MOCK EXAM SOLUTIONS

1. a) Write down the formal definition of $\Theta(g(n))$. [10%]

ANSWER:

$$\Theta(g(n)) = \{f(n) \mid \text{ there exist positive constants } c_1, c_2, \text{ and } n_0 \text{ such that for all } n \geq n_0 \\ 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)\}$$

No deduction of marks if they forget the inequality "$0 \leq \ldots$" or if they forget to say that constants are positive. Half marks if "for all $n \geq n_0$" is missing.

b) Simplify the following functions by expressing them in Θ-notation. Give a formal justification for each case, referring to the definition of Θ-notation.

(i) $n \cdot (3 + \log n)$

ANSWER:

This is $\Theta(n \log n)$ as for all $n \geq n_0 := 2$ we have $c_1 n \log n \leq n \cdot (3 + \log n) \leq c_2 n \log n$ for $c_1 := 1$ and $c_2 := 4$.
Marking: deduct [5%] for each mistake (constants or "for all $n \geq n_0$" missing).

(ii) $4n^2 + n - 100$

ANSWER:

This is $\Theta(n^2)$ as for all $n \geq n_0 := 100$ we have $c_1 n^2 \leq 4n^2 + n - 100 \leq c_2 n^2$ for $c_1 := 4$ (as then $n - 100 \geq 0$) and $c_2 := 5$ (as $4n^2 + n \leq 5n^2$).
Marking: deduct [5%] for each mistake (constants or "for all $n \geq n_0$" missing).

[30%]

c) For each of the following statements, decide whether the statement is true or false. Explain your answers.

(i) $o(n) = O(n)$ [15%]

ANSWER:

True. Formal or informal explanations are acceptable such as the following.

An informal answer: the set $o(n)$ describes all functions that grow slower than n. Each such function grows at most as fast as n. Hence "$f(n) = o(g(n))$" implies "$f(n) = O(g(n))$".

A formal answer: every function $f(n) = o(n)$ has $\lim_{n\to\infty} f(n)/n = 0$. Hence there must exist positive constants c, n_0 such that $f(n)/n \leq c$ for all $n \geq n_0$, which implies $f(n) \leq cn$ for all $n \geq n_0$.

(ii) $\Omega(n^2) = \Theta(n^2)$ [15%]

ANSWER:

False. The set $\Omega(n^2)$ describes all functions that grow at least as fast as n^2. This includes functions with a faster growth, such as n^3. This is not in $\Theta(n^2)$ as the latter only contains functions that grow proportionally to n^2.

d) The following algorithm counts the number of zeros within an array $A[1 \ldots n]$ of length $n \geq 1$.

COUNT-ZEROS(A)

```
x = 0
for i = 1 to A.length do
    if A[i] = 0 then
        x = x + 1
return x
```

Prove the correctness of COUNT-ZEROS by stating an appropriate loop invariant and showing the three properties: initialisation, maintenance, and termination.

[30%]

ANSWER:

The loop invariant is: *at the start of the for loop, x contains the number of zeros in the subarray $A[1], \ldots, A[i-1]$* [10%].

Initialisation: the loop invariant is true as for $i = 1$ the subarray is empty and $x = 0$. [5%].

Maintenance: if the loop invariant is true at iteration i, if $A[i] = 0$ then x is incremented and at the end of the loop, x will be the number of zeros in $A[1], \ldots, A[i]$ [5%]. Otherwise, x is unchanged as the number of zeros in $A[1], \ldots, A[i-1]$ equals the number of zeros in $A[1], \ldots, A[i]$. Incrementing i re-establishes the loop invariant [5%].

Termination: the loop ends when $i = n+1$. Then the loop invariant states that x is the number of zeros in $A[1], \ldots, A[n]$ [5%].

2. a) Copy the following table to your answer booklet and fill in asymptotic statements that best describe the running time of the given algorithms across inputs of n elements, using appropriate symbols Θ, O, and/or Ω.

Algorithm	running time
INSERTIONSORT	
SELECTIONSORT	
MERGESORT	
QUICKSORT	
BUBBLESORT	

[25%]

ANSWER:

Each correct line gives [5%]. Where the running time is $\Theta(f(n))$, stating $\Theta(f(n))$ is sufficient; no deduction of points if $O(f(n))$ and/or $\Omega(f(n))$ are stated in addition to $\Theta(f(n))$.

