Delft University of Technology
EEMCS Faculty

CSE1305 Algorithms & Data Structures

Midterm Written Exam

7 December 2021, 09:00–10:30

Examiners:

Examiner responsible: Joana Gonçalves and Ivo van Kreveld
Examination reviewer: Stefan Hugtenburg

Parts of the examination and determination of the grade:

Exam part	Number of questions	Question specifics	Grade (%)	Grade (points)
Multiple-choice	14 questions (equal weights)	One correct answer per question	50%	$5 \cdot \frac{\text{score}}{14}$
Open questions	3 questions (different weights)	Multiple parts	50%	$5 \cdot \frac{\text{score}}{14}$

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 cannot leave the examination room during the first 30 minutes.
- If you are eligible for extra time, place the declaration form "Verklaring Tentamentijd Verlenging" together with your student card on your table. We don't want to interrupt you to be able to check this form.

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.
- 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%, 14 points)

1. (1 point) Which of the following statements is **true**?
   A. The expression $\sum_{k=1}^{n} \sum_{i=1}^{k} i$ is $\Omega(n^4)$.
   B. The Big-Ω, Big-Θ, and Big-$\mathcal{O}$ notations are used to denote respectively the best case, the average case, and the worst case runtimes of an algorithm.
   C. The Big-Ω, Big-Θ, and Big-$\mathcal{O}$ notations denote respectively a lower bound, a tight bound, and an upper bound on the runtime of an algorithm for a given case of interest.
   D. The expression $n^3 \log_2 n$ is $\mathcal{O}(n \log_8 n^3)$.

2. (1 point) Given an algorithm with time complexity $\Theta(f(n))$ and space complexity $\Theta(g(n))$, which of the following claims is **true**?
   A. $g(n)$ is $\Omega(f(n))$.
   B. $g(n)$ is $\mathcal{O}(f(n))$.
   C. $f(n)$ is $\Theta(g(n))$.
   D. $f(n)$ is unrelated to $g(n)$.

3. (1 point) Consider the implementation of method `select` below.

```
 1  public static Integer select(List<Integer> list, int k) {
 2    if (list == null || list.isEmpty())
 3      return null;
 4    if (list.size() == 1)
 5      return list.get(0);
 6
 7    int elem = list.get(0);
 8    List<Integer> lesser = new ArrayList<>();
 9    List<Integer> equal = new ArrayList<>();
10    List<Integer> greater = new ArrayList<>();
11    for (int i : list) {
12      if (i < elem) { lesser.add(i); }
13      else if (i > elem) { greater.add(i); }
14      else { equal.add(i); }
15    }
16    if (k <= lesser.size())
17      return select(lesser, k);
18    if (k <= lesser.size() + equal.size())
19      return elem;
20    return select(greater, k - lesser.size() - equal.size());
21  }
```

   Assume that the input size n denotes the number of elements in `list`, and that the created sublists `lesser` and `greater` end up each with approximately $n/2$ of the elements of the input `list`. What is the recurrence equation for the runtime of algorithm `select`?
   A. $T(n) = T(n/2) + c_1 n + c_2$
   B. $T(n) = \frac{T(n/2)}{2} + c_1 n + c_2$
   C. $T(n) = 2T(n/2) + c_1 n + c_2$
   D. $T(n) = T(n-2) + c_1 n + c_2$

4. (1 point) Consider a sequence S of n elements. We would like to extract the subsequence of elements of sequence S between indices i and j, with $0 < i < j < n$ to make a new sequence P (note that extracting the subsequence means that the entire subsequence is removed from S). What are the time and space complexities of performing this operation when the sequences S and P are both implemented using either arrays or singly-linked lists (SLL), where you have full access to the array, as well as full access to the head reference and the references between the nodes of the SLL?

