Delft University of Technology
EEMCS Faculty

Resit exam A&D (TI1316/TI1316TW)

3 July 2018, 18:30–21:30

Examiners:

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

Parts of the examination and determination of the grade:

Exam part	Number of questions	Question specifics	Grade
Multiple-choice (paper)	24 questions (equal weights)	One correct answer per question	35%
Open questions (paper)	3 questions	Multiple parts	35%
Implementation (Weblab)	2 questions (Weblab weights)	Multiple parts	30%

Use of information sources and aids:

- Any version of the book Algorithms & Data Structures in Java by Tamassia et al. is allowed.
- The use of any other notes is not permitted. This means that lecture slides are not permitted either.
- Scrap paper sheets are provided at the beginning of the exam. Additional scrap paper can be requested.
- The use of any devices other than the assigned computer is not permitted.

Additional instructions:

- Solve the exam on your own. Any form of collaboration is fraud.
- You may not leave the examination room during the first 30 minutes.
- You will not be allowed to start the exam if you arrive after the first 30 minutes.
- If you are eligible for extra time, show your "Verklaring Tentamentijd Verlenging" to the surveillants.

Instructions on the paper (analysis) part:

- Write your name and student number on every sheet of paper.
- Write the total number of sheets of paper you hand in.
- **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.

Instructions on the Weblab (implementation) part:

- Make sure that all of your code compiles. If it does not compile, you get no points.
- **Computer login:** log in to the computer using the following details:

Username:	`EWI_Weblab`
Password:	`Tentamen01`

- **Weblab login:** log in to WebLab as usual using Single Sign-On.
- **Weblab exam:** access the exam assignment in the correct version of the course:

CS students:	`https://weblab.tudelft.nl/ti1315/2017-2018/` or
Math students:	`https://weblab.tudelft.nl/ti1315/TW+2017-2018/`

- **Weblab exam registration:** access your exam submission (pencil icon), register using your Weblab key.
- The Java API documentation can be found at `https://weblab.tudelft.nl/java8/api/`
- Busy Weblab server: if this happens, you might not be able to compile your solution immediately. Please save it, so you can compile and run it later. You can open other tabs and continue working while waiting.
- Do not close the browser after registering for the exam. If you do so accidentally, ask the surveillants.
- Questions: use the comment functionality of Weblab to ask for clarification about phrasing of the exam.

This page is intentionally left blank.

Multiple-choice questions (35%, 24 points)

1. (1 point) What is the asymptotic relationship between the functions $(c+2)^n$ and $(n+2)^k$? Assume constants $k \geq 1$ and $c > 1$.

   A. $(c+2)^n$ is $\mathcal{O}((n+2)^k)$.

   B. $(c+2)^n$ is $\Omega((n+2)^k)$.

   C. $(c+2)^n$ is $\Theta((n+2)^k)$.

   D. None of the above.

Answer: When n goes to infinity, $n \gg c$ and $n \gg k$ so we can disregard the values of c and k. The expression $(c+2)^n$ denotes an exponential function, while $(n+2)^k$ denotes a polynomial function. Asymptotically, this means that $(c+2)^n > (n+2)^k$ and therefore $(n+2)^k$ is a lower bound of $(c+2)^n$.

2. (1 point) Consider that the amortized time complexity of n operations is $O(nm)$. Which of the following statements **is false**?

   A. Some of the n operations can run in $\mathcal{O}(1)$ time.

   B. Each of the n operations takes $\mathcal{O}(m)$ time on average.

   C. Each of the n operations involves on average cm instructions, with constant $c \geq 1$.

   D. None of the n operations can run in $\mathcal{O}(m^2)$ time.

Answer:

A. Correct. An operation takes average $\mathcal{O}(m)$ time, so some operations could take $\mathcal{O}(1)$ time.

B. Correct. Since n operations take amortized $\mathcal{O}(nm)$ time, one takes $\mathcal{O}(m)$ time on average.

C. Correct. An operation takes $\Theta(m)$ time on average, therefore cm instructions with $c \geq 1$.

D. False. An operation takes average $\mathcal{O}(m)$ time, so some operations could be $\mathcal{O}(m^2)$.

3. (1 point) Consider the implementation of class DataS below.

```
public class DataS<E> {
  private int f;
  private E[] elements;

  public DataS<E>(int capacity) {
    elements = new E[capacity];
    f = -1;
  }

  public void insert(E x) {
    if (f >= elements.length)
      return;
    f = f+1;
    elements[f] = x;
  }

  public E remove() {
    if (f < 0)
      return null;
    f = f-1;
    return elements[f+1];
  }
}
```

What type of data structure does class `DataS` above implement?

