

# Corso di Laurea Magistrale in Ingegneria Informatica

# CORSO DI ALGORITMI E STRUTTURE DATI

Prof. ROBERTO PIETRANTUONO

Homeworks set #1

# Istruzioni

Si prepari un file PDF riportante il vostro nome e cognome (massimo 2 studenti). Quando è richiesto di fornire un algoritmo, si alleghi un file editabile (ad esempio, .txt, .doc) riportante l'algoritmo in un linguaggio a scelta, corredato da almeno tre casi di test. Laddove opportuno, si fornisca una breve descrizione della soluzione: l'obiettivo non è solo eseguire l'esercizio e riportare il risultato, ma far comprendere lo svolgimento.

# Esercizio 1.1. Notazione asintotica

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.

- $\bullet \quad f(n) = O(f(n)^2)$
- $f(n) + O(f(n)) = \Theta(f(n))$
- $f(n) = \Omega(g(n))$  e f(n) = o(g(n)) Nota la notazione *little-o*

### Esercizio 1.2. Notazione asintotica e crescita delle funzioni

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

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

## Esercizio 1.3. Complessità.

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

## Esercizio 1.4. Ricorrenze

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

- $T(n)=2T(n/3)+n \lg n$
- $T(n) = 3T(n/5) + \lg^2 n$

### Problema 1.

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

#### Problema 2.

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.

### Problema 3.

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 lg 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