   A. **array:** time $\mathcal{O}(n)$ and space $\mathcal{O}(1)$ **SLL:** time $\mathcal{O}(n)$ and space $\mathcal{O}(1)$
   B. **array:** time $\mathcal{O}(n)$ and space $\mathcal{O}(n)$ **SLL:** time $\mathcal{O}(1)$ and space $\mathcal{O}(1)$
   C. **array:** time $\mathcal{O}(n)$ and space $\mathcal{O}(n)$ **SLL:** time $\mathcal{O}(n)$ and space $\mathcal{O}(1)$
   D. **array:** time $\mathcal{O}(n^2)$ and space $\mathcal{O}(n)$ **SLL:** time $\mathcal{O}(n^2)$ and space $\mathcal{O}(n)$

5. (1 point) How many next and tail references need to be set or updated when inserting a new element at the head of a non-empty circularly-linked list?

   A. 1 next reference
   B. 1 next reference and the tail reference
   C. 2 next references
   D. 2 next references and the tail reference

6. (1 point) Consider the implementation of method `operation` below.

```
1  public static void operation(Queue queue) {
2    if (queue == null) return;
3    Stack<Integer> stack = new Stack<>();
4    while (!queue.isEmpty()) {
5      stack.push(queue.peek());
6      queue.dequeue();
7    }
8    while (!stack.isEmpty()) {
9      queue.enqueue(stack.peek());
10     stack.pop();
11   }
12 }
```

   Assume that the `queue` was previously initialized with elements [1,2,3,4,5,6,7,8] enqueued into the queue in the order that they appear in the sequence from left to right. If we execute `operation(queue)` and then remove all elements in `queue` one by one by calling `queue.dequeue()`, which elements will be retrieved and in what order?

   A. 1, 2, 3, 4, 5, 6, 7, 8
   B. 1, 2, 3, 4, 8, 7, 6, 5
   C. 8, 7, 6, 5, 1, 2, 3, 4
   D. 8, 7, 6, 5, 4, 3, 2, 1

7. (1 point) Consider a positional list implemented using a doubly-linked list with header and trailer nodes (guard or dummy nodes). How many times are `prev` references used by method `addBefore` (either to get or set their value), which adds a new element before the given position? Assume the optimal implementation, that is, the references are not used more times than is needed to perform the operation.

   A. 1 `prev` reference
   B. 2 `prev` references
   C. 3 `prev` references
   D. 4 `prev` references

8. (1 point) Consider that a given data structure implemented using an array of capacity C contains n elements, where $C > n$. Which of the following operations does **not** necessarily place the new element at index n? Assume that the data structure is implemented such that it performs its typical operations in the most efficient way.

    A. Insert the element into an unsorted list priority queue.

    B. Add the element to a heap, before any bubbling operations.

    C. Push the element onto the top of a stack.

    D. Enqueue the element at the back of the queue.

9. (1 point) Which of the statements about the removal operation in a heap is **true**?

    A. When removing the element at the root in an array-based heap, all subsequent elements need to be shifted backwards (or right to left) by one position.

    B. After removing the element, we check if the heap-order property is satisfied between the root and its children. We perform down-heap bubbling if needed to restore the heap-order.

    C. After removing the element, the bubbling operation starts at the last position in the heap and performs swaps along a single path up the tree until the heap-order is re-established.

    D. Bubbling after removal takes $\mathcal{O}(n)$ time, where n is the number of elements in the heap.

10. (1 point) Consider a linked tree implemented using nodes of the type shown below.

```
protected static class Node<E> {
  private E element;
  private Node<E> parent;
  private Node<E>[] children;
  /* ... */
}
```

    Just like in family relationships, we define the cousin of a node `v` in a linked tree as a child of a sibling of the parent of `v`. Consider that node `v` is at index `k` in the array of children of its parent, and that the parent of `v` is at index `p` in the array of children of its own parent (which is also the grandparent of `v`). Where can we find the first cousin to the right of a given node `v` in a linked tree, if it exists?

    A. `v.parent.children[k+1]`

    B. `v.parent.parent.children[p-1]`

    C. `v.parent.parent.children[p-1].children[0]`

    D. `v.parent.parent.children[p+1].children[0]`

11. (1 point) Which of the following statements relating traversals and shape of a binary tree with at least 3 nodes is **false**?

