

# Introduction to algorithms and data structures

The IT University of Copenhagen

January 17. 2003

This examination assignment consists of 3 exercises with a total of 13 subexercises. Subexercises are given equal weight in the grading. You have 4 hours to answer all 13 subexercises. Remember to write the page number, your name and your CPR.-number on each page of your written answer. The complete assignment consists of 5 numbered pages.

CLRS refers to “Introduction to Algorithms” by Cormen, Leiserson, Rivest and Stein, Second Edition, 2001.

For exercises that asks for efficient algorithms, you should also give the asymptotic time complexity of your algorithm, and the asymptotic complexity of the specified solution will be taken into account when grading. Exercises asking for time complexity must be answered using  $O$ -notation with least possible asymptotic growth.

## Exercise 1

a) Assume  $f(n) \leq n^2$ , for  $n > 10^9$ , and  $f(n) = n^4$ , for  $1 \leq n \leq 10^9$ . Is it true that  $f(n) = O(n^3 \log n)$ . Justify your answer.

**Answer a)** Yes. Choosing  $n_0 = 10^9$  we have to show that  $n^2 \leq c \cdot n^3 \log n$  for some constant  $c$ , which is trivially true.

Consider the following procedures:

F1( $n$ )

```
1  $i \leftarrow 1$ 
2 while  $i < n$ 
3     do  $i \leftarrow 2i$ 
```

F2( $n$ )

```
1  $i \leftarrow 1$ 
2 while  $i < n$ 
3     do for  $j \leftarrow 1$  to  $i$ 
4         do  $k \leftarrow 1$ 
5          $i \leftarrow 2i$ 
```

F3( $n$ )

```
1 for  $i \leftarrow 1$  to  $n$ 
2     do  $j \leftarrow n$ 
3         while  $j > i$ 
4             do  $j \leftarrow j - 1$ 
```

b) Assume that the procedures above are only called with an integer  $n > 0$ . What is the time complexity expressed in  $n$  for each of the procedures above?

**Answer b)** The running times are:

F1:  $O(\log n)$

F2:  $n + \frac{n}{2} + \frac{n}{4} \cdots + 1 = O(n)$

F3:  $1 + 2 + 3 + \cdots + n = O(n^2)$

Let  $A$  be an array representing a **max-heap** with  $n > 0$  elements as defined in CLRS chapter 6. Let LEFT, RIGHT, etc. also be defined as in chapter 6. Consider the following procedures:

INC1( $A, i$ )

1  $n \leftarrow \text{heap-size}[A]$

2  $A[i] \leftarrow A[i] + 1$

3 **if** LEFT( $i$ )  $\leq n$

4     **then** INC1( $A, \text{LEFT}(i)$ )

5 **if** RIGHT( $i$ )  $\leq n$

6     **then** INC1( $A, \text{RIGHT}(i)$ )

INC-MORE( $A, i$ )

1  $n \leftarrow \text{heap-size}[A]$

2 INC1( $A, i$ )

3 **if** LEFT( $i$ )  $\leq n$

4     **then** INC-MORE( $A, \text{LEFT}(i)$ )

5 **if** RIGHT( $i$ )  $\leq n$

6     **then** INC-MORE( $A, \text{RIGHT}(i)$ )

Recall that for a **max-heap**  $A$  we have that  $A[\text{PARENT}(i)] \geq A[i]$ , for all  $i > 1$ .

c) Assume  $A[\text{PARENT}(i)] > A[i]$ , for all  $i > 1$ , before a call to INC-MORE( $A, 1$ ). Notice that “ $>$ ” is used in the assumption. Is  $A$  a **max-heap** after the call to INC-MORE( $A, 1$ )? Justify your answer.

**Answer c)** Yes it is. INC1 adds 1 to node  $i$  and all subnodes recursively. The call INC-MORE( $A, 1$ ) first adds 1 to all nodes in the heap and recursively adds 1 to children. Therefore any child gets a 1 added once more than its parent. But since  $>$  held before and a child can only be 1 larger than its parent after the call the  $\geq$  must hold afterwards and so  $A$  is still a max-heap.

