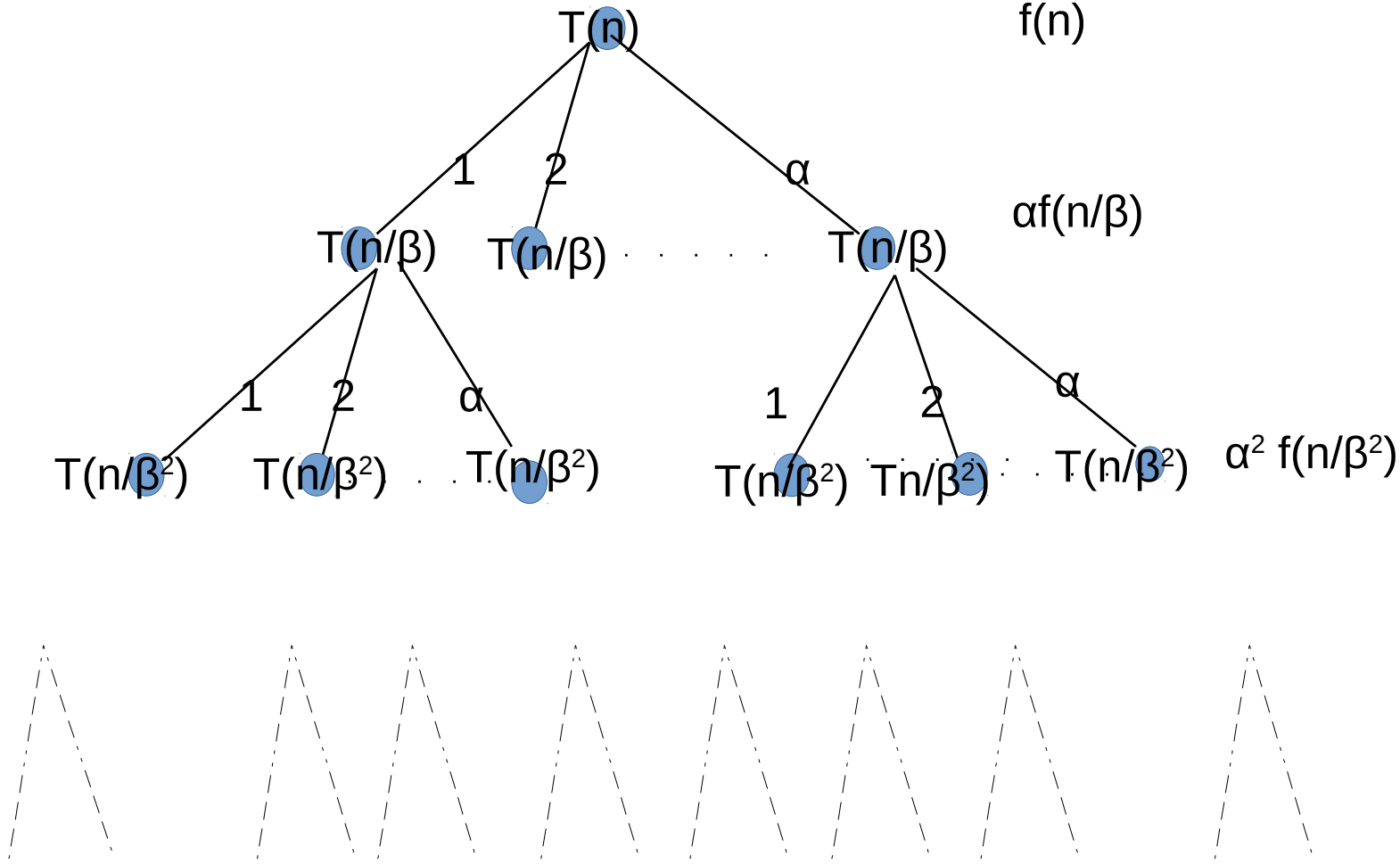
Soluzione delle relazioni di ricorrenza

- Consideriamo la seguente relazione di ricorrenza:

$$T(n) \leq \begin{cases} c_0 & \text{se } n \leq c \\ \alpha T(n/\beta) + f(n) & \text{altrimenti} \end{cases}$$

  con $\alpha \geq 1$ e $\beta > 1$ costanti.
- Nel caso in cui $T(n)$ sia la funzione che scaturisce dall'analisi di un algoritmo basato sul paradigma del Divide et Impera, $f(n)$ è il tempo per il lavoro di suddivisione e di ricombinazione. In altre parole, $f(n) = d(n) + r(n)$.
- In realtà nella ricorrenza n/β dovrebbe essere $\lceil n/\beta \rceil$ oppure $\lfloor n/\beta \rfloor$. Per stimare $T(n)$, assumiamo per semplicità che n sia una potenza di β in modo da poter omettere le parti intere superiori o inferiori.

Soluzione delle relazioni di ricorrenza



Sia h l'altezza dell'albero (h+1 livelli). Per i<h, Il tempo di esecuzione per tutte le chiamate ricorsive a livello i e` al piu` $\alpha^i f(n/\beta^i)$.

Soluzione delle relazioni di ricorrenza

- L'algoritmo non effettua chiamate ricorsive quando l'input ha dimensione al più c. Quindi la ricorrenza non sarà più applicata quando si arriva al livello i per cui per la prima volta $n/\beta^i \leq c$, cioè $i = \lceil \log_\beta n/c \rceil$.

- Il numero di livelli dell'albero è quindi $\lceil \log_\beta n/c \rceil + 1$ e ciascun nodo sul livello $\lceil \log_\beta n/c \rceil$ corrisponde al tempo $T(n/\beta^{\lceil \log_\beta n/c \rceil}) \leq T(c) \leq c_0$. Il tempo totale per eseguire le $\alpha^{\lceil \log_\beta n/c \rceil}$ chiamate ricorsive in quest'ultimo livello è quindi $\leq \alpha^{\lceil \log_\beta n/c \rceil} c_0$.

- Abbiamo visto che per $i < \lceil \log_\beta n/c \rceil$, il tempo per eseguire tutte le chiamate sul livello i è $\alpha^i f(n/\beta^i)$.

- Sommando su tutti i livelli si ha

$$T(n) \leq \alpha^{\lceil \log_\beta n/c \rceil} c_0 + \sum_{i=0}^{\lceil \log_\beta n/c \rceil - 1} \alpha^i f(n/\beta^i).$$

Soluzione delle relazioni di ricorrenza

- Vogliamo stimare la funzione

$$T(n) \leq \begin{cases} c_0 & \text{se } n \leq c \\ \alpha T(n/\beta) + f(n) & \text{altrimenti} \end{cases}$$

  quando la funzione $f(n)$ è limitata da $c' n^k$, dove c' e k sono due costanti tali che $k \geq 0$, $c' > 0$ ($f(n)$ polinomiale).

- Da quanto ottenuto nella slide precedente si ha che

$$\begin{aligned} T(n) &\leq \alpha^{\lceil \log_\beta n/c \rceil} c_0 + \sum_{i=0}^{\lceil \log_\beta n/c \rceil - 1} \alpha^i f(n/\beta^i) \\ &\leq \alpha^{\lceil \log_\beta n/c \rceil} c_0 + \sum_{i=0}^{\lceil \log_\beta n/c \rceil - 1} \alpha^i c' (n/\beta^i)^k \end{aligned}$$

Soluzione delle relazioni di ricorrenza

- Abbiamo visto che se $f(n) \leq c' n^k$, dove c' e k sono due costanti tali che $k \geq 0$, $c' > 0$, allora

$$T(n) \leq \alpha^{\lceil \log_\beta n/c \rceil} c_0 + c' n^k \sum_{i=0}^{\lceil \log_\beta n/c \rceil - 1} (\alpha/\beta^k)^i.$$

