UNIVERSITA' DEGLI STUDI DI
NAPOLI FEDERICO II

Scuola Politecnica e delle Scienze di Base
Corso di Laurea Magistrale in Ingegneria Informatica

Algoritmi e strutture dati:

Homework 1

Anno Accademico 2023/2024

Salvatore Santoro M63001420
Andrea Russo M63001634

Esercizio 1.1. Notazione asintotica

Traccia:

Per ognuna delle seguenti affermazioni, si dica se essa è **sempre vera**, **mai vera**, o **a volte vera** , per funzioni asintoticamente non-negative. Se la si considera sempre vera o mai vera, si spieghi il perché. Se è a volte vera, si dia un esempio per cui è vera e uno per cui è falsa.

1. $f(n) = O(f(n)^2)$
2. $f(n) + O(f(n)) = \Theta(f(n))$
3. $f(n) = \Omega(g(n))$ e $f(n) = o(g(n))$ - Nota la notazione little-o

Soluzioni:

1. **A volte vera**. Sotto ipotesi di funzione asintoticamente non negativa, la definizione di O-grande $f(n) \leq cf(n)^2$ è verificata $\forall c > 0$ solamente se la funzione non è asintoticamente compresa tra 0 e 1, Un esempio di funzione che non verifica la definizione precedente è e^{-n}, un esempio di funzione che la verifica è n.

2. **Sempre vera**. Applicando la definizione di O-grande abbiamo:
   $\exists\, c : f(n) \leq f(n) + O(f(n)) \leq f(n) + cf(n) \qquad \forall n > n_0$
   da questo risultato è facile verificare la definizione di Θ dato che:
   $d_1 f(n) \leq f(n)$ e $f(n) + cf(n) \leq d_2 f(n) \quad$ per $\quad d_1 \leq 1$ e $d_2 \geq (1 + c)$

3. **Mai vera**. Basta applicare la definizione di Ω e di o-piccolo per giungere ad un assurdo.
   $\exists\, c_1 : f(n) \geq c_1 g(n) \quad \forall n > n_1 \quad$ e $\quad \forall\, c_2 > 0\, \exists\, n > n_2 \,:\, f(n) < c_2 g(n)$
   in pratica dato che la definizione di o-piccolo deve essere valida per ogni costante positiva, dovrà essere verificata anche per la costante che troviamo applicando la definizione di Ω, stiamo quindi dicendo che per la stessa costante si ha contemporaneamente $f(n) \geq cg(n)$ e $f(n) < cg(n)$.

Esercizio 1.2. Notazione asintotica e crescita delle funzioni

Traccia:

Per ognuna delle seguenti coppie di funzioni $f(n)$ e $g(n)$, trovare una appropriata costante positiva c tale che $f(n) \leq c * g(n)$ per tutti i valori di $n > 1$.

1. $f(n) = n^2 + n + 1$, $g(n) = 2n^3$
2. $f(n) = n\sqrt{n} + n^2$, $g(n) = n^2$
3. $f(n) = n^2 - n + 1$, $g(n) = \frac{n^2}{2}$

Soluzioni:

1. $n^2 + n + 1 \leq 2\,c\,n^3$
   $c \geq \dfrac{1}{2n} + \dfrac{1}{2n^2} + \dfrac{1}{2n^3}$
   dato che il valore più piccolo che n può assumere è 2 allora necessariamente $c \geq \dfrac{7}{16}$
2. $n\sqrt{n} + n^2 \leq c\,n^2$
   $c \geq \dfrac{\sqrt{n}}{n} + 1$ il rapporto per $n \to \infty$ va a 0 perché il denominatore è un infinito più grande del numeratore quindi ci basta determinare solo il limite più stretto $(n = 2)$ ovvero $c \geq \dfrac{\sqrt{2}}{2} + 1$
3. $n^2 - n + 1 \leq \dfrac{cn^2}{2}$
   $c \geq 2 - \dfrac{2}{n} + \dfrac{2}{n^2}$
   la somma dei termini dipendenti da n è sempre < 2 ed è sempre una quantità negativa, basta portare le frazioni allo stesso denominatore per vederlo:
   $\dfrac{-2n + 2}{n^2}$, ne deriva che il limite più stretto è $c \geq 2 \quad \forall n > 1$

Esercizio 1.3. Complessità

Traccia:

Dimostrare che per qualsiasi costante reale a e b con $b > 0$, $(n + a)^b = \Theta(n^b)$.