**d)** What is the time complexity of a call to INC-MORE( $A, 1$ )? Justify your answer.

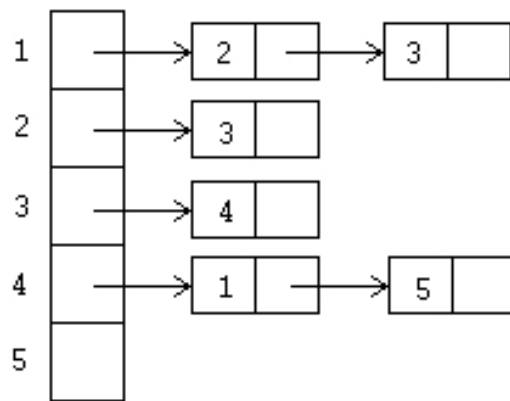
**Answer d)**  $T(n) = n + 2 \cdot T(\frac{n}{2}) = O(n \log n)$

## Exercise 2

Consider the following directed graph :

$$H = (\{1, 2, 3, 4, 5\}, \{(1, 2), (1, 3), (2, 3), (3, 4), (4, 1), (4, 5)\}).$$

a) Give the adjacency-list representation for  $H$  in the same way as Figure 22.2 page 528 in CLRS.



**Answer a)**

Let  $G = (V, E)$  be a directed graph. Let  $G_1 = (V, E - \{e\})$ ,  $e \in E$ . If  $G_1$  is acyclic, then  $e$  is denoted a *bad* edge.

b) List the bad edges from  $H$ .

**Answer b)**  $(3,4), (4,1)$

In the following we assume all graphs are represented as adjacency-lists.

Let  $T = (V, E_T)$  be a minimum spanning tree for the undirected weighted connected graph  $G = (V, E)$ . Let  $E' \subseteq \{(v, w) | v, w \in V\}$  be a set of weighted edges. Let  $G_1 = (V, E_T \cup E')$  and  $G_2 = (V, E \cup E')$ .

c) Prove that the weight of a minimum spanning tree for  $G_1$  equals the weight of a minimum spanning tree for  $G_2$ .

**Answer c)** Let  $E = E_T \cup E_D$ . For any edge in  $E_D$ , there is a cycle with this edge and a path in  $E_T$ , where the edge from  $E_D \geq$  any other edge in the cycle. Otherwise  $E_T$  would not be a MST for  $G$ .

For the graph  $G_1$  consisting of  $E_T \cup E'$  this observation still holds. That is  $E'$  must be a subset of  $E_D$ .  $G_2$  has edges  $E_T \cup E_D \cup E'$ , but no edge from  $E_D$  is in the MST, either is any edge from  $E'$ . Left is again  $E_T$ .

Let  $G = (V, E)$  be a directed weighted graph, with positive edge weight. Let  $V_1 \subseteq V$ .

**d)** Describe an efficient algorithm which given  $G$ ,  $V_1$ , and  $v \in V$ , finds a node in  $V_1$  which has a minimum distance to  $v$  among the nodes in  $V_1$ .

**Answer d)** Swap orientation of every edge. This gives a graph  $G_1$ . Calculate a shortest path tree from  $f$  in  $G_1$ . This gives the shortest path tree from all other nodes to  $f$  from the original graph. Now select the node  $f$ , which has the shortest distance in the shortest path tree and also is in  $V_1$ .

With a graph consisting of  $m$  edges and  $n$  nodes this gives a running time of  $(m+n) \log n$  with ordinary heaps and Dijkstra. distance ti

### Exercise 3

Consider the following directed graph :

$J = (\{1, 2, 3, 4, 5\}, \{(1, 2), (3, 1), (3, 4), (4, 2)\})$ .

a) Give a topological sorting of the graph  $J$ .