- consideriamo i 2 seguenti casi:
- $\alpha = \beta^k$: In questo caso si ha

$$\alpha^{\lceil \log_\beta n/c \rceil} c_0 = (\beta^k)^{\lceil \log_\beta n/c \rceil} c_0 < (\beta^k)^{\log_\beta (n/c)+1} c_0 = \beta^k (n/c)^k c_0 = O(n^k)$$

e

$$c' n^k \sum_{i=0}^{\lceil \log_\beta n/c \rceil - 1} (\alpha/\beta^k)^i = c' n^k \sum_{i=0}^{\lceil \log_\beta n/c \rceil - 1} 1 = c' n^k \lceil \log_\beta n/c \rceil = O(n^k \log_\beta n).$$

Quindi $T(n) = O(n^k) + O(n^k \log_\beta n) = O(n^k \log_\beta n) = O(n^k \log n)$.

Soluzione delle relazioni di ricorrenza

- $\alpha \neq \beta^k$: In questo caso si ha

$$\begin{aligned} \alpha^{\lceil \log_\beta n/c \rceil} c_0 &< c_0 \alpha^{\log_\beta n/c + 1} = c_0 \alpha \cdot \alpha^{\log_\beta n/c} = c_0 \alpha \cdot \alpha^{\log_\alpha (n/c) \log_\beta \alpha} \\ &= c_0 \alpha (n/c)^{\log_\beta \alpha} = O(n^{\log_\beta \alpha}), \end{aligned} \tag{1}$$

e

$$c' n^k \sum_{i=0}^{\lceil \log_\beta n/c \rceil - 1} (\alpha/\beta^k)^i = c' n^k \cdot \frac{(\alpha/\beta^k)^{\lceil \log_\beta n/c \rceil} - 1}{(\alpha/\beta^k) - 1}. \tag{2}$$

Soluzione delle relazioni di ricorrenza

Consideriamo i due sottocasi di $\alpha \neq \beta^k$: $\alpha < \beta^k$ e $\alpha > \beta^k$

- Caso $\alpha < \beta^k$:

$$\begin{aligned} \frac{(\alpha/\beta^k)^{\lceil \log_\beta (n/c) \rceil} - 1}{(\alpha/\beta^k) - 1} &= \frac{1 - (\alpha/\beta^k)^{\lceil \log_\beta n/c \rceil}}{1 - (\alpha/\beta^k)} \\ &< \frac{1}{1 - (\alpha/\beta^k)} = \frac{\beta^k}{\beta^k - \alpha} = O(1). \end{aligned}$$

Si ha quindi che la suddetta relazione insieme alla (1) e alla (2) della slide precedente implicano:

$$T(n) \leq O(n^{\log_\beta \alpha}) + c' n^k O(1) = O(n^{\log_\beta \alpha} + n^k).$$

Si noti che $\alpha < \beta^k$ implica $\log_\beta \alpha < k$ e di conseguenza si ha

$$T(n) = O(n^{\log_\beta \alpha} + n^k) = O(n^k).$$

Soluzione delle relazioni di ricorrenza

- Caso $\alpha > \beta^k$:

$$\begin{aligned}
\frac{(\alpha/\beta^k)^{\lceil \log_\beta(n/c)\rceil} - 1}{(\alpha/\beta^k) - 1} &< \frac{(\alpha/\beta^k)^{\log_\beta(n/c)+1} - 1}{(\alpha/\beta^k) - 1} = \frac{(\alpha/\beta^k)(\alpha/\beta^k)^{\log_\beta(n/c)} - 1}{(\alpha/\beta^k) - 1} \\
&= O((\alpha/\beta^k)^{\log_\beta(n/c)}) = O((\alpha/\beta^k)^{\log_{\alpha/\beta^k}(n/c)\log_\beta(\alpha/\beta^k)}) \\
&= O((n/c)^{\log_\beta(\alpha/\beta^k)}) = O((n/c)^{\log_\beta \alpha - \log_\beta \beta^k}) \\
&= O(n^{\log_\beta(\alpha)-k})
\end{aligned}$$

Si ha quindi che la suddetta relazione insieme alla (1) e alla (2) della slide precedente implicano:

$$T(n) \leq O(n^{\log_\beta \alpha}) + c' n^k O(n^{\log_\beta(\alpha)-k}) = O(n^{\log_\beta \alpha} + n^k n^{\log_\beta(\alpha)-k}) = O(n^{\log_\beta \alpha}).$$

Soluzione delle relazioni di ricorrenza

- Abbiamo stimato la funzione

$$T(n) \leq \begin{cases} c_0 & \text{se } n \leq c \\ \alpha T(n/\beta) + c' n^k & \text{altrimenti,} \end{cases}$$

  dove c' e k sono due costanti tali che $k \geq 0$, $c' > 0$.