Algorithm	running time
INSERTIONSORT	$\Omega(n)$ and $O(n^2)$
SELECTIONSORT	$\Theta(n^2)$
MERGESORT	$\Theta(n \log n)$
QUICKSORT	$\Omega(n \log n)$ and $O(n^2)$
BUBBLESORT	$\Theta(n^2)$

b)

(i) Define the term *max-heap property*, referring to an array $A[1 \ldots n]$. [10%]

ANSWER:

For all nodes i except for the root, the value of the parent must be no smaller than that of the child: $A[\text{Parent}(i)] \geq A[i]$.

(ii) Does the following array represent a max-heap? Justify your answer.

42	33	15	20	24	18	4	5

[10%]

ANSWER:

No, the element 18 (index 6) is larger than its parent, 15 (index $\lfloor 6/2 \rfloor = 3$).

[0%] marks when no justification is given.

c) Recall that QUICKSORT uses the last element of the input as pivot element. Write down the contents of the following array $A[1\dots n]$ after the execution of PARTITION$(A,1,8)$.

4	3	8	2	7	5	1	6

[20%]

ANSWER:

The answer can be derived by simulating PARTITION. The input is the same as for Exercise Sheet 4, Question 4.1:

A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]
4	3	8	2	7	5	1	6
4	3	8	2	7	5	1	6
4	3	8	2	7	5	1	6
4	3	8	2	7	5	1	6
4	3	2	8	7	5	1	6
4	3	2	8	7	5	1	6
4	3	2	5	7	8	1	6
4	3	2	5	1	8	7	6
4	3	2	5	1	6	7	8

So the answer is:

4	3	2	5	1	6	7	8

d) Consider an array $A[1\dots n]$ of n integers in the range 0 to k. Give two algorithms PREPROCESS(A,n,k) and COUNT-LESS-OR-EQUAL-ELEMENTS(a) in pseudocode (or Java syntax) such that PREPROCESS preprocesses the input A in time $O(n+k)$. After preprocessing, COUNT-LESS-OR-EQUAL-ELEMENTS(a) must be able to return the number of elements in A which are less or equal to a ($\leq a$) in time $O(1)$, for arbitrary inputs $0 \leq a \leq k$. Explain why your algorithms meet the stated running time bounds.

[35%]

ANSWER:

This task is implicitly solved by COUNTINGSORT, and the idea can be borrowed from there. PREPROCESS creates an array $C[0\dots k]$ initialised with zeros. Then we scan the input array A and for the i-th element we increment $C[A[i]]$. At the end, $C[j]$ contains the number of elements in A that have value j. Then we compute a running sum to add up the elements less or equal to j. After that, COUNT-LESS-OR-EQUAL-ELEMENTS(a) simply returns $C[a]$.

```
Preprocess(A, n, k)
Let C[0...k] be a new array
for i = 0 to k do
    C[i] = 0
for j = 1 to A.length do
    C[A[j]] = C[A[j]] + 1
for i = 1 to k do
    C[i] = C[i] + C[i - 1]
```

```
Count-Less-or-Equal-Elements(a)
return C[a]
```

The running time of Preprocess is $\Theta(n+k)$: the second for loop takes time $\Theta(n)$ and the other two each take time $\Theta(k)$. The running time of Count-Less-or-Equal-Elements is $O(1)$.

Marking: [25%] (20% + 5%) for correct algorithms in pseudocode/Java as specified, and [10%] (5% + 5%) for analysing runtimes. No deduction if the initialisation of C is not mentioned. Deduct [20%] if at least one of the algorithms does not run in the required time. No marks for incorrect algorithms.

Answers without code (just explaining ideas or drawing a link to CountingSort) may receive up to [10%] for algorithms and [10%] for analysis.

3. a) Prove by induction that every nonempty binary tree satisfies $|V| = |E| + 1$.

ANSWER:

Proof by induction on $n = |V|$ for a tree T.

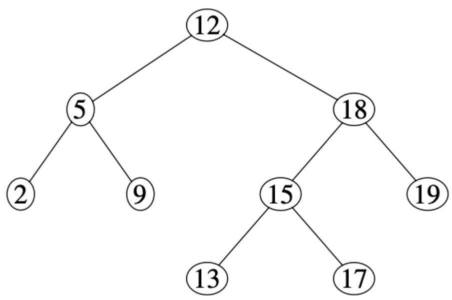
- Base case: For $n = 1$, T has only a root node, hence $|E| = 0$.
- Induction step: Suppose $|V| = |E| + 1$ for $|V| \leq n$ (Induction hypothesis). We must prove the claim for $|V| = n + 1$.

