

Introduction to Algorithms and Datastructures

IT University of Copenhagen

9. januar 2002

This exam consists of 3 exercises containing in total 13 subexercises. Each of these 13 subexercises are given the same weight in the evaluation. You have 4 hours to complete the exam. Remember to number the pages and write your name and CPR number on every page. The exam consists of 4 numbered pages. CLR refers to “Introduction to Algorithms” by Cormen, Leiserson and Rivest, 18. print, 1997. CLRS refers to “Introduction to Algorithms” Second Edition by Cormen, Leiserson, Rivest and Stein, 2001.

For exercises, in which effective algorithms must be specified, the asymptotic time 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

This exercise is about determining O -functions and sorting from the chapters 2 and 8 in CLRS, or correspondingly the chapters 1 and 9 in CLR.

Look at the code A :

```
1 for  $i \leftarrow 1$  to  $n$ 
2   do  $A[i] \leftarrow i$ 
3      $B[i] \leftarrow 1$ 
4 MERGE-SORT( $A, 1, n$ )
5 COUNTING-SORT( $A, B, n$ )
6 for  $i \leftarrow 1$  to  $n$ 
7   do COUNTING-SORT( $A, B, n$ )
```

a) Give the time complexity for executing the lines 1-5 in the code A .

Answer a) $O(n \log n)$, time is dominated by the call to MERGE-SORT.

b) Give the time complexity for executing the lines 6-7 in the code A .

Answer b) $O(n^2)$, n iterations, each taking time $O(n)$.

Look at the procedure $\text{ADD1}(A, m)$

```
1 if  $m \geq 2$ 
2   then for  $i \leftarrow 1$  to  $m$ 
3     do  $A[i] \leftarrow A[i] + 1$ 
4     ADD1( $A, \lfloor m/2 \rfloor$ )
```

Look at the code B :

```
1 for  $i \leftarrow 1$  to  $n$ 
2   do  $A[i] \leftarrow 0$ 
```

3 ADD1 (A, n)

c) Give the time complexity for executing ADD1 i linie 3 for koden B .

Answer c) $1 + 2 + 4 + \dots + n/2 + n = O(n)$

d) Using O -notation, denote the largest value in the array A after executing ADD1 in line 3 of the code B .

Answer d) $O(\log n)$, entry 1 in A is incremented with 1 for each recursive call, which there are at most $\log n$ of, since m is halved in each call.

Exercise 2

This exercise is about graphs and minimum spanning trees with terminology as defined in chapter 22 and 23 of CLRS, or chapter 23 and 24 in CLR correspondingly. Look at the following unoriented graph

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

a) Draw the graph H .

Answer a) A drawing of a graph with five nodes, corresponding to H .

b) Give the adjacency representation for H as in figure 22.1 page 528 in CLRS or correspondingly figure 23.1 page 466 in CLR.

Answer b)

1		→	2	→	5				
2		→	1	→	3	→	5		
3		→	2	→	4	→	5		
4		→	3	→	5				
5		→	1	→	2	→	3	→	4

c) Denote edges crossing the cut $(\{1, 2\}, \{3, 4, 5\})$ in H .

Answer c) $(1, 5), (2, 3), (2, 5)$

Let $G = (V, E)$ be a connected, unoriented, weighted graph with n vertices and m edges. Let the set of vertices V be defined as the integers $\{1, 2, \dots, n\}$ as usually.

d) Describe an effective algorithm which, given G in adjacency-list representation, fills a table A with n entries, where each entry i contains a j such that the edge (i, j) is part of a minimum spanning tree for G . That is; for all vertices $i \in V$ it is the case that $(i, A[i])$ is an edge in a minimum spanning tree for G . Provide the time complexity of your solution.

Answer d) $A[i]$ is calculated by running through the adjacency-list for vertice i choosing neighbour j , where $w(i, j)$ is as small as possible. The time for this is $O(|Adj[i]|)$ for entry i , for all vertices in V this is, $O(|V| + \sum_i |Adj[i]|) = O(|V| + |E|)$.

e) Describe an effective algorithm which, given G in adjacency-list representation and an edge $e \in E$, decides whether e is part of some minimum spanning tree of G . Provide the time complexity of your solution.