Soluzione:

Supponiamo $n \geq |a|$ (a è una costante fissata)
$(n + a)^b = n^b(1 + \frac{a}{n})^b$ per il termine $\frac{a}{n}$ abbiamo 2 alternative quando n va all'infinito:
$\frac{a}{n} \to 0^+$ se $a > 0$
$\frac{a}{n} \to 0^-$ se $a < 0$
tenendo questo a mente dimostriamo separatamente che vale la definizione di O e di Ω per dimostrare l'assunto:

1. $n^b(1 + \frac{a}{n})^b \leq d_1 n^b$
   basta prendere $d_1 \geq (1 + \varepsilon)^b$ con $\varepsilon > 0$ e sufficientemente grande in valore assoluto rispetto a $\frac{a}{n}$. Dato che $(1+\varepsilon)^b$ è il termine più stringente da verificare ($a > 0$) perché che se a fosse negativa il termine che moltiplica n^b sarebbe sicuramente minore di 1.

2. $d_2 n^b \leq n^b(1 + \frac{a}{n})^b$
   basta prendere $d_2 \leq (1 - \varepsilon)^b$ con $\varepsilon > 0$ e sufficientemente grande in valore assoluto rispetto a $\frac{a}{n}$. Dato che $(1 - \varepsilon)^b$ è il termine più stringente da verificare
   ($a < 0$) perché che se a fosse positiva il termine che moltiplica n^b sarebbe sicuramente maggiore di 1.

Da notare che non sono state fatte assunzioni particolari sul termine b visto che ci basta la sola ipotesi di positività. Gli unici casi di interesse potrebbero essere b pari oppure b dispari, ma ciò se si considera n all'infinito è ininfluente perché i termini in parentesi che eleviamo alla b risultano essere sempre positivi.

Esercizio 1.4. Ricorrenze

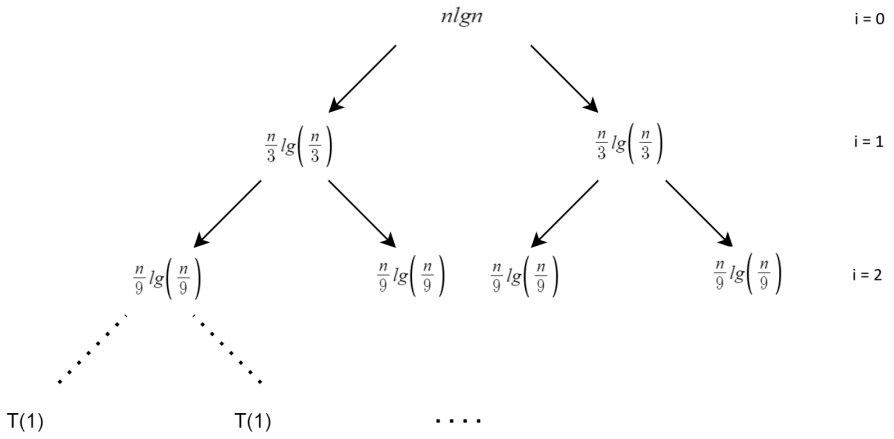
Traccia:

Fornire il limite inferiore e superiore per $T(n)$ nelle seguenti ricorrenze, usando il metodo dell'albero delle ricorrenze ed il teorema dell'esperto se applicabile. Si fornisca il limite più stretto possibile giustificando la risposta.

1. $T(n) = 2T\left(\frac{n}{3}\right) + n \log n$
2. $T(n) = 3T\left(\frac{n}{5}\right) + \log^2 n$

Soluzioni:

1. la funzione ricorsiva non avendo l'operatore floor è definita solo per n potenza di 3, quindi l'albero di ricorrenza è applicabile senza effettuare approssimazioni. Supponiamo $T(1) = \Theta(1)$. Sviluppando l'albero di ricorrenza si può vedere che il numero di livelli è $\log_3 n + 1$, il numero di nodi in ogni livello è 2^i e il costo a sottoproblema è $\frac{n}{3^i} \log \frac{n}{3^i}$. Guardando il costo a sottoproblema si può già dire che un limite inferiore per la ricorrenza è proprio $n \log n$ che è il costo per risolvere il primo problema(somma dei costi in ogni livello decrescente), quindi $T(n) = \Omega(n \log n)$.