    A. If the tree is complete, its postorder traversal visits all leaves before any internal nodes.

    B. If the tree is complete, its breadth-first traversal visits all internal nodes before all leaves.

    C. If the tree is a line, its preorder and breadth-first traversals visit all nodes in the same order.

    D. If the tree has the maximum possible number of nodes at every level, its inorder and breadth-first traversals visit all leaves in the same order.

12. (1 point) Which of the following statements about sorting algorithms is **false**?

    A. Insertion and selection sort are more space efficient than merge sort.

    B. Insertion, selection, and heap sort can all be implemented in-place.

    C. Insertion sort does more comparisons than selection sort.

    D. Merge sort establishes an ordering of the elements in the combine step, when merging the two subsequences back together.

13. (1 point) What is the tightest upper bound on the total number of swap operations performed in the **worst-case** by the selection sort algorithm?
    A. $\mathcal{O}(1)$
    B. $\mathcal{O}(\log n)$
    C. $\mathcal{O}(n)$
    D. $\mathcal{O}(n^2)$

14. (1 point) Consider that you apply the **in-place** heap sort algorithm to sort the following input sequence [23,10,4,16,5,2] in **decreasing** order. What is the state of the array after two removal operations (and any required bubbling)? Note that you are expected to use the most efficient heap construction algorithm.
    A. [5,10,23,16,4,2]
    B. [10,5,23,16,4,2]
    C. [5,16,10,23,4,2]
    D. [23,16,10,5,4,2]

Open questions (50%, 14 points)

15. (5 points) A student is asked to implement the merge sort algorithm. They remember the general idea of the algorithm, but are unsure about what data structure to use. The student decides to use a priority queue, since they remember that priority queues enables fast sorting. The student produces the following Java implementation of method `mergeSort`. Note that the `PriorityQueue` class implements a priority queue using a heap.

```
1  public static void mergeSort(PriorityQueue<Integer> pq) {
2    if(pq.size() < 2)
3      return;
4    int oldSize = pq.size();
5    PriorityQueue<Integer> pq1 = new PriorityQueue();
6    PriorityQueue<Integer> pq2 = new PriorityQueue();
7
8    while(pq.size() > oldSize/2)
9      pq1.offer(pq.poll());
10   while(!pq.isEmpty())
11     pq2.offer(pq.poll());
12
13   mergeSort(pq1);
14   mergeSort(pq2);
15   while(!pq1.isEmpty() && !pq2.isEmpty()) {
16     if(pq1.peek() < pq2.peek())
17       pq.offer(pq1.poll());
18     else
19       pq.offer(pq2.poll());
20   }
21   while(!pq1.isEmpty())
22     pq.offer(pq1.poll());
23   while(!pq2.isEmpty())
24     pq.offer(pq2.poll());
25 }
```

As you might notice, this is a less desirable implementation of merge sort. Explain why using the recurrence equations for the runtime of this implementation and of the typical implementation of the merge sort algorithm. Give these 2 recurrence equations. Then explain what causes the difference between these recurrence equations and why the issue can be avoided (you do not have to explain all the terms that appear in both recurrence equations, explain only the terms that differ between them). You can assume that the number of elements n in the input priority queue `pq` is a power of two ($n = 2^k$), where k is an integer.

16. (5 points) Derive the closed form of the following recurrence equation:

$$T(2) = c_0$$
$$T(n) = c_1 + T\left(\frac{n+1}{2}\right) \qquad \text{if } n > 2$$

You may assume that the function is only called with integers that result in all recursive calls being called with an odd value for n, resulting in $\frac{n+1}{2}$ being a whole number. This means the function is only called for values $f(k) = 2^k + 1$ where k is a natural number, namely 2, 3, 5, 9, 17, 33, 65 etc.

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.

Note: one of the steps in the derivation/proof is a bit tricky. If you can't figure it out, do a reasonable guess and reason further from there, noting any inconsistencies that were caused by your incorrect guess. You can still get 4 out of 5 points for this question that way.

17. Prove that $\log_4(n)$ is $\Omega(\log_2(n))$.
    (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.
    Hint: you might need to use following logarithms rule: $\log_y(x) = \frac{\log_b(x)}{\log_b(y)}$

End of the exam