Answer e) The edge e is in a minimum spanning tree, if and only if e connects two different connected components in the graph $G' = (V, E')$, where E' is defined to be edges from E , with edgeweights less than the edgeweight for e . The proof is as follows. assume e connects two different connected components in G' . We now must prove that e can be in a minimum spanning tree for G . Let $(S, V - S)$ be a cut, which respects the connected components in G' (no edge in E' crosses the cut), but at the same time separates the two components, connected by e . It is then the case that e is a *safe* edge in G , because it is the lightest in the mentioned cut, because only edges in $E - E'$ crosses the cut. That is; referring to theorem 23.1 in CLRS e is in a minimum spanning tree for G .

On the other hand if $e = (u, v)$ doesn't connect two connected components, then u and v are already connected with edges all having weight less than the weight for e . That is; e has greater weight than one of the other edges on a cycle in G . That is; according to exercise 23.1-5 in CLRS it implies, that there is a minimum spanning tree T without the edge e . The last implies that e can't be in a MST, since each spanning tree T' , which uses the edge $e = (u, v)$ has edgeweight-sum strongly greater than the tree $T'' = T(u) \cup \{e'\} \cup T(v)$, where $T(u)$ and $T(v)$ are the subtrees, which is obtained by erasing edge (u, v) , and $e' \neq e$ is an edge on the above cycle, which connects the trees $T(u)$ and $T(v)$ again. The conclusion is that T'' has edgeweight-sum strongly less than T' , which therefore can't be a minimum spanning tree. Therefore e is not in any MST. QED

To investigate whether e connects two connected components in G' , the adjacency-list representation for G' is first calculated, by in one iteration over G to remove edges with weight $\geq w(e)$. After that it is investigated for $e = (u, v)$, by for instance using a depth first search in G' , whether u is connected to v . In total this takes linear time in the size of the input. That is a running time of $O(|V| + |E|)$.

Exercise 3

This exercise is about the datastructure for disjoint sets as defined in chapter 21 in CLRS or chapter 22 in CLR, as well as binary search trees.

Look at the following code C :

```
1 for  $i \leftarrow 1$  to 5
2   do MAKE-SET( $i$ )
3 UNION(1, 3)
4 UNION(2, 3)
5 UNION(5, 3)
```

a) Is it the case that $\text{FIND-SET}(2) = \text{FIND-SET}(5)$ after executing line 1-5 in the code C ?

Answer a) Yes, because 2 and 5 are in the same set as 3.

Look at the following code D :

```
1 for  $i \leftarrow 1$  to  $n$ 
2   do MAKE-SET( $i$ )
3 for  $i \leftarrow 2$  to  $n$ 
4   do UNION(1,  $i$ )
```

b) Assume that the above union operations are implemented as *disjoint-set forests* (CLRS chapter 21.3 and coorespondingly CLR chapter 22.3) and with the *union by rank* heuristic. What is the *worst-case* time complexity for one UNION operation in the sequence of operations executed in the above code D ?

Answer b) At all times a leaf is linked to the tree. That is; the height of the tree with element 1 remains constant. Therefore the time is $O(1)$.

c) Let A be a table with n integers, orderet in ascending order. Describe an effective algorithm, which builds a balanced binary searchtree for the numbers in A . That is; the height must be $O(\log n)$. Provide the time complexity of your solution.

Answer c) The middle element, $A[\lfloor n/2 \rfloor]$ in A is made root, and the trees comprised of the elements to the left $A[1.. \lfloor n/2 \rfloor - 1]$ and right $A[\lfloor n/2 \rfloor + 1..n]$ for this element is calculated recursively and is made left and right subtree respectively. If A only consists of 0 elements NIL is returned. The time is expressed by $T(n) = 2T(n/2) + O(1)$, that is; $O(n)$. The height of the tree is $O(\log n)$, because the depth of the recursion only is $\log n$ (for each recursive call n is halved).

d) Let A be a table with n integers between 1 and $3n$. Describe an effective datastructure, which saves the integers in A in linear space in such a way that the datastructure afterwards can support the operation:

SEARCH(k): Returns **true** if the integer k is in A and **false** otherwise.

Provide the time complexity for building the datastructure and **SEARCH**.

Answer d) Initialize a table S with $3n$ entries to 0 in all entries. Run through A , and set $S[A[i]] = 1$ for all $1 \leq i \leq n$. **SEARCH(k)** now just return $S[k] = 1$. The time for initializing and filling S is clearly $O(n)$ and **SEARCH** $O(1)$.