Information

Date: Monday, October 24, 2022
Time: 08:00—13:00
Teachers: Pontus Ekberg, Sarbojit Das, Stephan Spengler

This exam has 10 problems, for a total of 100 points. The grade thresholds are

- 50 points for grade 3,
- 75 points for grade 4,
- 90 points for grade 5.

Important instructions:

1. No books/notes or computer/phone/calculator allowed, only writing tools (pencil, etc.).
2. Only write on one side of each sheet of paper.
3. Start your answers to each problem on a new sheet of paper.
4. Hand in your answer sheets sorted in the same order as the problems.
5. Answer questions precisely and concisely. Avoid excessively long answers.
6. Write your answers in English.
7. Draw figures when requested, but feel free to do so for other questions as well, if you think they help.
8. When writing pseudocode you can use a reasonable syntax of your choice, but make sure that the meaning is unambiguous.
9. When asked to describe a new algorithm, you are allowed to use algorithms taught in the course as subroutines.
10. You can keep this question sheet after the exam.

Good luck!

Problem 1 (10 points in total)

For each of the claims below, state whether it is *true* or *false*. No motivation is needed.

a) $12n = O(n)$ (1)

b) $n^2 = O(n^3)$ (1)

c) $n^2 = \Theta(n^3)$ (1)

d) $n^2 = \Omega(n^3)$ (1)

e) $10^{10} \log n = O(n)$ (1)

f) $n = O(\log n)$ (1)

g) $n^{20} = O(2^n)$ (1)

h) If $f(n) = O(g(n))$, then $2^{f(n)} = O(2^{g(n)})$ (1)

i) $\Theta(f(n)) \subseteq \Omega(f(n))$ (1)

j) $O(f(n)) \subseteq \Theta(f(n))$ (1)

Problem 2 (11 points in total)

Answer the below questions. No motivation is needed.

a) What is the worst-case running time of INSERTION-SORT in Θ-notation? (2)

b) What is the worst-case running time of MERGE-SORT in Θ-notation? (2)

c) What is the closed form solution to the below recurrence in Θ-notation? (3)

$$T(n) = \begin{cases} 1, & \text{if } n = 1 \\ 2T(n/2) + 3n, & \text{if } n > 1 \end{cases}$$

d) Assume that it takes k comparisons in the worst case to find a specified element in a sorted array of size m using binary search. How many comparisons would it take in the worst case to instead find an element in a sorted array of size $2m$? (2)

e) Assume that you have an algorithm with running time 2^n, where n is the size of the input, and on your computer it can solve instances of size m in one hour. What size of input could it solve in one hour if you instead used a computer twice as fast? (2)

Problem 3 (13 points in total)

Two words are *anagrams* if the letters of one word can be reordered to get the second word. Examples of anagrams are the words "algorithm" and "logarithm", since the letters in "algorithm" can be reordered to get "logarithm".

a) Describe an algorithm ANAGRAM that takes two words (i.e., two arrays of characters) as input and returns *true* if the words are anagrams (and *false* otherwise). The algorithm should have a worst-case running-time that is $O(n \log n)$, where n is the combined length of the words. You can describe the algorithm in pseudocode or in unambiguous English. Describe why your algorithm works and why it has a worst-case running time of $O(n \log n)$. (8)

(Note: The given words may be any arrays of characters, we do not require that they are actual words in any natural language.)

b) If the words only use the characters A–Z, can we write an algorithm that correctly reports if the words are anagrams with a worst-case running time of $O(n)$? If yes, outline such an algorithm. If no, explain why it is not possible. (5)

Problem 4 (8 points in total)

Draw the Binary Search Trees (BSTs) that we would get after performing the following list of tree operations on an initially empty BST, in the order that the operations are written.

(For each of problems a—d below, start with a new empty BST.)

a) INSERT(10), INSERT(12), INSERT(11) (2)

b) INSERT(4), INSERT(2), INSERT(10), INSERT(12), INSERT(9) (2)

c) INSERT(1), INSERT(43), INSERT(31), INSERT(10), INSERT(11), INSERT(9), DELETE(31) (2)

d) INSERT(10), INSERT(12), INSERT(9), INSERT(11), INSERT(20), INSERT(4), DELETE(10) (2)

Problem 5 (12 points in total)

Consider the below algorithm TREE-SORT, which creates a BST from an array A and then prints the elements of A in sorted order by performing an inorder walk on the resulting tree.

```
TREE-SORT($A$)
$T \leftarrow$ new empty BST

$\triangleright$ Insert the elements of $A$ into $T$
for $i \leftarrow 1$ to $length[A]$
    do INSERT($T, A[i]$)

$\triangleright$ Output the keys of $T$ using an inorder walk
INORDER-TREE-WALK($T$)
```

a) Describe the type of inputs that would result in the *worst-case* running time for TREE-SORT. Give the worst-case running time of TREE-SORT in Θ-notation. Explain your answers carefully. (6)

b) Describe the type of inputs that would result in the *best-case* running time for TREE-SORT. Give the best-case running time of TREE-SORT in Θ-notation. Explain your answers carefully. (6)

Problem 6 (14 points in total)

Assume that we have a hash table of size 15. The hash function used is $h(k) = k \mod 15$ (let the hash table be indexed starting from 0).

a) Let the hash table use *chaining* for collision handling. Draw the hash table after inserting the following keys in order: 7, 18, 30, 26, 20, 3, 14, 60, 33 (4)