A. Priority queue.

B. Queue.

C. Deque.

D. Stack.

Answer:

A. Not a priority queue, since it does not establish a particular order among its elements.

B. Not a queue, insertion and removal is not done according to the FIFO principle.

C. Not a deque, as it does not enable the retrieval of both the first and last elements.

D. Stack: `DataS` implements insertion and removal according to the LIFO principle.

4. (1 point) Consider class `DataS` from question 3. Assume that we insert all elements of a sequence of integers S into a data structure of type `DataS` and then perform an equal number of `remove` operations. In what order are the elements of S returned?

A. Same order as in S.

B. Reverse order.

C. Increasing order.

D. Decreasing order.

Answer: The elements are returned in reverse order, given that the data structure is a stack and therefore implements insertion and removal according to the LIFO principle. The last element in is the first element out, and thus the elements are returned in reverse order.

5. (1 point) Consider an algorithm to move the last but one element of a sequence S with n elements to the front of that sequence. Example: if $S = \{1, 2, 3, 4, 5\}$, the algorithm should change S into $\{4, 1, 2, 3, 5\}$. What is the complexity of the most time-efficient algorithm for this operation when S is implemented by an array, a singly-linked list, or a doubly-linked list?

A. **array:** $\mathcal{O}(1)$ **singly-linked list:** $\mathcal{O}(1)$ **doubly-linked list:** $\mathcal{O}(1)$

B. **array:** $\mathcal{O}(1)$ **singly-linked list:** $\mathcal{O}(n)$ **doubly-linked list:** $\mathcal{O}(1)$

C. array: $\mathcal{O}(n)$ **singly-linked list:** $\mathcal{O}(n)$ **doubly-linked list:** $\mathcal{O}(1)$

D. **array:** $\mathcal{O}(n)$ **singly-linked list:** $\mathcal{O}(n)$ **doubly-linked list:** $\mathcal{O}(n)$

Answer:

- Array: $\mathcal{O}(1)$ to access last but one element via index (arithmetic), $\mathcal{O}(n)$ to insert at the front.
- SLL: $\mathcal{O}(n)$ to find the last but one element, $\mathcal{O}(1)$ to insert at the front.
- DLL: $\mathcal{O}(1)$ to access the last element (via tail $\rightarrow$ previous), and $O(1)$ to insert at the front.

6. (1 point) What is the best time complexity to access the last element (rightmost leaf) in a heap containing n nodes, implemented by a linked tree structure without an explicit reference to the last node (tip: algorithm using binary encoding of tree paths)?

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

   B. $\mathcal{O}(\log_2 n)$

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

   D. $\mathcal{O}(n \log_2 n)$

Answer: Book chapter 13.4 Comparing sorting algorithms.

7. (1 point) Consider the most time-efficient algorithm for finding the second largest value in a sequence of integers S using only $\mathcal{O}(1)$ additional space. On which of the following data structures storing S **is it always impossible** to do this in $O(\log_2 n)$ time?

   A. Max-heap.

   B. Binary tree.

   C. AVL tree.

   D. Red-black tree.

Answer: Finding the maximum element in a binary tree takes $\mathcal{O}(n)$. Since the elements are in no particular order, it is necessary to traverse all nodes in the tree.

8. (1 point) Consider that a recursive algorithm has run-time recurrence equation $T(n) = 4T(\lfloor n/4 \rfloor) + k_1$, with constant $k_1 \geq 1$. What is the recurrence equation $S(n)$ denoting the **minimum** space complexity used by the same algorithm, assuming constant $c_1 \geq 1$?

   A. $S(n) = S(\lfloor n/4 \rfloor) + c_1$**.**

   B. $S(n) = 2S(\lfloor n/2 \rfloor) + c_1$.

   C. $S(n) = 2S(\lfloor n/4 \rfloor) + c_1$.

   D. $S(n) = 4S(\lfloor n/4 \rfloor) + c_1$.

Answer: The four recursive calls have identical input size, and therefore similar space requirements. The minimum space needed is equivalent to the space used by a single recursive call, as long as it only uses temporary space which can be reused by the subsequent calls.

9. (1 point) Which of the following statements on sorting algorithms **is false**?

   A. Quick sort sorts in the partition step, while merge sort sorts in the combining step.

   B. Sorting with a priority queue can be done in $\mathcal{O}(n \log_2 n)$ time.

   C. To sort variable-length keys, radix sort applies bucket sort from the least to the most significant key (right to left).