Per quanto riguarda la dimostrazione di O basta sommare il numero di sottoproblemi in ogni livello, moltiplicati per il proprio costo e aggiungere eventualmente il

numero di casi base moltiplicati per il loro costo.

$\sum_{i=0}^{\log_3 n-1} \frac{2^i n}{3^i} \log \frac{n}{3^i} + 2^{\log_3 n}\Theta(1) =$

$\sum_{i=0}^{\log_3 n-1} \frac{2^i n}{3^i} (\log n \ - \ \log 3^i) + \Theta(n^{\log_3 2}) =$

$\sum_{i=0}^{\log_3 n-1} \frac{2^i n}{3^i} \log n \ - \sum_{i=0}^{\log_3 n-1} \frac{2^i n}{3^i} \log 3^i + \Theta(n^{0.631}) \leq$ (prima sommatoria ad ∞)

$3n \log n \ - n \log 3 \sum_{i=0}^{\log_3 n-1} \frac{2^i i}{3^i} + \Theta(n^{0.631}) \leq$

$3n \log n \ - n \log 3 \sum_{i=0}^{\infty} \frac{2^i i}{3^i} + \Theta(n^{0.631})$

applichiamo il criterio del rapporto alla seconda serie, sotto ipotesi di serie a termini positivi, e dimostriamo che converge:

$\lim_{n\to\infty} |\frac{a_{n+1}}{a_n}|$ dove a è la ragione della serie.

$\lim_{n\to\infty} \left| \frac{(n+1)\left(\frac{2}{3}\right)^{n+1}}{n\left(\frac{2}{3}\right)^{n}} \right| = \lim_{n\to\infty} \frac{2}{3} \left| \frac{n\left(\frac{2}{3}\right)^{n} + \left(\frac{2}{3}\right)^{n}}{n\left(\frac{2}{3}\right)^{n}} \right| = \lim_{n\to\infty} \frac{2}{3} \left| \frac{n+1}{n} \right| = \frac{2}{3} \leq 1$

quindi la serie converge e il risultato precedentemente ottenuto è

$3n \log n \ - \Theta(n) + \Theta(n^{0.631}) = \Theta(n \log n)$

dato le maggiorazioni possiamo quindi dire che $T(n) = O(n \log n)$.

Dimostriamo la validità della precedentemente affermazione col metodo di sostituzione.

Passo induttivo: supponiamo $T(k) = O(k \log k)$ per $k = \frac{n}{3}$

$T(n) \leq 2\left(\frac{cn}{3} \log \frac{n}{3}\right) \ + \ n \log n = \frac{2}{3} cn (\log n - \log 3) + n \log n \leq cn \log n$

Dividiamo tutto per $n \log n$ (supponiamo $n > 1$) e mettiamo in evidenza i termini dipendenti da c, per ottenere:

$c\left(\frac{1}{3} + \frac{2 \log 3}{3 \log n}\right) \geq 1$

per $n \to \infty$ il secondo termine in parentesi tende a 0, per cui ci basta scegliere una qualsiasi $c \geq 3$ per verificare la disequazione per ogni $n > 1$.

Caso base: dato che il caso base della ricorsione $T(1) = \Theta(1)$ non è $\leq cn \log n$ per $n = 1$ (il logaritmo si annulla), scegliamo come caso base per il metodo di sostituzione $T(3) = 2T(1) + 3 \log 3 = 3.431$ che risulta essere $\leq c3 \log 3 = c \cdot 1.431$ per $c \geq 3$.