  There are three subcases.

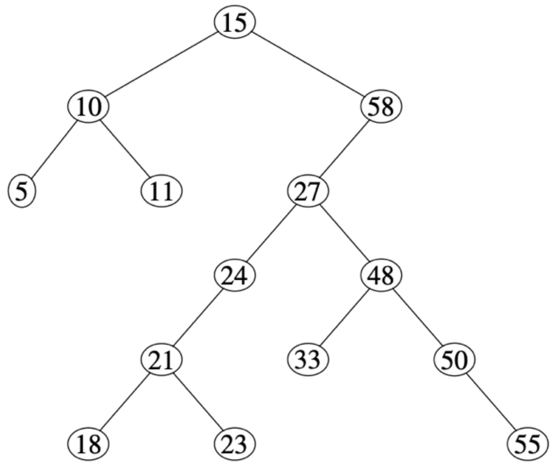
  1. The root of T has only a left child. Then this child has n nodes and therefore $n - 1$ edges by the induction hypothesis. Since T adds one single edge from the root to its left child, it has n edges (and $n + 1$ nodes by assumption).
  2. The case where the root of T has only a right child is symmetric.
  3. The root of T has two children. Suppose the left subtree has n_L nodes and the right subtree has n_R nodes. Then $n = n_L + n_R$ and it follows that $n_L < n + 1$ and $n_R < n + 1$. We can therefore apply the induction hypothesis to both subtrees, which yields $e_L = n_L - 1$ and $e_R = n_R - 1$, writing e_L for the number of edges in the left subtree of T and e_R for the number of edges in its right subtree. Hence, by substitution, $n = e_L + e_R + 2 = |E|$, and therefore $|E| = |V| - 1$.

b) Insert the numbers 12, 5, 9, 18, 15, 2, 17, 19 and 13 in that order into a binary search tree, which is initially empty.

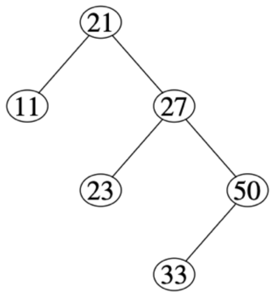
ANSWER:



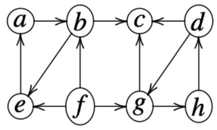
c) Delete the nodes labelled with 15, 58, 55, 48, 18, 10, 5 and 24 in that order from the following binary search tree. Show the resulting binary search tree.



ANSWER:



4. a) Perform a depth-first search on the directed graph below, visiting nodes in alphabetical order. Write down the timestamps of each node.



ANSWER:

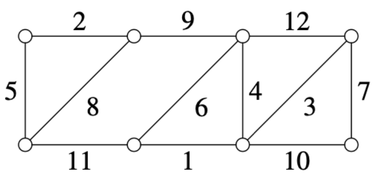
	d	f
a	1	8
b	2	7
c	3	4
d	9	14
e	5	6
f	15	16
g	10	13
h	11	12

b) Write down the strongly connected component graph of the graph from (a).

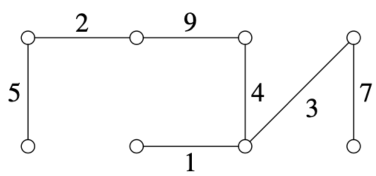
ANSWER:

Vertices are $\{a,b,e\}$, $\{c\}$, $\{f\}$ and $\{d,g,h\}$. Edges are $(\{a,b,e\},\{c\})$, $(\{f\},\{a,b,e\})$, $(\{f\},\{d,g,h\})$ and $(\{d,g,h\},\{c\})$.

c) With Kruskal's algorithm, compute the minimal spanning tree of the following weighted graph.



ANSWER:



d) Prove that every directed graph, which can be topologically sorted, is acyclic.

ANSWER:

By contradiction, suppose the graph G can be sorted topologically to the linear order $v_1 < v_2 < \cdots < v_n$ and contains a cycle. Let v_i be the lowest index vertex on that cycle and v_j its predecessor on the cycle. On the one hand, (v_j, v_i) is an edge of G. On the other hand, by minimality, $i < j$. Hence the above sequence cannot be a topological sorting.

END OF QUESTION PAPER