   D. When sorting variable-length keys, radix sort may not need to process all individual keys.

Answer:

A. Quick sort sorts in the partition step, by moving the elements smaller than the pivot to its left and the elements larger than the pivot to the right. Merge sort sorts in the combining step, when merging the sorted sequences that result from the recursive calls.

B. Sorting with a priority queue takes $\mathcal{O}(n \log_2 n)$ time using a heap.

C. To sort variable-length keys, radix sort applies bucket sort from the most to the least significant key (left-to-right).

D. When sorting variable-length keys, radix sort may not need to process all individual keys. Since it starts with the most significant key, it might be possible to determine the order of all the elements using only a subset of the keys. Example: {pig, monkey, elephant, parrot}. After first character {{elephant}, {monkey}, {pig, parrot}}. After second character {{elephant}, {monkey}, {parrot}, {pig}}. Each bucket has only one element, thus the sequence is sorted! Only two characters explored, despite the average number of 5.8 characters per element.

10. (1 point) Which sorting algorithm is best suited to sort a sequence that **does not** fit into main memory?

A. Heap sort.
B. Merge sort.
C. Quick sort.
D. Radix sort.

Answer: Book chapter 13.4 Comparing sorting algorithms.

11. (1 point) Which algorithm has the best **expected** time complexity to find the median element of an unordered sequence S with n elements, assuming that n is odd?

A. Binary search.
B. Quick select.
C. Quick sort followed by retrieval of the $\lceil n/2 \rceil^{th}$ element.
D. Build a min-heap and extract $\lceil n/2 \rceil$ elements. The median is the $\lceil n/2 \rceil^{th}$ element.

Answer:

A. Not directly applicable, the sequence is not sorted.

B. Quick select, worst-time $\mathcal{O}(n^2)$, expected time $\mathcal{O}(n)$.

C. Quick sort takes $\mathcal{O}(n \log_2 n)$ expected time, thus the solution is slower than quick select.

D. Building a min-heap takes only $\mathcal{O}(n)$ time, but extracting the elements takes $\mathcal{O}(\lceil n/2 \rceil \log_2 n)$.

12. (1 point) Consider you need to keep track of a large number of scores for two competitions, **FootLeague** and **CodingCup**. You will use a data structure per competition to store n pairs (t, s), where t is a team name and s the corresponding score. You will access and update scores, and users will often check the score of their favorite team as the competition progresses. For **FootLeague**, you are asked to keep the teams sorted in lexicographic/alphabetic order, so they can be retrieved in that order in $\mathcal{O}(n)$ time. For **CodingCup**, this is not necessary. Which data structures provide the best time complexity in each case?

A. **FootLeague**: Sorted array list. **CodingCup**: Doubly-linked list.
B. **FootLeague**: Array-based heap. **CodingCup**: Linked tree heap.
C. FootLeague: AVL tree. CodingCup: Hash table.
D. **FootLeague**: Red-black tree. **CodingCup**: AVL tree.

Answer:

A. Linked lists are not a good choice, since they do not enable efficient searching by key (time complexity is $\mathcal{O}(n)$).

B. Heaps are not a good choice, since they do not enable efficient searching of arbitrary elements (only finding maximum/minimum is $\mathcal{O}(1)$, finding arbitrary elements is $\mathcal{O}(n)$).

C. The AVL tree is a good choice for FootLeague, as searching/accessing takes $\mathcal{O}(\log_2 n)$ time while keeping the teams sorted. A hash table is a good choice for CodingCup, since searching takes $\mathcal{O}(1)$ time on average.

D. The AVL tree is not the best choice for **FootLeague**, since the entries don't need to be sorted, and a properly behaved hash table will be more efficient (see C).

13. (1 point) Which of the following statements about hash functions and hash tables **is false**?

A. A hash function takes a key of arbitrary length and generates a fixed length code.

B. A collision happens when `a.equals(b)` is false and `a.hashCode() == b.hashCode()` is true.

C. The load factor of a hash table is the ratio n/C between the number of entries n and the capacity of the hash table C.

D. Open addressing with linear probing uses more space than chaining.

Answer:

A. Correct, usually a 32-bit integer.

B. Correct, a collision happens when two different keys have the same hash code.

C. Correct definition of load factor.

D. False. Linear probing uses less space than chaining.

14. (1 point) Consider the following fixed size hash table using linear probing (associated values are omitted):

0	1	2	3	4
8		12	7	19

The hash function is $h(k) = k \mod 5$. Which of the following sequences denotes a valid order by which entries were inserted in an initially empty hash table?