Analogamente si può dimostrare che $T(n) = \Omega(n \log n)$:

Passo induttivo: supponiamo $T(k) = \Omega(k \log k)$ per $k = \frac{n}{3}$

$T(n) \geq 2\left(\frac{cn}{3}\log\frac{n}{3}\right) + n\log n = \frac{2}{3}cn\left(\log n - \log 3\right) + n\log n \geq cn\log n.$

Dividiamo tutto per $n\log n$ (supponiamo $n > 1$) e mettiamo in evidenza i termini dipendenti da c, per ottenere:

$c\left(\frac{1}{3} + \frac{2\log 3}{3\log n}\right) \leq 1$

per $n = 2$ il termine in parentesi è $\backsimeq 1.38$, per cui ci basta scegliere una qualsiasi $c \leq \dfrac{1}{1.38} \backsimeq 0.71$ per verificare la disequazione per ogni $n > 1$.

Caso base: il caso base della ricorsione $T(1) = \Theta(1)$ è $\geq cn\log n$ per $n = 1$ (il logaritmo si annulla), $\forall c > 0$ quindi la costante trovata nel passo induttivo risulta essere sufficiente anche per verificare il caso base.

il metodo dell'esperto risulta applicabile perché $f(n) = n\log n$ e

$g(n) = n^{log_3 2} = n^{0.631}$ quindi le funzioni sono polinomialmente comparabili e risulta $f(n) = \Omega(n^{0,63+\epsilon})$ con $\epsilon > 0$. Siamo nel caso 3 del metodo dell'esperto, quindi dimostriamo che $af(n/b) \leq cf(n)$ con $c < 1$.

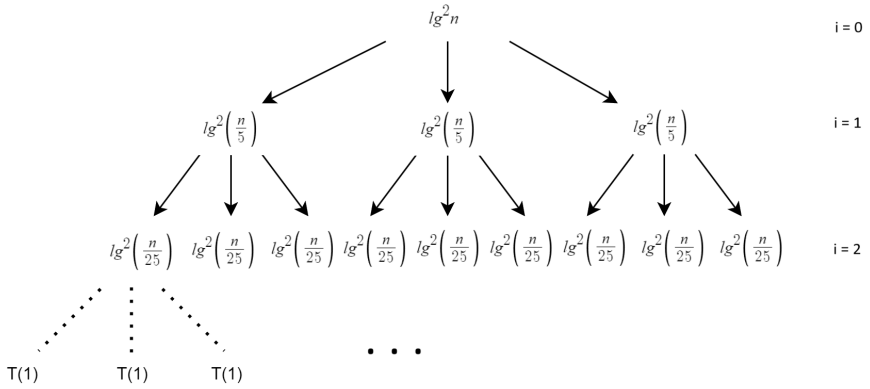
$\frac{2n}{3}\log\left(\frac{n}{3}\right) \leq cn\ \log(n)$

$\frac{2n}{3}\left(\log n\ - \log 3\right) \leq\ cn\log(n)$

$c \geq \frac{2}{3}\left(1 - \frac{\log 3}{\log n}\right) \Longrightarrow c \geq \dfrac{2}{3}$ per $n > 2$

Il metodo dell'esperto ci garantisce quindi che $T(n) = \Theta(n\log n)$.

2. La ricorrenza di partenza è la seguente:



tuttavia andremo a risolverla effettuando la seguente sostituzione che ci facilita i calcoli: $\log n = m \implies n = 5^m$ (consideriamo il logaritmo del termine $\log^2(n)$ con base 5, perché un cambio di base risulterebbe semplicemente nel moltiplicare il logaritmo per un termine costante che volutamente tralasciamo per non appesantire la notazione).

fatta questa considerazione la nostra espressione diventa:$T(5^m) = 3T(\frac{5^m}{5}) + m^2 \implies T(5^m) = 3T(5^{m-1}) + m^2$, da qui basta prendere in considerazione la ricorsione equivalente:

$U(m) = 3U(m-1) + m^2$ è facile vedere che sono equivalenti perché nella ricorrenza precedente, seppure l'input decresce in maniera esponenziale, il tempo a sottoproblema è sempre influenzato solo dal valore dell'esponente e non dalla grandezza reale dell'input.

