Delft University of Technology
EEMCS Faculty

CSE1305 Algorithms & Data Structures

Resit Written Exam

16 April 2019, 18:30–20:30

Examiners:

Examiner responsible:	Joana Gonçalves and Robbert Krebbers
Examination reviewer:	Stefan Hugtenburg

Parts of the examination and determination of the grade:

Exam part	Number of questions	Question specifics	Grade
Multiple-choice	22 questions (equal weights)	One correct answer per question	50%
Open questions	3 questions (different weights)	Multiple parts	50%

- The grade for the multiple-choice questions is computed as $10 \cdot \frac{\text{score}}{22}$.
- The grade for the open questions is computed as $10 \cdot \frac{\text{score}}{30}$.

Use of information sources and aids:

- A hand-written double-sided A4 cheat sheet can be used during the exam.
- No other materials may be used, including but not limited to books, lecture slides in any form, or devices such as laptops and phones.
- Scrap paper sheets are provided at the beginning of the exam. Additional scrap paper can be requested.

General instructions:

- Solve the exam on your own. Any form of collaboration is prohibited.
- You may not leave the examination room during the first 30 minutes.
- If you are eligible for extra time, put the "Verklaring Tentamentijd Verlenging" on your table.

Instructions for writing down your answers:

- You should answer the questions on the provided answer sheets.
- Write your name and student number on every sheet of paper.
- **For multiple-choice questions**, the order of the choices on the answer form **might not be A-B-C-D!**
- Tip: mark multiple-choice answers on this exam paper first, copy them to the answer form after revising.
- **For open questions**, provide all requested information and always give an explanation. Avoid irrelevant data, it could lead to deductions.
- **For proofs**, make sure your proof is properly structured and sufficiently explained. Statements or steps without justification could lead to point deductions.

This page is intentionally left blank.

Multiple-choice questions (50%, 22 points)

1. (1 point) Consider the Java implementation of method `operation` below.

```java
public int operation(int n) {
  int i, j, k = 0;
  for (i = n / 2; i <= n; i++)
    for (j = 2; j <= n; j = j * 2)
      k = k + n / 2;
  return k;
}
```

   Let n be a positive integer. What is the tightest worst-case time complexity of method `operation`?

   A. $\mathcal{O}(n)$
   B. $\mathcal{O}(n \log n)$
   C. $\mathcal{O}(n^2)$
   D. $\mathcal{O}(n^2 \log n)$

2. (1 point) Consider that the space complexity of an algorithm X is $O(1)$. Which of the following statements about algorithm X is **false**?

   A. The time complexity of algorithm X is guaranteed to be $\mathcal{O}(1)$.
   B. The time complexity of algorithm X is guaranteed to be $\Omega(1)$.
   C. The space used by algorithm X does not asymptotically depend on the input size.
   D. Algorithm X uses a constant amount of space in addition to the input.

3. (1 point) Consider the insertion of n elements (using `push` operations) in two different array stacks, `stack1` and `stack2`. For `stack1`, the array grows when full from capacity C to $\lceil \frac{5}{3}C \rceil$. For `stack2`, the array grows when full from capacity C to $C + \lceil \frac{5}{3} \rceil$. What are the tightest **amortized** time complexities of the set of n `push` operations in these two stacks?

   A. stack1: $\mathcal{O}(n^2)$ stack2: $\mathcal{O}(n^2)$
   B. stack1: $\mathcal{O}(n^2)$ stack2: $\mathcal{O}(n)$
   C. stack1: $\mathcal{O}(n)$ stack2: $\mathcal{O}(n^2)$
   D. stack1: $\mathcal{O}(n)$ stack2: $\mathcal{O}(n)$

4. (1 point) Which of the following data structures is **not** suitable to implement breadth-first traversal with the same tightest big-$\mathcal{O}$ time and space complexity than the others?