A. $\{19, 12, 8, 7\}$

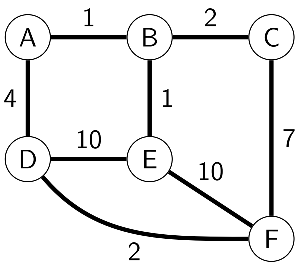
B. $\{12, 19, 8, 7\}$

C. $\{19, 12, 7, 8\}$

D. $\{12, 8, 7, 19\}$

Answer: According to the hash function: $12 \mod 5 = 2$, $7 \mod 5 = 2$, $8 \mod 5 = 3$, $19 \mod 5 = 4$. Entry 8 has to be last, since its original hash code is 3 and probing occurs until hash code is 0. This means that it skips the positions with indexes 3 and 4, which must be already occupied. So entries 12, 7 and 19 all go before 8. Entry 12 is placed before 7, since they both have hash code 2, and entry 12 is at index 2 (meaning that 7 is placed using probing). The only option that meets these conditions is C.

15. (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?

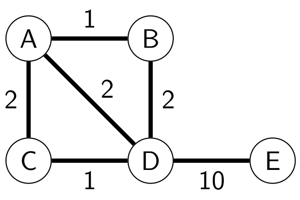
A. 1

B. 2

C. 3

D. 4

16. (1 point) Consider the following weighted graph:



Applying Kruskal's algorithm on the above graph will identify a *single* minimum spanning tree. However, in general, a graph can have multiple minimum spanning trees. How many different minimum spannings trees does the above graph have?

A. 1

B. 2

C. 3

D. 4

Answer: All edges with weight 1 and 10 should be part of the minimum spanning tree. Apart from that, there is the choice of adding one edge with weight 2.

17. (1 point) In general, a directed acyclic graph (DAG) can have multiple topological orders. If a DAG G has exactly one topological order, which of the following statements about properties of G is **true**?

A. G is a forest.

B. G is a tree.

C. G has no self-loops.

D. All of the above.

18. (1 point) Assume we have a large graph under the following conditions: fixed number of vertices, dynamic number of edges (frequent edge insertions and deletions), and frequent calls to method `getEdge(u,v)`. Which graph data structure is most time-efficient under this scenario?

A. Edge list.

B. Adjacency matrix.

C. Adjacency list.

D. Adjacency map.

Answer: The adjacency matrix has time complexity $\mathcal{O}(1)$ for `insertEdge(u,v)`, `removeEdge(u,v)` and `getEdge(u,v)`, even though it uses $\mathcal{O}(n^2)$ space (where n is the number of vertices).

19. (1 point) Consider a shortest path p from a vertex s to some other vertex t in a weighted undirected graph G, without negative weights. Under which conditions will p still be a shortest path from s to t?

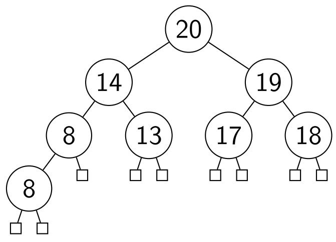
    A. If the weights of all edges in G are increased by one (i.e. weight w is replaced by weight $w+1$).

    B. If the weights of all edges in G are squared (i.e. weight w is replaced by weight w^2).

    C. If the weights of all edges in G are increased by two (i.e. weight w is replaced by weight $w+2$).

    D. If the weights of all edges in G are multiplied by two (i.e. weight w is replaced by $2w$).

20. (1 point) Consider the following tree:



Which of the following is **true**?

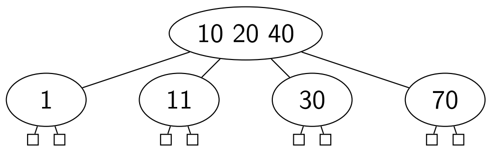
A. The tree is an AVL tree.

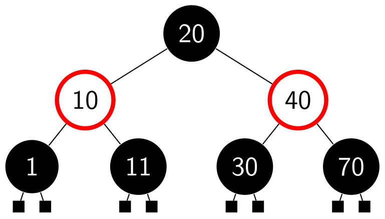
B. The tree can be colored as a red-black tree.

C. The tree is a minimum-oriented heap.

D. The tree is a maximum-oriented heap.

21. (1 point) Consider the following (2,4) tree on the left and binary tree on the right. In the binary tree on the right, filled circles denote black nodes and unfilled circles denote red nodes.





Which of the following statements is **true**?

