FONDAMENTI DI INFORMATICA II – modulo Algoritmi e Strutture dati

3 luglio 2009 - ANNO ACCADEMICO 2008/2009

NOME: **COGNOME:** **MATRICOLA:**
PROGETTO: **SI** **NO** **(barrare SI se si intende svolgere lo scritto ridotto)**

Esercizio 1
Sia dato il seguente vettore A di interi:

[46 69 52 28 49 45 21]

Scrivere tutte le chiamate a quicksort generate dalla chiamata quicksort(A, 0, 6). Per ogni chiamata indicare lo stato dell'array e il valore dei parametri attuali e del perno.

Array A	**inf**	**sup**	**perno**
[46 69 52 28 49 45 21]	0	6	28
[21 28 52 69 49 45 46]	0	1	21
[21 28 52 69 49 45 46]	2	6	49
[21 28 46 45 49 69 52]	2	3	46
[21 28 45 46 49 69 52]	5	6	69
[21 28 45 46 49 52 69]			

Situazione	***Esercizi***	***Tempo***
Senza progetto	**1,2,3,4,5**	80 min
Con progetto	**1,2,3**	40 min

1	***2***	***3***	***4***	***5***

Esercizio 2 Siano date le seguenti strutture dati

```
struct Node {
   char symbol;
   int frequency;
   Node *left, *right;
};
```

```
template <typename T>
class Heap {
   /* ... */
   void insert(T value);
   T extract();
   bool empty();
};
Heap<Node*> huff;
```

rappresentanti rispettivamente un nodo dell'albero del codice di Huffman, e un min-heap su cui sono definite le operazioni di inserimento ed estrazione con le complessità note.

a) Si scriva la funzione necessaria a costruire l'albero del codice, ipotizzando che l'istanza **`huff`** contenga **`n`** nodi relativi agli **`n`** simboli diversi incontrati in un testo, ognuno con la rispettiva frequenza. Tutti i puntatori **`left`** e **`right`** sono inizialmente a NULL. Si calcoli la complessità del codice in funzione di **`n`**.

b) Si calcoli la complessità del codice se il min-heap viene sostituito da una lista ordinata. Motivare la risposta.

c) Si calcoli la complessità del codice se il min-heap viene sostituito da un albero binario di ricerca. Motivare la risposta.

a)

```
for(int i=0; i<n-1;i++) {
  Node *t = new Node();
  t->left = huff.extract();
  t->right = huff.extract();
  t->frequency = t->left->frequency + t->right->frequency;
  huff.insert(t);
}
return huff.extract();
```

Complessità: O(n log n) (estrazione e inserimento in uno heap hanno complessità logaritmica**)**

b)

Complessità: $O(n^2)$ (inserimento in una lista ordinata ha complessità lineare, l'estrazione dalla testa complessità costante**)**

c)

Complessità: O(n log n) (estrazione e inserimento in uno albero binario di ricerca hanno complessità logaritmica se l'albero è quasi bilanciato**)**

Esercizio 3
Calcolare la complessità del blocco (indicando le relazioni di ricorrenza di tempo e risultato per ogni funzione) in funzione di n:

```
{
  int a = 0;
  for (int i=0; i <= f(g(n)); i++)
     a += f(n);
}
```

con le funzioni **f** e **g** definite come segue:

```
int f(int x) {
   if (x<=1)
     return 1;
   cout << f(x/3) + f(x/3);
   return f(x/3) + 1;
}
```

```
int g(int x) {
  if (x<1) return 1;
  int a=0;
  for (int i=0; i <= x; i++)
     a+=i;
  return a + g(x-1);
}
```

Indicare per esteso le relazioni di ricorrenza e, per ogni comando ripetitivo, il numero di iterazioni e la complessità dell'iterazione singola.

Calcolo f(n)

$T_f(0,1)= k$
$T_f(n)= h + 3\ T_f(n/3)$ $T_f(n)$ è $O(n)$

$R_f(0,1)= 1$
$R_f(n)= 1 + R_f(n/3)$ $R_f(n)$ è $O(\log n)$

Calcolo g(n)
for:
numero iterazioni: O(n) Complessità singola iterazione: O(1)
complessità del for: O(n)
alla fine del for a contiene la somma dei primi x numeri

$T_g(0)= k$
$T_g(n)= hn + T_g(n-1)$ $T_g(n)$ è $O(n^2)$

$R_g(0)= k$
$R_g(n)= hn^2 + R_g(n-1)$ $R_g(n)$ è $O(n^3)$

Calcolo for del blocco:
numero iterazioni: $f(g(n))= f(n^3)= \log n3 = \log n$

Complessità singola iterazione: $T_g(n) + T_f(n^3) = O(n^2) + O(n^3)= O(n^3)$
Complessità totale blocco = $O(n^3 \log n)$

Esercizio 4

Sia dato un albero binario di ricerca che contiene etichette intere. Si supponga che l'albero contenga almeno k nodi. Si scriva una funzione int kmax(Node* t, int k) che restituisce la k-esima etichetta più grande (Ad esempio, se k=1 restituisce la massima etichetta, se k=2 restituisce la seconda etichetta più grande, etc).

```
void km(Node *t, int& k) {
  if (t) {
    km(t->right, k);
    if (k==1) throw t->label;
    k--;
    km(t->left,k);
  }
}

int kmax(Node* t, int k) {
  int i = k;
  try {
    km(t,i);
  } catch (int result) {
    return result;
  }
}
```

```
int km2(Node *t, int& k) {
  if (t) {
    int res = km2(t->right, k);
    k--;
    if (k<0) return res;
    if (k==0) return t->label;
    return km2(t->left,k);
  }
}

int kmax2(Node* t, int k) {
  int i = k;
  return km2(t,i);
}
```

Esercizio 5

Sia dato un albero binario ad etichette intere t. Si scriva una funzione che restituisce il numero dei nodi dell'albero che hanno più discendenti a sinistra che discendenti a destra. Calcolare la complessità della soluzione proposta in funzione del numero dei nodi dell'albero.

```
int morelthanr(Node* t, int& nodes) {
  if (!t) {
    nodes = 0;
    return 0;
  }
  int nl, nr;
  int sum = morelthanr(t->left,nl)+
    morelthanr(t->right,nr);
  nodes = nl + nr + 1;
  return sum + (nl>nr);
}
```

Complessità O(n)