   A. Dynamic array
   B. Queue
   C. Singly-linked list
   D. Stack

5. (1 point) Consider a maximum-oriented array-based heap given by the sequence $(25, 14, 16, 13, 10, 8, 12)$. What is the sequence of elements in the array after two `removeMax` operations?

   A. (14,8,12,13,10)
   B. (14,13,12,8,10)
   C. (8,14,12,13,10)
   D. (14,10,12,13,8)

6. (1 point) Consider the partial implementation of class `SortedArray` below, which maintains a sequence of integer elements sorted in non-decreasing order.

```java
public class SortedArray {
  private Integer[] array;    // array containing the sorted elements
  private int size;           // number of elements in sorted array

  /* ... */
  public Integer operation(int element) {
    int a = 0, b = size-1, m = -1;

    while(a <= b) {
      m = a + (b-a)/2;
      if (element == array[m]) break;
      if (element < array[m]) b = m - 1;
      else a = m + 1;
    }
    if (a == size || (b <= a && array[a] != element)) return null;

    int i;
    if (b > a) i = m;
    else i = a;
    int e = array[i];

    for (int k = i+1; k < size; k++)
      array[k-1] = array[k];
    size--;
    return e;
  }
  /* ... */
}
```

What operation is performed by method `operation` in the Java code above?

A. Search element.

B. Search element index.

C. Insert element.

D. Remove element.

7. (1 point) Consider the method `operation` from question 6. What kind of search does it perform?

A. Linear search.

B. Brute-force search.

C. Binary search.

D. Multiway search.

8. (1 point) Consider the method `operation` from question 6. What is its tightest time complexity in Big-$\mathcal{O}$ notation, if n denotes the number of elements `size`?

A. $\mathcal{O}(n \log n)$

B. $\mathcal{O}(n)$

C. $\mathcal{O}(\log n)$

D. $\mathcal{O}(1)$

9. (1 point) Which of the following statements on sorting algorithms is **false**?
   A. An algorithm is called stable if it maintains the relative order of identical elements (or elements with an identical key).
   B. Four sorting algorithms ordered by **best-case** time complexity: insertion sort $\leq$ quicksort $\leq$ heap sort $\leq$ selection sort.
   C. The minimum possible time complexity of a comparison-based sorting algorithm is $\Omega(n \log n)$ for an input sequence with n elements.
   D. Quicksort is the fastest algorithm for sorting 1 million license plate numbers.

10. (1 point) What is the recurrence equation with $n > 1$ for the **worst-case** of the quicksort algorithm, where n denotes the input size and c_1 and c_2 are positive integer constants?
    A. $T(n) = T(n-2) + c_1 n + c_2$
    B. $T(n) = T(n-1) + c_1 n + c_2$
    C. $T(n) = T(n/2) + c_1 n + c_2$
    D. $T(n) = 2T(n/2) + c_1 n + c_2$

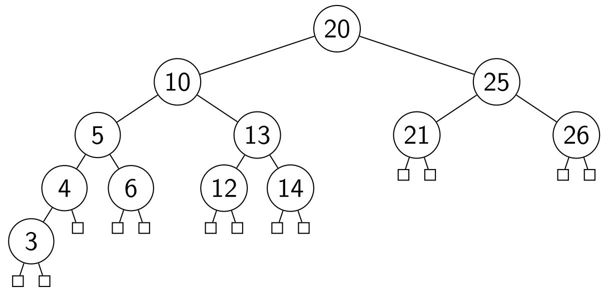
11. (1 point) Consider that the selection sort algorithm is applied to sort the sequence (1,7,6,2,8,4,5,3) in increasing order. What is the state of the sequence after a number of iterations resulting in 3 swaps?
    A. (1,2,3,7,8,4,5,6)
    B. (1,2,6,7,8,4,5,3)
    C. (1,2,3,4,8,7,5,6)
    D. (1,2,3,4,5,7,8,6)

12. (1 point) Consider the following fixed size hash table using linear probing (associated values are omitted):