A. The binary tree is a red-black tree, which represents the given (2,4) tree. At the same time, there is at least one other red-black tree that represent the given (2,4) tree.

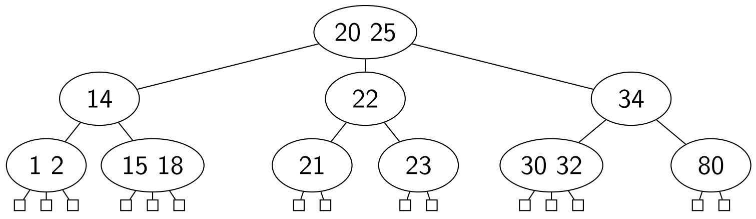
B. The binary tree is a red-black tree, which uniquely represents the given (2,4) tree.

C. The binary tree is a red-black tree, which does not represent the given (2,4) tree.

D. The binary tree is not a red-black tree.

Answer: Note that for each 3-node, there is a choice, but for 2 and 4-nodes there is no choice.

22. (1 point) Consider the following (2,4) tree:



When deleting **21** from the tree, how many underflows are caused in the tree?

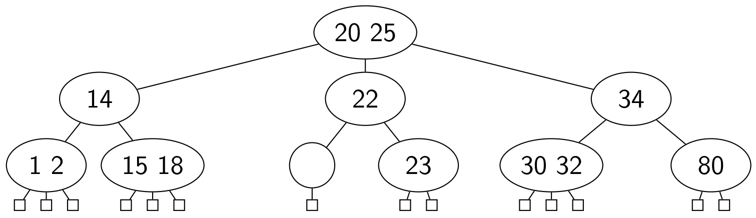
A. One underflow, which needs to be fixed by a fusion.

B. Two underflows, which need to be fixed by a transfer and fusion.

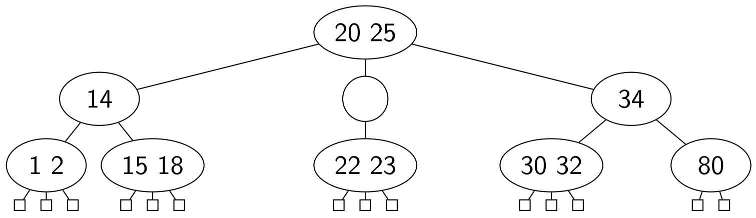
C. Two underflows, which need to be fixed by two fusions.

D. Two underflows, which need to be fixed by two transfers.

Answer: After removing 21, we end up with:

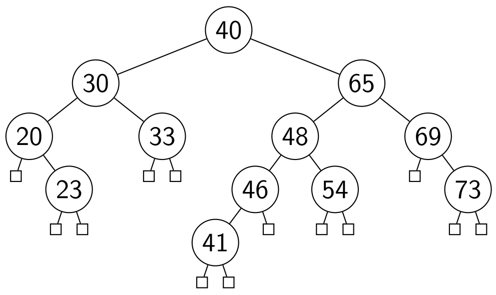


After performing a fusion, we end up with:



Which needs to be fixed by yet another fusion.

23. (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).

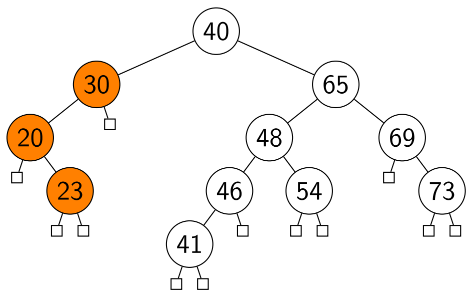
A. No tri-node restructuring.

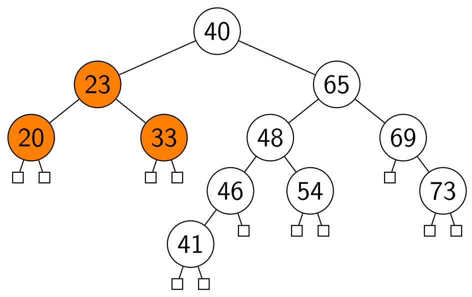
B. One tri-node restructuring.

C. Two tri-node restructurings.

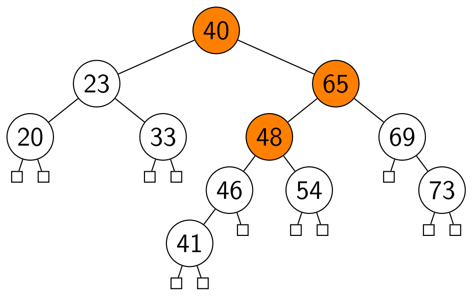
D. Three tri-node restructurings.