**Answer a)** 3,1,4,2 or 3,4,1,2, where 5 can be placed anywhere.

b) Let  $T$  be a binary rooted tree, where the nodes have associated keys. Give an efficient algorithm which checks if the keys in the tree satisfies the **binary-search-tree property**, see CLRS chapter 12, page 254

**Answer b)** Create a method `TEST( $x, least, largest$ )` which controls whether all values are between  $least$  and  $largest$  in the tree with root  $x$ . Call the method with  $(root, -\infty, \infty)$ .

```
TEST( $x, least, largest$ )
  if  $key(x) < least$  or  $key(x) > largest$ 
    then return false
  else if leaf( $x$ )
    then return true
  else return test(left( $x$ ),  $least, key(x)$ ) and
             test(right( $x$ ),  $key(x), largest$ )
```

The running time is  $O(n)$ .

Let  $A$  be an array of  $n > 0$  integers, each integer belonging to  $\{1, 2, \dots, n\}$ . Consider the following procedure:

`JUMP( $A, i$ )`

1  $n \leftarrow length[A]$

2 **for**  $j \leftarrow 1$  **to**  $A[i]$

3     **do**  $k \leftarrow 1$

4 **if**  $A[i] + i \leq n$

5     **then** `JUMP( $A, A[i] + i$ )`

c) What is the time complexity for a call to `JUMP( $A, i$ )`? Justify your answer.

**Answer c)** One iteration of the for-loop takes  $n$  time at most, since  $A[i] \leq n$ . The amount of time spent in the for-loop is linear in the length of the jump in the table. Therefore at most  $O(n)$  time is used.

**d)** Let  $G$  be a directed graph. Assume the edge weight on each edge is either  $1/2$  or  $2/3$ . Give an efficient algorithm, which given two nodes  $s, t \in V[G]$ , computes the length of a shortest path from  $s$  to  $t$  in  $G$ .

**Answer d)** Let there be two paths in graph  $G$  with lengths  $a$  and  $b$  from  $s$  to  $t$ , where  $a \leq b$ . If we multiply all edge-weights with 6, we obtain a new graph  $G_1$ , where these edges have length  $6a$  and  $6b$ . Since  $a \leq b$  then  $6a \leq 6b$  will also hold and therefore a path will be a shortest path in  $G_1$  if it is a shortest path in  $G$ .

In the graph  $G_1$  the edges have weight 3 and 2. Therefore the shortest path in this graph can be computed in linear time in different ways. For instance two dummy nodes are inserted on all edges of length 3 and one dummy node on edges with weight 2. As a consequence the new graph has unit weight on the edges and we find the shortest path using a breadth-first-search. After having found a shortest path we must remember to divide by 6.

Let  $G = (V, E)$  be a directed graph. Each node  $v$  has initially a unique positive integer value,  $U(v)$ , associated.

Consider the following program:

FUN( $G$ )

1 **while** there is an edge  $(v, w)$  where  $U(v) > U(w)$

2     **do**  $U(w) \leftarrow U(v)$

3 output for each node  $v$  the value  $U(v)$

**e)** Assume the graph  $G$  is acyclic. Describe an efficient algorithm which given  $G$  outputs the same as FUN( $G$ ).

**Answer e)** Claim: When the program terminates,  $U(v)$  for a node will be the largest possible initial  $U(w)$  value for a node  $w$  where in the graph there is a path from  $w$  to  $v$ .

Proof: Assume that  $U(v)$  ends with value  $x$ . Assume that there is a path  $P$  in  $G$  from  $w$  to  $v$  and  $U(w)$  initial is  $y > x$ . The claim is now shown correct by contradiction. Let  $P = w, v_0, v_1, v_2, \dots, v_k, v$ . When the program terminates it must be the case that:  $y \leq U(v_0) \leq U(v_1) \leq U(v_2) \leq \dots \leq U(v_k) \leq U(v) = x$  and therefore  $y \leq x$  which is a contradiction.

Because the graph is without cycles it can be ordered in linear time. After that we can consider the nodes in increasing topological order,  $v_1, v_2, \dots, v_n$ . When we look at node  $v_i$  to calculate its final  $U$ -value, we now the final  $U(v_j)$  for all edges  $(v_j, v_i)$ . Therefore we can calculate all  $U$ -values in linear time.