0	1	2	3	4	5	6
7	5	0			12	6

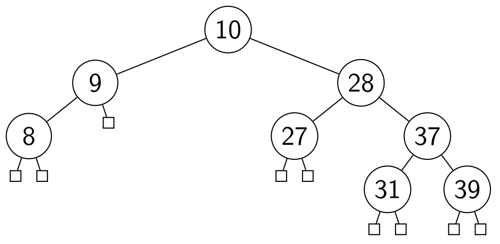
The hash function is $h(k) = k \mod 7$. Which of the following sequences **does not** denote a valid order by which the elements could have been inserted into the initially empty hash table?
    A. (12,7,6,5,0)
    B. (12,6,7,5,0)
    C. (6,12,5,7,0)
    D. (6,12,7,5,0)

13. (1 point) Consider the following tree:



Which of the following is **true**?
    A. The tree is an AVL tree and can be colored as a red-black tree.
    B. The tree is an AVL tree and cannot be colored as a red-black tree.
    C. The tree is not an AVL tree and can be colored as a red-black tree.
    D. The tree is not an AVL tree and cannot be colored as a red-black tree.

14. (1 point) Consider the following AVL tree:



How does the insertion of **30** affect the above AVL tree?

A. The insertion does not affect the AVL property.

B. The insertion violates the AVL property, which needs to be fixed with a single rotation.

C. The insertion violates the AVL property, which needs to be fixed with a double rotation.

D. The insertion violates the AVL property, which needs to be fixed with two double rotations.

15. (1 point) What is the maximum possible number of non-null nodes in an AVL tree of depth 4?

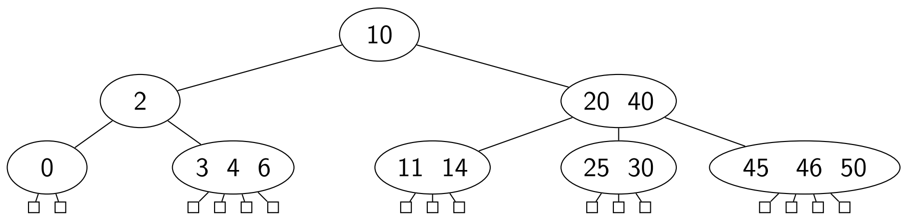
A. 8

B. 14

C. 15

D. 16

16. (1 point) Consider the following (2,4) tree.



What is the number of red-black trees that correspond to the given (2,4) tree?

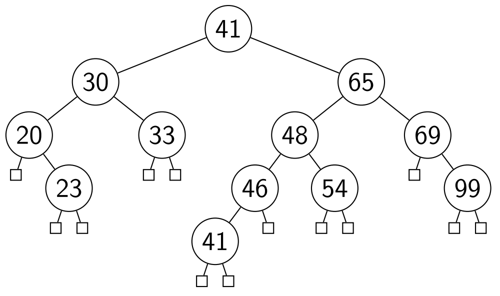
A. 2

B. 3

C. 4

D. 8

17. (1 point) Consider the following AVL tree:



When deleting **33** from the given AVL tree, how many tri-node restructurings are caused in the tree? If the node to be deleted has two children, the node will be replaced with the in-order predecessor (i.e. the maximal node in the left child).

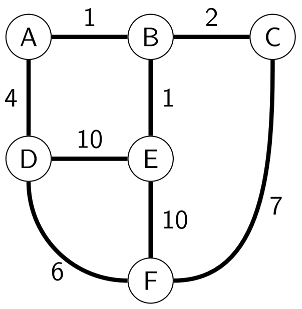
A. No tri-node restructuring.

B. One tri-node restructuring.

C. Two tri-node restructurings.

D. Three tri-node restructurings.

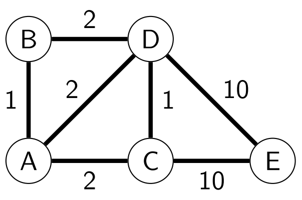
18. (1 point) Consider the following weighted graph:



When performing Dijkstra's algorithm starting from vertex A, how many **non-infinite distinct** labels will the vertex F have?