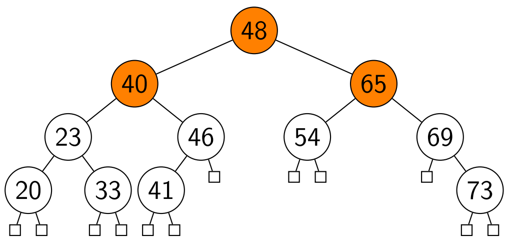
Answer: After removing 33, the node containing 30 becomes unbalanced, which needs to be fixed by a double rotation (i.e. a tri-node restructuring):





In turn, the node containing 40 becomes unbalanced, which needs to be fixed by a double rotation (i.e. another tri-node restructuring):





24. (1 point) Given a binary search tree, what traversal should be used to obtain a sorted sequence of the keys in the tree?

    A. Pre-order traversal.

    B. Post-order traversal.

    C. In-order traversal.

    D. Depth-first traversal.

Open questions (35%, 32 points)

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

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

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

Answer: The base and recurrence equation for the worst-case time complexity of method `precursive` are:

$$T(0) = c_1 \qquad T(n) = 2T(n-1) + c_2.$$

The parts of the recurrence equation are justified as follows:

c_1 accounts for the constant time operations of the base case including calling the method `precursive` (line 1), the conditional statement (line 1), and the explicit return from method `precursive` (lines 3).

$2T(n-1)$ accounts for the two recursive calls with input size $n-1$ (line 3).

c_2 accounts for the constant time operations in the recursive case, i.e. calling the method `precursive` (line 1), the conditional statement (line 1), and the explicit return from method `precursive` (lines 3, 4).

Point distribution:

- 1 point for correctly describing the constant c_1 in the base case.
- 1 point for correctly identifying 2 recursive calls in $2T(n-1)$.
- 1 point for correctly identifying the input size $n-1$ of the recursive calls in $2T(n-1)$.
- 1 point for correctly describing the constant c_2 in the recursive case.

(b) (6 points) Derive the closed form of the given recurrence equation. You should **either**: guess the closed form and prove its correctness by induction, **or** derive the closed form by repeated unfolding.

Answer: By repeated unfolding:

$$\begin{aligned}
T(n) &= 2\cdot T(n-1) + c_2 && \text{(by unfolding } T(n)\text{)}\\
&= 2\cdot(2\cdot T(n-2) + c_2) + c_2 && \text{(by unfolding } T(n-1)\text{)}\\
&= 4\cdot T(n-2) + 3\cdot c_2 && \text{(by arithmetic)}\\
&= 2^k\cdot T(n-k) + (2^k-1)\cdot c_2 && \text{(by repeating } k \text{ times)}\\
&= 2^n\cdot T(n-n) + (2^n-1)\cdot c_2 && \text{(by letting } k=n\text{)}\\
&= 2^n\cdot T(0) + (2^n-1)\cdot c_2 && \text{(by arithmetic)}\\
&= 2^n\cdot c_1 + (2^n-1)\cdot c_2 && \text{(by unfolding } T(0)\text{)}\\
&= 2^n\cdot(c_1+c_2) - c_2 && \text{(by arithmetic)}
\end{aligned}$$

Point distribution: For an answer using repeated unfolding:

- 1 point for unfolding a few first steps, i.e. for using $T(n) = 2T(n-1) + c_2$ one or more times.

- 1 point for simplifying the previous unfoldings.
- 2 points for correctly deriving the expression for number of repetitions k.
- 1 point for a correct final answer, even when containing k and n (and constants).
- 1 point for expressing the final answer only as a function of n (and constants).

By induction: we wish to prove that $T(n) = 2^n \cdot (c_1 + c_2) - c_2$ for all n.

- Base case ($n = 0$). Here we have to prove $T(0) = 2^0 \cdot (c_1 + c_2) - c_2 = c_1$. This statement holds trivially by arithmetic.
- Inductive case ($n > 0$). Here we have to prove $T(n) = 2^n \cdot (c_1 + c_2) - c_2$ assuming $2^{n-1} \cdot (c_1 + c_2) - c_2$ (the induction hypothesis).

$$\begin{aligned}
T(n) &= 2T(n-1) + c_2 && \text{(by unfolding } T(n)\text{)}\\
&= 2 \cdot (2^{n-1} \cdot (c_1 + c_2) - c_2) + c_2 && \text{(by the induction hypothesis)}\\
&= 2^n \cdot (c_1 + c_2) - 2 \cdot c_2 + c_2 && \text{(by arithmetic)}\\
&= 2^n \cdot (c_1 + c_2) - c_2 && \text{(by arithmetic)}
\end{aligned}$$