- Abbiamo provato

$$T(n) = \begin{cases} O(n^k) & \text{se } \alpha < \beta^k \\ O(n^k \log n) & \text{se } \alpha = \beta^k \\ O(n^{\log_\beta \alpha}) & \text{se } \alpha > \beta^k \end{cases}$$

- **Esempi:** Nel caso di MergeSort $\alpha = 2$, $\beta = 2$ e $k = 1$. Si ha $\alpha = \beta^k$ e quindi $T(n) = O(n^k \log n) = O(n \log n)$.
  Nel caso dell'algoritmo per la ricerca binaria $\alpha = 1$, $\beta = 2$ e $k = 0$. Si ha $\alpha = \beta^k$ e quindi $T(n) = O(n^k \log n) = O(\log n)$.

Ordinamento per distribuzione

L'algoritmo di ordinamento per distribuzione (*quicksort*) opera nel modo seguente.

Decomposizione: se la sequenza ha almeno due elementi, scegli un elemento **pivot** e dividi la sequenza in due sotto-sequenze in modo tale che la prima contenga elementi minori o uguali al pivot e la seconda gli elementi maggiori o uguali del pivot.

Ricorsione: ordina ricorsivamente le due sotto-sequenze.

Ricombinazione: non occorre fare alcun lavoro.

```
1  QuickSort( a, sinistra, destra ):
2
3    IF (sinistra < destra) {
4      scegli pivot nell'intervallo [sinistra...destra];
5      indiceFinalePivot = Distribuzione(a, sinistra, pivot, destra);
6      QuickSort( a, sinistra, indiceFinalePivot-1 );
7      QuickSort( a, indiceFinalePivot+1, destra );
8    }
```

Distribuzione

- Data la posizione `px` del pivot in un segmento $a[\texttt{sx}, \texttt{dx}]$:
  - scambia gli elementi $a[\texttt{px}]$ e $a[\texttt{dx}]$, se $\texttt{px} \neq \texttt{dx}$
  - usa due indici `i` e `j` per scandire il segmento: `i` parte da `sx` e va verso destra e `j` parte da $\texttt{dx} - 1$ e va verso sinistra fino a quando $\texttt{i} \leq \texttt{j}$
  - ogni volta che si ha $a[\texttt{i}] > \texttt{pivot}$ e $a[\texttt{j}] < \texttt{pivot}$, scambia $a[\texttt{i}]$ con $a[\texttt{j}]$ e poi riprende la scansione
  - alla fine della scansione posiziona il pivot nella sua posizione corretta

Ordinamento per distribuzione

```
1  Distribuzione( a, sx, px, dx ):
2    IF (px != dx) Scambia( px, dx );
3    i = sx;
4    j = dx-1;
5    WHILE (i <= j) {
6      WHILE ((i <= j) && (A[i] <= A[dx]))
7        i = i+1;
8      WHILE ((i <= j) && (A[j] => A[dx]))
9        j = j-1;
10     IF (i < j) Scambia( i, j ); i=i+1,j=j-1;
11   }
12   IF (i != dx) Scambia( i, dx );
13   RETURN i;
```

```
1  Scambia( i, j ):                                  ⟨pre: sx ≤ i, j ≤ dx⟩
2    temp = a[j]; a[j] = a[i]; a[i] = temp;
```

