FONDAMENTI DI INFORMATICA II – Complessità, Algoritmi e Strutture dati

30 giugno 2008 - ANNO ACCADEMICO 2007/08

NOME: **COGNOME:** **MATRICOLA:**

SCRITTO RIDOTTO: **SI** **NO** **(barrare SI se si intende svolgere lo scritto ridotto)**

Esercizio 1

Sia dato il seguente vettore A di interi:

[15 -2 88 55 21 6 9]

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*
[15 -2 88 55 21 6 9]	0	6	55
[15 -2 9 6 21 55 88]	0	4	9
[6 -2 9 15 21 55 88]	0	1	6
[-2 6 9 15 21 55 88]	3	4	15
[-2 6 9 15 21 55 88]	5	6	55

Esercizio 2

Sia data la seguente struttura che rappresenta un grafo di **`num_nodi`** nodi ad etichette intere, memorizzato tramite una matrice di adiacenza **`adiacenza`**.

```
struct Grafo {
  int* nodi;              // etichette dei nodi
  int num_nodi;           // numero di nodi
  bool** adiacenza;       // matrice di adiacenza
};
```

Scrivere una funzione `int num_archi(Grafo& g);` che restituisce il numero di archi del grafo **`g`** e calcolarne la complessità in funzione del numero *n* di nodi e del numero *m* di archi.

```
int num_archi(Grafo& g) {
  int archi = 0;
  for (int i = 0; i<g.num_nodi;i++)
    for (int j = 0; j<g.num_nodi;j++)
      if (g.adiacenza[i][j])
        archi++;
  return archi;
}
```

Complessità: $T_{num_archi}(n,m) = O(n^2)$

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

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

Esercizio 3

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

```
for (int i=0; i <= f(g(n)); i++)
    a += 1;
```

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

```
int f(int x) {
  if (x<=1) return 1;
  cout << f(x-1)+f(x-2);
 return x;
}
```

```
int g(int x) {
 if (x<=1) return 1;
 int a=0;
 for (int i=0; i<=f(200);i++)
     a+=1;
 return a + g(x/2);
}
```

Stima del tempo di f

$T_f(0)= d$

$T_f(n)= c + T_f(n-1) + T_f(n-2)$

T_f è $O(2^n)$

risultato

$R_f(n)$è $O(n)$

Stima del tempo di g:

numero iterazioni del for: O(1)

complessità di un'iterazione: O(1)

tempo del for: O(1)

tempo di g

$T_g(1)= d$

$T_g(m)= c + T_g(n/2)$

T_g è $O(\log n)$

Stima del risultato di g

$R_g(1)= d$

$R_g(m)= c + R_g(n/2)$

$R_g(n)$ è $O(\log n)$

Complessità del for:

numero iterazioni del for:

$R_f((R_g(n))= R_f(\log n)= O(\log n)$

complessità di un'iterazione:

$T_g(n)+T_f(R_g(n))= O(\log n)+T_f(\log n)= O(\log n)+O(2^{\log n}) =O(\log n)+O(n) = O(n)$

Complessità del for = $O(n \log n)$

Esercizio 4

Sia dato un albero binario di ricerca ad etichette intere e due interi **a** e **b**. Si scriva una funzione che ritorna la mediana delle etichette presenti nell'albero che sono maggiori di **a** e minori di **b**. Si supponga che nell'albero esista almeno un'etichetta compresa fra **a** e **b**. Calcolarne la complessità in funzione del numero di nodi *n* dell'albero.

```
void between(Node* t,
             list<int>& l,
             int a, int b) {
  if (t) {
    if (t->label <= a)
      between(t->right, l, a ,b);
    else if (t->label<b) {
      between(t->left,l,a,b);
      l.push_back(t->label);
      between(t->right,l,a,b);
    } else {
      between(t->left,l,a,b);
    }
  }
}

int mediana(Node* t,
            int a, int b) {
  list<int>::iterator i;
  list<int> l;
  between(t,l,a,b);
  int n = l.size();
  i = l.begin();
  for (int j=0;j<n/2;j++)
    ++i;
  if (n%2)
    return *(i);
  else {
    int risultato = *i;
    risultato += *(--i);
    return risultato/2;
  }
}
```

```
void between2(Node* t, Elem*& l,
int a, int b, int& n) {
  if (t) {
    between2(t->right,l,a,b,n);
    if ((t->label >a) &&
        (tree->label<b)) {
      Elem* e = new Elem;
      e->info = tree->label;
      e->next = l;
      ++n;
      l = e;
    }
    between2(tree->left,l,a,b,n);
  }
}

int mediana2(Node* t,
             int a, int b) {
  Elem *l = NULL, *p;
  int n=0, result;
  between2(t,l,a,b,n);
  p = l;
  if (n%2) { // dispari
    for(int i=0;i<n/2;i++)
      p = p->next;
    result = p->info;
  } else { // pari
    for(int i=0;i<n/2-1;i++)
      p = p->next;
    result = (p->info +
              p->next->info)/2;
  }
  p = l;
  while (l) {
    l = l->next;
    delete p;
    p = l;
  }
  return result;
}
```

complessità O(n) (l'inserzione nella lista ha complessità costante e nel caso peggiore viene visitato tutto l'albero, scorrere metà della lista non altera la complessità)
Inserire in testa o in coda non fa differenza visto che la mediana non cambia.

Esercizio 5

Rispondere alle seguenti domande, motivando le risposte.

(a) Tutti gli array ordinati in ordine decrescente rappresentano un max-heap?

(b) Tutti i max-heap sono array ordinati in ordine decrescente?

(c) In uno heap le foglie sono tutte minori o uguali di tutti i nodi interni? Se sì, dimostrarlo per induzione sfruttando le proprietà dello heap, se no mostrare un controesempio.

(a) Sì, un array ordinato in ordine decrescente rappresenta sempre uno heap (linearizzato). Infatti la proprietà che ogni nodo è maggiore dei suoi discendenti deriva dall'ordinamento decrescente dell'array in quanto i discendenti dei nodi hanno indici maggiori nell'array.

(b) No, non è vero il contrario in quanto non esiste alcun vincolo di ordinamento fra nodi dello stesso livello. Un possibile controesempio è dato dal seguente array
[100 70 80]
che rappresenta uno heap ma non è ordinato in ordine decrescente.

(c) No, in quanto nello heap la proprietà di ordinamento mette in relazione solamente ogni nodo con i suoi discendenti.
Un possibile controesempio è dato dall'array:
[100 80 70 75 60 30]
che rappresenta uno heap. I suoi nodi interni sono 100, 80, e 70. Esiste quindi una foglia (75) che è maggiore di un nodo interno (70).