By the base and inductive case we conclude that the statement holds for all n. □

Point distribution: For an answer using induction:

- 2 points for writing down the correct closed form solution $T(n) = 2^n \cdot (c_1 + c_2) - c_2$ (or correct variant thereof).
- 1 point for proving the base case, showing the closed form provides the correct solution when $n = 0$.
- 1 point for stating the induction hypothesis (IH) and that we want to prove that the closed form solution holds for any $n > 0$.
- 1 point for applying induction hypothesis correctly in the induction step (not required, but possibly involves stating the assumption that the IH holds for $n - 1$).
- 1 point for finishing induction step (correct arithmetic).

(c) (2 points) State the tightest worst-case Big-$\mathcal{O}$ **time** complexity of `precursive` in function of n. Justify your answer.

Answer: We established that the closed form is $T(n) = 2^n \cdot (c_1 + c_2) - c_2$. The constants in the closed form can be disregarded, therefore the time complexity of method `precursive` in Big-Oh notation is $\mathcal{O}(2^n)$.

Point distribution:

- 1 point for the correct answer.
- 1 point for the justification.

(d) (2 points) Explain what the algorithm `precursive` calculates. Describe a faster solution to perform the same calculation and state its tightest worst-case **time** complexity.

Answer: The algorithm calculates 2 to the power of n (2^n). A faster solution could be obtained by replacing `return precursive(n-1) + precursive(n-1);` by `return 2 * precursive(n-1);`, with complexity $\mathcal{O}(n)$. An even faster $\mathcal{O}(\log_2 n)$ solution would use a `precursive(n/2)` recursive call together with some arithmetic operations, as described in the book/lecture. The best solution would be an iterative version of the recursive $\mathcal{O}(\log_2 n)$ solution.

Point distribution:

- 1 point for identifying what the algorithm does.
- 1 point for describing a faster solution with the correct complexity.

26. Method `operateOnMatrix` below performs an operation on a symmetric n by n matrix (row x column).

```
public class SymmetricMatrix {                                                1
  private int[][] matrix;                                                     2
  /* ... */                                                                   3
  public int[] operateOnMatrix(int x) {                                       4
    for (int i = 0; i < matrix.length; i++) {                                 5
      int j = operateOnArray(matrix[i], i, x);                                6
      if (j > -1)                                                             7
        return new int[]{i,j};                                                8
    }                                                                         9
    return null;                                                              10
  }                                                                           11
                                                                              12
  private static int operateOnArray(int[] row, int startIdx, int x) {         13
    for (int j = startIdx; j < row.length; j++)                               14
      if (row[j] == x)                                                        15
        return j;                                                             16
    return -1;                                                                17
  }                                                                           18
}                                                                             19
```

Entries in a symmetric matrix are symmetric with respect to the main diagonal. Example: $M = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 4 & 5 \\ 3 & 5 & 6 \end{pmatrix}$

(a) (4 points) Write the polynomial for the worst-case **time** complexity of method `operateOnMatrix` as a function of n. Define all constants. Explain each term of the polynomial, refer to lines of code.

Answer: In the worst-case, method `operateOnMatrix` will have to search for x through the whole upper triangle of the symmetric matrix.

$$\begin{aligned} T(n) &= c_0 + c_1 n + c_2(n + (n-1) + \ldots + 2 + 1) \\ &= c_0 + c_1 n + c_2 \sum_{i=1}^{n} i \end{aligned}$$

where:

- n denotes one of the dimensions of the n by n matrix;
- c_0 accounts for the primitive instructions associated with calling the method `operateOnMatrix` (line 4), declaring and initializing variable `j` (line 5), and returning from the method `operateOnMatrix` (lines 10,11);
- c_1 accounts for the operations performed in the n iterations of the `for` loop, including declaration, initialization, conditional test and increment of variable `i` (line 5), update of variable `j` (line 6), and call and return from method `operateOnArray` (lines 6,18);

c_2 accounts for the operations performed within the second `for` loop (see helper method `operateOnArray`), including initialization, conditional test and increment of variable `j` (line 14), as well as the if statement (line 15) which always evaluates to false in the worst-case.

Point distribution:

- 1 points for an expression that contains both an (unsimplified) $\mathcal{O}(n)$ and $\mathcal{O}(n^2)$ term.
- 1 points for identifying the sum.
- 1 point for defining the three constants.
- 1 point for explaining the three constants.