Nella nuova ricorsione ottenuta il numero di livelli dell'albero è $m - i = 1 \implies i = m - 1$, ne segue:

$\sum_{i=0}^{m-2} 3^i m^2 + 3^{m-1} T(1) = m^2 \left(\frac{1-3^{m-1}}{1-3}\right) + 3^{m-1}\Theta(1) = -\frac{m^2}{2}(1 - 3^{m-1}) + 3^{m-1}\Theta(1)$

$\leq -m^2 + m^2 3^m + 3^m \Theta(1)$ sostituendo "m" otteniamo:

$-\log_5^2 n + \log_5^2 n \cdot 3^{\log_5 n} + \Theta(3^{\log_5 n}) = -\log_5^2 n + \log_5^2 n \cdot n^{\log_5 3} + \Theta(n^{\log_5 3})$

Quindi $T(n) = O(\log_5^2 n \cdot n^{\log_5 3})$, per quanto riguarda il limite inferiore guardando la ricorsione possiamo dire che: $T(n) = \Omega(\log^2 n)$ ma ciò sicuramente non è un limite stretto dato che la somma dei costi in ogni livello, per un n sufficientemente

grande, risulta crescente e sicuramente più grande di $\log^2 n$.

Il limite superiore e inferiore appena trovati con l'albero di ricorrenza non sono sufficientemente stretti, Applichiamo il metodo dell'esperto per trovare dei limiti più precisi:

$f(n) = \log^2 n$, $g(n) = n^{log_5 3} \backsimeq n^{0.68}$

$f(n) = O(n^{0.68-\epsilon}) \Longrightarrow T(n) = \Theta(n^{log_5 3})$ per il caso 1 del metodo dell'esperto.

infatti si può dimostrare usando de l'Hôpital che una funzione logaritmica è asintoticamente sempre più piccola di una funzione polinomiale, qualsiasi sia il suo esponente.

$\lim_{n\to\infty} \frac{\log^2 n}{n^a}$ applichiamo de l'Hôpital 2 volte per ottenere:

$\lim_{n\to\infty} \frac{2\frac{\log n}{n}}{an^{a-1}} = \lim_{n\to\infty} \frac{2\log n}{an^a}$ applicando la seconda volta de l'Hôpital

otteniamo:

$\lim_{n\to\infty} \frac{2}{na^2n^{a-1}} = \lim_{n\to\infty} \frac{2}{a^2n^a} = 0$

da qui dimostrato che n^a è un infinito di ordine superiore rispetto $log^2 n$ $\forall a > 0$.

Problema 1.

Traccia:

Si implementi un algoritmo di ordinamento che sfrutta l'inserimento e la visita in un albero binario di ricerca. Dato un vettore di n numeri interi in input, l'algoritmo procede prima ad inserire i numeri in un albero binario di ricerca (usando ripetutamente TREE-INSERT per inserire i numeri uno alla volta), e poi stampa i numeri in ordine con un attraversamento in ordine simmetrico dell'albero. Si analizzi la complessità nel caso peggiore e nel caso migliore di per questo algoritmo di ordinamento. Si alleghi al PDF un file editabile riportante l'implementazione in un linguaggio a scelta, corredato da almeno tre casi di test

Soluzione:

L'algoritmo prevede l'inserimento di ciascun valore in un albero binario di ricerca inizialmente vuoto e risulta essere analogo ad un algoritmo di Quicksort con partioning non randomizzato: il primo elemento inserito nell'albero, ovvero la radice, gioca il ruolo di "primo pivot" per tutti gli inserimenti successivi, il secondo nodo, diventando la radice del nuovo sottoalbero destro o sinistro, fungerà da pivot per tutti gli elementi futuri aggiunti in quel particolare sottoalbero (concetto di partizionamenti ricorsivi) e così via lo stesso ragionamento è applicabile a tutti i successivi nodi. Per questo motivo il caso peggiore, ovvero quando l'input è un array già ordinato, si verifica quando l'albero binario degenera in una lista e l'inserimento di ogni elemento necessita del confronto con tutti gli elementi precedentemente inseriti, arrivando ad effettuare il massimo numero possibile di controlli. Dopo aver inserito tutti gli elementi si percorre l'albero binario con un algoritmo di In-Order Traversal ($\Theta(n)$).

Complessità temporale:

La complessità temporale dipende dalle insert e dall'in-order traversal: $O(nh + n)$ con h altezza dell'albero binario.

