EPFL

1

Teacher : Michael Kapralov
Algorithms CS-250 - SC
11 Nov 2022
Duration : 110 minutes

Student Sample

SCIPER : **325664**

Do not turn the page before the start of the exam. This document is double-sided, has 20 pages, the last ones possibly blank. Do not unstaple.

- Place your student card on your table.
- You are only allowed to have a handwritten A4 page written on both sides.
- Communication, calculators, cell phones, computers, etc... are not allowed.
- The exam consists of two parts. The first part consists of multiple choice questions and the second part consists of open ended questions.
- Use a **black or dark blue ballpen** and clearly erase with **correction fluid** if necessary.
- If a question is wrong, the teacher may decide to nullify it.

Good luck!

Respectez les consignes suivantes | Read these guidelines | Beachten Sie bitte die unten stehenden Richtlinien

choisir une réponse | select an answer Antwort auswählen	ne PAS choisir une réponse | NOT select an answer NICHT Antwort auswählen	Corriger une réponse | Correct an answer Antwort korrigieren

ce qu'il ne faut **PAS** faire | what should **NOT** be done | was man **NICHT** tun sollte

CORRECTED

First part: multiple choice questions

For each question, mark the box corresponding to the correct answer. Each question has **exactly one** correct answer.

Question 1 : *(4 pts)*
Give a tight asymptotic bound for the recurrence $F(n) = 3 \cdot F(n/4) + \Theta(n)$, where we assume $F(n) = \Theta(1)$ for $n \leq 10$.

- ☐ $F(n) = \Theta(n^{\log_3 4} \log n)$
- ☐ $F(n) = \Theta(n^{\log_3 4})$
- ■ $F(n) = \Theta(n)$
- ☐ $F(n) = \Theta(n^{\log_4 3})$
- ☐ $F(n) = \Theta(n \log n)$

Solution. By the Master theorem.

Question 2 : *(4 pts)*
Give a tight asymptotic bound for the recurrence $G(n) = 4 \cdot G(n/2) + \Theta(n^2)$, where we assume $G(n) = \Theta(1)$ for $n \leq 10$.

- ☐ $G(n) = \Theta(n)$
- ☐ $G(n) = \Theta(n^2)$
- ☐ $G(n) = \Theta(n^4)$
- ■ $G(n) = \Theta(n^2 \log n)$
- ☐ $G(n) = \Theta(n^4 \log n)$

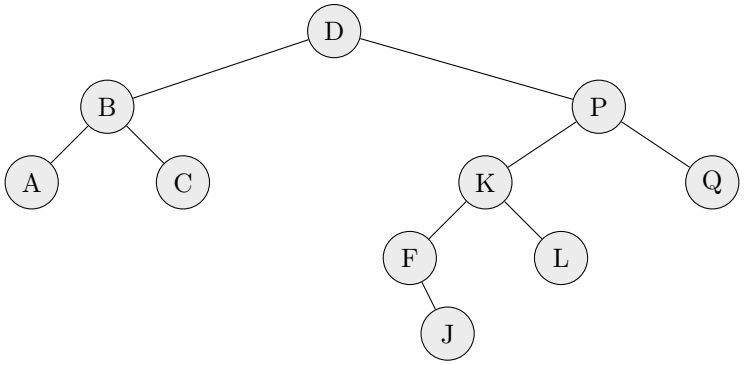
Solution. By the Master theorem.

Question 3 : *(4 pts)*
Give a tight asymptotic bound for the recurrence $H(n) = H(\lfloor n^{1/3} \rfloor) + \Theta(\log n)$, where we assume $H(n) = \Theta(1)$ for $n \leq 10$.

- ☐ $H(n) = \Theta(n^{1/3})$
- ☐ $H(n) = \Theta(n^{1/3} \log n)$
- ☐ $H(n) = \Theta(\log^2 n)$
- ■ $H(n) = \Theta(\log n)$
- ☐ $H(n) = \Theta(\log n \log \log n)$

Solution. Letting $m = \log n$ and $S(m) = H(n)$, we get that $S(m) \approx S(m/3) + \Theta(m)$ and apply the Master theorem.

Question 4 : *(6 pts)*
Let T be the following binary search tree:



Suppose that we delete the root of T using the procedure seen in class. What is the pre-order traversal of the resulting binary search tree:

- ☐ $F, A, B, C, P, L, K, Q, J$
- ☐ $F, B, A, C, P, J, K, L, Q$
- ■ $F, B, A, C, P, K, J, L, Q$
- ☐ $F, A, B, C, P, J, K, L, Q$
- ☐ $P, F, A, B, C, L, K, Q, J$
- ☐ $P, F, Q, K, L, J, B, C, A$
- ☐ $F, P, Q, K, J, L, B, A, C$
- ☐ $F, P, Q, K, L, J, B, C, A$

CORRECTED

Question 5 : *(8 pts)*
Arrange the following functions in increasing order according to asymptotic growth.