A. 1
B. 2
C. 3
D. 4

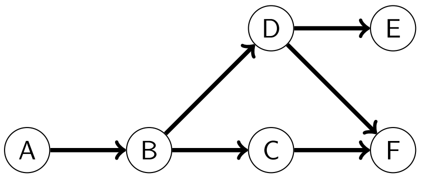
19. (1 point) Consider the following weighted graph:



In general, a graph can have multiple minimum spanning trees. How many different minimum spanning trees does the above graph have?

A. 3
B. 4
C. 5
D. 6

20. (1 point) Consider the following directed acyclic graph (DAG):



In general, DAGs can have multiple topological orders. How many different topological orders does the above graph have?

A. 2
B. 3
C. 4
D. 5

21. (1 point) Consider an undirected graph with n vertices and m edges that is a forest. Which of the following properties is **true**?

A. $m \leq n$
B. $m \leq n - 1$
C. $m \geq n$
D. $m \geq n - 1$

22. (1 point) Assume we need to represent dense graphs, and we are primarily interested in accessing edges, and removing and inserting vertices. What is the most efficient graph data structure for this purpose?

    A. Edge list.

    B. Adjacency list.

    C. Adjacency map.

    D. Adjacency matrix.

Open questions (50%, 30 points)

23. Consider the following Java implementation of `methodX`. Assume that the input array `a` is non-null.

```
 1  public static int methodX(Integer[] a) {
 2    int i = 0, n = a.length;
 3    while (i < n) {
 4      if(a[i] % 2 == 0) {
 5        for (int j = i; j < n - 1; j++)
 6          a[j] = a[j+1];
 7        a[n-1] = null;
 8        n--;
 9      } else {
10        i++;
11      }
12    }
13    return n;
14  }
```

(a) (5 points) Write the polynomial expressing the worst-case **time** complexity of method `methodX` as a function of n, where n is the number of elements in array `a`. Define all variables and constants. Explain each term of the polynomial by referring to the lines of code.

(b) (4 points) Simplify your polynomial expression as much as possible. State the tightest worst-case Big-$\mathcal{O}$ **time** complexity of `methodX` as a function of n. You **do not** have to give a proof, but you should clearly justify your answer.

(c) (3 points) Consider that the array `a` given as input to `methodX` only contains non-null elements. Explain what the algorithm `methodX` computes, i.e. what is the content of array `a` after calling `methodX(a)` and what is the return value? Describe a faster solution to perform the same operation and state its tightest worst-case **time** complexity.

24. Consider the following Java implementation of a recursive algorithm.

```
1  public static int methodY(int n) {
2    if (n == 0)
3      return 0;
4
5    return 1 + methodY(n - 1) + methodY(n - 1);
6  }
```

(a) (4 points) State the base and recurrence equation for the **time** complexity of method `methodY` as a function of n. Refer to the relevant parts of the code to justify your answer.

(b) (6 points) Derive the closed form of the given recurrence equation. You should either:

- derive the closed form solution by repeatedly unfolding the recurrence equation, or
- guess the closed form and prove correctness of your solution by induction.

(c) (2 points) State the tightest worst-case Big-$\mathcal{O}$ **time** complexity of `methodY` as a function of n. You **do not** have to give a proof, but you should clearly justify your answer.

(d) (2 points) State the tightest worst-case Big-$\mathcal{O}$ **space** complexity of `methodY` as a function of n. You **do not** have to give a proof, but you should clearly justify your answer.

25. Prove that $\left(\frac{n}{2}\right)^2 + \log(2)$ is $\mathcal{O}(n^4)$.

(a) (2 points) State in detail the mathematical conditions that should be proven.

(b) (2 points) Prove that these conditions hold. Explain all the steps in your proof.

End of the exam