Analisi di Distribuzione

1. per stimare il tempo richiesto dal while esterno dobbiamo stimare il numero di iterazioni eseguite complessivamente dei due while interni.
2. numero totale di iterazioni del primo while interno = numero confronti tra un elemento $a[\texttt{i}]$ con il pivot
3. numero totale di iterazioni del secondo while interno = numero confronti tra un elemento $a[\texttt{j}]$ con il pivot
4. dopo ogni confronto di $a[\texttt{i}]$ con il pivot o viene incrementato `i` (nel while stesso o nell'if) o il while esterno termina dopo al più un'iterazione
5. dopo ogni confronto di $a[\texttt{j}]$ con il pivot o viene decrementato `j` (nel while stesso o nell'if) o il while esterno termina dopo al più un'iterazione
6. per i due punti precedenti si ha che ad ogni iterazione di ciascuno dei while interni, fatta al più eccezione per l'ultima, viene confrontato un nuovo elemento con il pivot. Inoltre, siccome $i \leq j$, vi è un unico elemento che può essere confrontato in entrambi i while interni (se nell'ultima iterazione $i = j$ e $a[i] > a[dx]$).
7. il numero totale di confronti con il pivot è quindi al più $n + 2$ e quindi il numero totale di iterazioni dei due while interni è complessivamente al più $n + 2$ (in realtà è al più $n + 1$ perchè si può vedere che al più uno dei due while interni può testare nell'ultima iterazione lo stesso elemento dell'iterazione precedente)
8. tempo $O(n)$

Analisi di QuickSort mediante relazione di ricorrenza

Relazione di ricorrenza per il tempo $T(n)$ di esecuzione dell'algoritmo.

- Caso base: $T(n) \leq c_0$ per $n \leq 1$.
- Passo ricorsivo: sia r il rango dell'elemento pivot. Ci sono $r-1$ elementi a sinistra del pivot e $n-r$ elementi a destra, per cui $T(n) \leq T(r-1) + T(n-r) + cn$.

Analisi di QuickSort mediante relazione di ricorrenza

caso pessimo

- Il pivot è tutto a sinistra ($r = 1$) oppure tutto a destra ($r = n$). In entrambi i casi, la relazione diventa $T(n) \leq T(n-1) + T(0) + cn \leq T(n-1) + c'n$ per un'opportuna costante c'
- Applichiamo iterativamente la relazione di ricorrenza:

$$T(n) \leq T(n-1)+c'n \leq T(n-2)+c'(n-1)+c'n \leq \ldots \leq T(n-i)+\sum_{j=0}^{i-1} c'(n-j).$$

- Sostitutendo $i = n-1$ nell'espressione più a destra, otteniamo

$$T(n) \leq T(1)+\sum_{j=0}^{n-2} c'(n-j) \leq c_0+\sum_{j=2}^{n} c'j \leq c_0+c'(n+1)n/2-c' = O(n^2),$$

Analisi di QuickSort mediante relazione di ricorrenza

caso ottimo

- La distribuzione è bilanciata ($r = n/2$)), la ricorsione avviene su ciascuna metà
- In questa situazione, il costo è simile a quella dell'ordinamento per fusione.
- Possiamo dimostrare che il costo è di $O(n \log n)$ tempo

Efficienza del QuickSort randomizzato: intuizione

- Affinché QuickSort abbia tempo di esecuzione $O(n \log n)$ non è necessario che ogni volta il pivot sia l'elemento centrale ma è sufficiente che una frazione costante degli elementi risulti minore o uguale del pivot.
- Sia m la dimensione del segmento di array da ordinare in una certa chiamata ricorsiva. Supponiamo che il segmento venga suddiviso in due segmenti (escluso il pivot) di dimensione rispettivamente pari circa a $(m-1)(\frac{1}{d})$ e $(m-1)(1-\frac{1}{d})$, con $d > 1$ costante. Diciamo "circa" perché in realtà per un segmento occorre prendere la parte intera superiore e per l'altra quella inferiore.
- Ovviamente quanto più sono diverse le lunghezze dei due segmenti (d molto piccolo o molto grande) tanto peggiore è il comportamento dell'algoritmo.
- Supponiamo che la chiamata ricorsiva in cui la suddivisione risulta più sbilanciata, suddivida il segmento da ordinare (privato del pivot) in due parti di dimensione pari rispettivamente a circa $\frac{1}{\beta}$ e $1-\frac{1}{\beta}$ della dimensione del segmento, dove β è una costante positiva.
- Il tempo richiesto è sicuramente non più grande di quello che sarebbe richiesto se una tale suddivisione si verificasse per ogni chiamata ricorsiva su input maggiori di β. Per input di dimensione minore o uguale di β ci mettiamo nel caso peggiore, cioè quello in cui un segmento è vuoto e l'altro contiene tutti gli elementi diversi dal pivot. Il tempo sarà comunque limitato da una costante che indichiamo con c_1.