Nel worst case si ha che h = n: complessità $\Theta(n^2 + n) = \Theta(n^2)$.
Nel best case si ha h = $\log n$ (albero bilanciato): complessità $\Theta(n \log n + n) = \Theta(n \log n)$.

Problema 2.

Traccia:

Si implementi un algoritmo che, a partire da un vettore di n numeri interi in input, costruisce un heap chiamando ripetutamente la procedura MAX-HEAP-INSERT (vedi slide su heapsort) per inserire gli elementi nell'heap. L'algoritmo di costruzione ha il seguente pseudocodice:

```
BUILD-MAX-HEAP_v2(A):
    A.heap_size = 1
    for (i=2) to A.length
        MAX-HEAP-INSERT(A, A[i])
```

Si alleghi al PDF un file editabile riportante l'implementazione in un linguaggio a scelta, corredato da almeno tre casi di test Si confronti la BUILD-MAX-HEAP vista a lezione con la BUILD-MAX-HEAP_ v2, in particolare: le due procedure creano sempre lo stesso heap se vengono eseguite con lo stesso array di input? Dimostrare che lo fanno o fornite un controesempio. Dimostrare che, nel caso peggiore, BUILD-MAX-HEAP v2 richiedo un tempo $\Theta(n \log n)$ per costruire un heap di n elementi.

Soluzione:

Il problema prevede la costruzione di un Max-Heap effettuando N-1 chiamate a max_heap_ insert (build_max_heap_v2) piuttosto che chiamare max_heapify a ritroso partendo dal primo nodo non foglia fino alla radice (build_max_heap classica). La funzione max_heap_insert prevede al più $\log n$ swap per far risalire l'elemento inserito nel max-heap. Se la sequenza in input è ordinata in maniera crescente ogni elemento di indice

i dovrà risalire $\log(i)$ livelli, in particolare l'ultimo elemento dovrà risalire $\lfloor \log(n) \rfloor$ livelli (worst case $\Theta(n)$). Se la sequenza in input è ordinata in maniera decrescente ogni elemento inserito sarà già nella posizione corretta (best case $O(1)$). Le due procedure build_max_heap e build_max_heap_v2 non generano lo stesso heap a partire da uno stesso array di input.

Ad esempio:

$input = [3, 4, 10, 12, 1]$

output con build_max_heap: $[12, 4, 10, 3, 1]$

output con build_max_heap_v2: $[12, 10, 3, 4, 1]$

Complessità temporale:

Vengono effettuate n chiamate a max_heap_insert

Nel worst case verranno effettuati $\Theta(\log n)$ swap: complessità $\Theta(n \log n)$.

Nel best case non verranno effettuati swap: complessità $\Theta(n)$.

Problema 3.

Traccia:

Sia A[1 . . n] un array di n elementi distinti "quasi ordinato", ovvero in cui ogni elemento dell'array si trova entro k slot dalla sua posizione corretta. Definiamo una "inversione": se $i < j$ e $A[i] > A[j]$, allora la coppia (i, j) è detta inversione di A. Si implementi un algoritmo che ordina il vettore A in tempo $(n \log k)$. Suggerimento: Si usi un Heap Si alleghi al PDF un file editabile riportante l'implementazione in un linguaggio a scelta, corredato da almeno tre casi di test.

Soluzione:

Il problema può essere risolto utilizzando un min-heap di lunghezza costante k+1. Essendo ogni elemento distante al più k dalla propria posizione corretta il minimo si troverà

sicuramente entro i primi k+1 elementi, quindi una volta costruito il min-heap sui primi k+1 elementi basta estrarre il minimo dalla testa con una pop ed inserire il nuovo elemento successivo a k+1 con una push iterando il processo fino ad aver fatto la push di tutti gli elementi all'interno dell'heap che verrà successivamente svuotato con una pop alla volta.

Complessità temporale:

La costruzione di un min-heap di lunghezza k ha complessità $\Theta(k)$ Vengono poi effettuate n operazioni di pop $O(\log k)$ e altrettante operazioni di push $O(\log k)$.
Complessità $= \Theta(k) + O(n)(O(\log k) + O(\log k)) = O(n \log k)$