(b) (4 points) Simplify your polynomial expression as much as possible. Derive and explain the tightest worst-case **time** complexity of method `operateOnMatrix` in Big-$\mathcal{O}$ notation.

Answer:

$$\begin{aligned}
T(n) &= c_0 + c_1 n + c_2(n + (n-1) + \ldots + 2 + 1) \\
&= c_0 + c_1 n + c_2 \sum_{i=1}^{n} i \\
&= c_0 + c_1 n + c_2 \frac{n(n+1)}{2} \\
&= c_0 + c_1 n + c_2 \frac{n^2+n}{2} \\
&= c_0 + \left(c_1 + \frac{c_2}{2}\right) \cdot n + \frac{c_2}{2} \cdot n^2
\end{aligned}$$

The constants can be disregarded, since $\{c_0, c_1, c_2/2\} \ll n$. The term n^2 grows faster than any other term in the polynomial when $n \to \infty$, therefore the time complexity of method `operateOnMatrix` in Big-Oh notation is $\mathcal{O}(n^2)$.

Point distribution:

- 1 point for correctly simplifying the expression to the $\mathcal{O}(n^2)$ term (**give this point also if the student does this as an answer to question 26.(a) above**).
- 1 point for correctly rewriting the formula in the form $const1 + const2 \cdot n + const3 \cdot n^2$.
- 1 point for concluding that the polynomial expression is $\mathcal{O}(n^2)$.
- 1 point for explaining why the polynomial expression is $\mathcal{O}(n^2)$.

(c) (3 points) Identify what `operateOnMatrix` does. Describe a faster solution to perform the same operation only on the upper triangle of the matrix assuming that the elements of each row are sorted, and state its tightest worst-case **time** complexity in Big-$\mathcal{O}$ notation.

Answer:

- The algorithm searches for element x in a symmetric matrix (due to the symmetry, the algorithm only needs to search the upper/lower triangle of the matrix - upper in this case), and returns its position in the matrix.
- If the elements of each row are sorted, the linear search performed by method `operateOnArray`

on each row of the matrix can be replaced by binary search.

- The tightest worst-case time complexity of the solution using binary search is $O(n \log_2 n)$.

Point distribution:

- 1 point for correctly identifying what the algorithm does.
- 1 point for describing a faster solution.
- 1 point for stating the correct time complexity.

27. Prove that $2^{n+3} + 2^{\log_2 n}$ is $\Theta(2^n)$.

(a) (3 points) State in detail the mathematical conditions that should be proved.

Answer: In order to prove that $2^{n+3} + 2^{\log_2 n}$ is $\Theta(2^n)$, we have to prove that $2^{n+3} + 2^{\log_2 n}$ is both $\mathcal{O}(2^n)$ and $\Omega(2^n)$.
Therefore, we have to show that there exist positive constants c_1, c_2, and n_0, such that:

$$c_1 \cdot 2^n \le 2^{n+3} + 2^{\log_2 n} \le c_2 \cdot 2^n \quad \text{for all } n \ge n_0.$$

Point distribution:

- 1 point for a correct proof statement $\exists c_1.\exists c_2.\exists n_0.\forall n \ge n_0. \ldots$.
- 2 points for the correct inequalities (1 point each).
- Only 1 point for the inequalities if no distinction between c_1 and c_2 is made.

(b) (4 points) Prove that these conditions hold. Explain all the steps in your proof.

Answer: Simplify $2^{n+3} + 2^{\log_2 n}$:

$$\begin{aligned} 2^{n+3} + 2^{\log_2 n} &= 2^n \cdot 2^3 + n \\ &= 8 \cdot 2^n + n \end{aligned}$$

Let $c_1 = 1$, $c_2 = 16$ and $n_0 = 1$. The inequality below holds for any positive n as $2^n \le 8 \cdot 2^n$ and $n \le 2^n$, so surely for any $n \ge n_0$:

$$2^n \le 8 \cdot 2^n + n \le 16 \cdot 2^n$$

The expression above can also be written as:

$$2^n \le 2^{n+3} + n \le 2^{n+4}$$

Point distribution:

- 1 point for correct choice of c_1.
- 1 point for correct choice of c_2.
- 1 point for correct choice of n_0.
- 1 point for a correct explanation, which includes why it holds $\forall n \ge n_0$.

Note: The simplification is not necessary, as long as the answer is correct.

Implementation questions (30%)

There are two implementation questions on Weblab.

End of the exam