Efficienza del QuickSort randomizzato: intuizione

- Omettiamo le parti intere inferiori e superiori. Si può dimostrare che ciò non influisce sul comportamento asintotico della ricorrenza.
- Consideriamo quindi la relazione di ricorrenza

$$T(n) \leq \begin{cases} T((n-1)/\beta) + T((n-1)(1-1/\beta)) + cn & \text{se } n > \beta \\ c_1 & \text{per } n \leq \beta \end{cases}$$

  Vogliamo dimostrare che questa relazione di ricorrenza ha soluzione $O(n \log n)$ per qualsiasi costante $\beta > 1$.
- Possiamo assumere che $\beta \neq 2$ in quanto abbiamo già visto che in quel caso $T(n) = O(n \log n)$. Possiamo inoltre assumere senza perdere di generalità che $\beta > 2$, cioè che il primo segmento sia più piccolo del secondo.
- Dimostriamo con il metodo della sostituzione che $T(n) = O(n \log n)$. Per far ciò dimostreremo per induzione che esiste una costante $c' > 0$ per cui $T(n) \leq c' n \log n$ per ogni $n \geq \beta$.
- Base induzione: per $n = \beta$, si ha $T(\beta) \leq c_1$. Perché sia $T(\beta) \leq c'(\beta \log \beta)$ basta quindi scegliere c' tale che $c' \geq c_1/(\beta \log \beta)$.

Efficienza del QuickSort randomizzato: intuizione

- Passo induttivo. Supponiamo vera la disuguaglianza per $2, \ldots, n-1$. Si ha

$$
\begin{aligned}
T(n) &\leq T((n-1)/\beta) + T((n-1)(1-1/\beta)) + cn \\
&\leq c'((n-1)/\beta)\log((n-1)/\beta) + c'((n-1)(1-1/\beta)\log((n-1)(1-1/\beta)) + cn \\
&\leq c'(n/\beta)\log(n/\beta) + c'n(1-1/\beta)\log(n(1-1/\beta)) + cn \\
&= c'(n/\beta)(\log(n/\beta) - \log(n(1-1/\beta))) + c'n\log(n(1-1/\beta)) + cn \\
&= -c'(n/\beta)\log(\beta-1) + c'n\log(n(1-1/\beta)) + cn \\
&\leq -c'(n/\beta)\log(\beta-1) + c'n\log(n-1) + cn.
\end{aligned}
$$

- Perché risulti $T(n) \leq c'n\log n$ basta imporre $-(c'/\beta)\log(\beta-1) + c \leq 0$ che è soddisfatta per $c' \geq c\beta/(\log(\beta-1))$
- Quindi dobbiamo scegliere

$$c' = \max\{c_1/(\beta\log\beta), c\beta/(\log(\beta-1))\}.$$

Efficienza del QuickSort randomizzato: intuizione

- Ci sono quindi molte possibili scelte del pivot che fanno in modo che l'algoritmo si comporti bene.
- Questo ci suggerisce che scegliere il pivot in modo random (con distribuzione di probabilità uniforme) porta con buona probabilità a scegliere un pivot "ben posizionato" e cioè un pivot che suddivide il segmento da ordinare nel modo descritto in precedenza e ad avere un tempo di esecuzione $O(n \log n)$.
- Si può dimostrare formalmente che il QuickSort randomizzato ha tempo di esecuzione medio $O(n \log n)$.