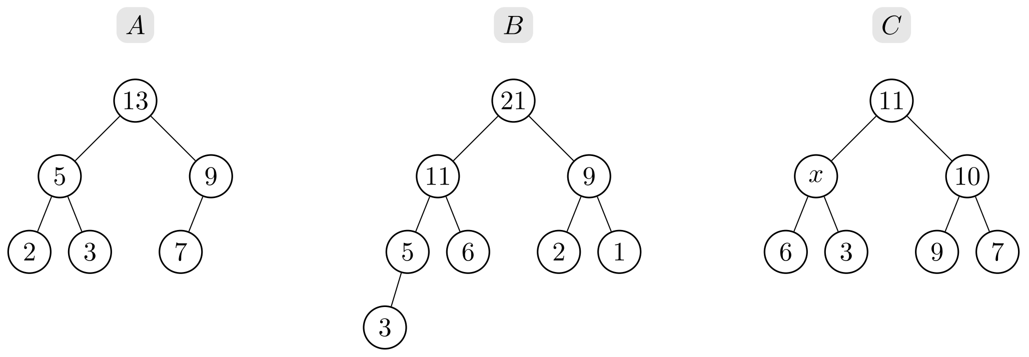
b) Let the hash table instead use *linear probing* for collision handling. Draw the hash table after inserting the same keys in order: 7, 18, 30, 26, 20, 3, 14, 60, 33 (4)

c) A supermarket wants to store the prices of their products in a hashtable. The keys are the prices of the products in öre (100 öre = 1 SEK). You take a look at a few producs with typical prices: 5 SEK, 19.95 SEK, 10 SEK, 9.95 SEK, 10.45 SEK, 16 SEK, 13.95 SEK. (6)

Should you use the hash function $h(k) = k \mod 15$ in this situation? Explain potential problems and propose a hash function that is more suitable for this particular problem. (If needed you may change the size of the hash table slightly.)

Problem 7 (12 points in total)

Consider the following max-heaps called A, B and C:



a) Draw the resulting heap after inserting the key 11 into heap A. (2)

b) Draw the resulting heap after inserting the key 15 into heap B. (2)

c) Draw the resulting heap after removing the maximal element from heap A. (2)

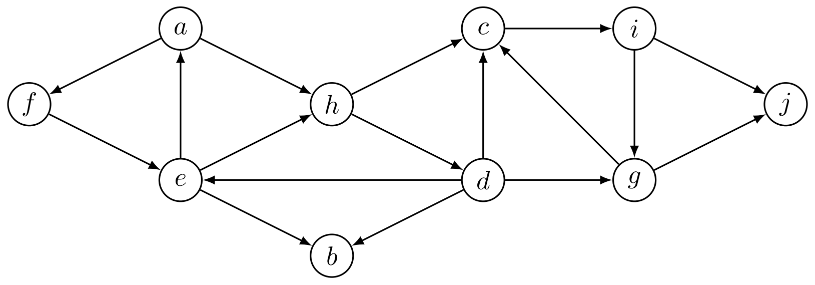
d) Draw the resulting heap after removing the maximal element from heap B. (2)

e) Heap C was obtained by inserting 7 distinct integers into an empty heap. What must be the integer denoted by x? (2)

f) Consider a heap with n nodes. In Θ-notation, what is the complexity of the following operations: INSERT, EXTRACT-MAX. (2)

Problem 8 (8 points in total)

Consider the directed graph G with node set $V = \{a, b, c, \ldots, j\}$ shown here:



We will use the Breadth-First Search (BFS) and Depth-First Search (DFS) algorithms, but let the nondeterministic choices usually present in those algorithms be determined by always looping over adjacent nodes in alphabetical order. (For example, when looping over the adjacent nodes to node a in either BFS or DFS, node f will be considered in the loop before node h.)

a) Starting in node a, what is the BFS path that leads to the discovery of node j? (2)

b) Starting in node a, what is the DFS path that leads to the discovery of node j? (2)

c) Starting in node d, state the order in which all nodes are discovered by BFS. (2)

d) Starting in node d, state the order in which all nodes are discovered by DFS. (2)

Problem 9 (7 points in total)

Describe an algorithm SHORTEST-PATH-WITH-STOP that does the following.

- It takes as input a directed graph G, with node set V and edge set E represented as an adjacency list, plus three specified nodes u, v, $w \in V$.
- It prints the shortest path that starts at u, goes through v, and ends at w, or `NIL` if no such path exists.

Explain clearly why your algorithm works and what its worst-case running time is. For full credit your algorithm should have a worst-case running time that is $O(|V| + |E|)$. You can describe the algorithm in pseudocode or in unambiguous English.

Problem 10 (5 points in total)

Consider the following (flawed) attempt at a proof.

Claim: It is possible to find the maximum element in an unsorted array of any length n in constant time. That is, with a worst-case running time of $O(1)$.

Proof: We prove the claim by induction on the length n of the array.

Base case ($n = 1$): If the array contains only one element, then it is the maximum element. It clearly takes only $O(1)$ time to find.

Induction step ($n > 1$): As the induction hypothesis (I.H.), assume that we can find the maximum of any array of length $n - 1$ in $O(1)$ time.

Given an array of length n, by using the I.H., we compute the maximum element of the first $n - 1$ entries of the array in $O(1)$ time. We then compute the maximum of that element and the last element in the array. This also takes $O(1)$ time. So, the claim also holds for n. □

This proof is incorrect! To find the maximum element in an unsorted array we have to look at each element at least once, so we cannot do better than linear time. Where did the proof go wrong?