$$2^N, \quad \log N, \quad N^{1/\log\log N}, \quad 2^{\sqrt{\log N}}, \quad (\log\log N)^5, \quad N^3, \quad N\log(N^5+N)$$

The correct order is:

- ☐ $N^{1/\log\log N}, \quad (\log\log N)^5, \quad \log N, \quad 2^{\sqrt{\log N}}, \quad N\log(N^5+N), \quad N^3, \quad 2^N$
- ☐ $\log N, \quad (\log\log N)^5, \quad N^{1/\log\log N}, \quad N^3, \quad N\log(N^5+N), \quad 2^{\sqrt{\log N}}, \quad 2^N$
- ☐ $\log N, \quad (\log\log N)^5, \quad N^{1/\log\log N}, \quad N\log(N^5+N), \quad N^3, \quad 2^{\sqrt{\log N}}, \quad 2^N$
- ☐ $N^{1/\log\log N}, \quad (\log\log N)^5, \quad \log N, \quad N\log(N^5+N), \quad N^3, \quad 2^{\sqrt{\log N}}, \quad 2^N$
- ☐ $(\log\log N)^5, \quad N^{1/\log\log N}, \quad \log N, \quad N\log(N^5+N), \quad N^3, \quad 2^{\sqrt{\log N}}, \quad 2^N$
- ☐ $(\log\log N)^5, \quad N^{1/\log\log N}, \quad 2^{\sqrt{\log N}}, \quad \log N, \quad N\log(N^5+N), \quad N^3, \quad 2^N$
- ■ $(\log\log N)^5, \quad \log N, \quad 2^{\sqrt{\log N}}, \quad N^{1/\log\log N}, \quad N\log(N^5+N), \quad N^3, \quad 2^N$
- ☐ $N^{1/\log\log N}, \quad \log N, \quad (\log\log N)^5, \quad N^3, \quad N\log(N^5+N), \quad 2^{\sqrt{\log N}}, \quad 2^N$
- ☐ $N^{1/\log\log N}, \quad (\log\log N)^5, \quad N\log(N^5+N), \quad \log N, \quad N^3, \quad 2^{\sqrt{\log N}}, \quad 2^N$
- ☐ $(\log\log N)^5, \quad N^{1/\log\log N}, \quad \log N, \quad 2^{\sqrt{\log N}}, \quad N\log(N^5+N), \quad N^3, \quad 2^N$
- ☐ $(\log\log N)^5, \quad N^{1/\log\log N} \quad \log N, \quad N\log(N^5+N), \quad N^3, \quad 2^{\sqrt{\log N}}, \quad 2^N$

Solution. To see that $2^{\sqrt{\log N}} = O\left(N^{1/\log\log N}\right)$, note that $N^{1/\log\log N} = 2^{\log N/\log\log N}$. Now the result follows since $\sqrt{N} = o(\log N/\log\log N)$.

CORRECTED

Heaps. Let $A[1 \ldots 9] =$ | 17 | 12 | 7 | 11 | 8 | 0 | 3 | 2 | 9 | be a heap consisting of 9 numbers. We remark that in each of the following subproblems we start with a fresh copy of this heap A, i.e., without any modifications.

Question 6 : *(4 pts)*
Illustrate what A looks like after executing the code HEAP-EXTRACT-MAX$(A, 9)$.

- [x] $A =$ | 12 | 11 | 7 | 9 | 8 | 0 | 3 | 2 |
- [ ] $A =$ | 12 | 11 | 9 | 7 | 8 | 0 | 3 | 2 |
- [ ] $A =$ | 11 | 2 | 7 | 9 | 8 | 0 | 3 | 12 |
- [ ] $A =$ | 12 | 11 | 9 | 8 | 7 | 2 | 3 | 0 |
- [ ] $A =$ | 12 | 11 | 7 | 8 | 9 | 0 | 3 | 2 |

Question 7 : *(4 pts)* Illustrate how A looks like after executing the code HEAP-INCREASE-KEY$(A, 6, 19)$. (6 is the index of the element we increase the key of and 19 is the new key value.)

- [ ] $A =$ | 19 | 12 | 17 | 11 | 9 | 7 | 3 | 2 | 8 |
- [ ] $A =$ | 19 | 12 | 17 | 9 | 11 | 7 | 3 | 2 | 8 |
- [ ] $A =$ | 19 | 17 | 12 | 9 | 11 | 7 | 3 | 2 | 8 |
- [x] $A =$ | 19 | 12 | 17 | 11 | 8 | 7 | 3 | 2 | 9 |
- [ ] $A =$ | 19 | 17 | 12 | 11 | 8 | 7 | 3 | 2 | 9 |

Second part, open questions

This part consists of three questions, each worth 22 points. Please follow the following instructions:

- Your explanations should be clear enough and in sufficient detail that a fellow student can understand them. In particular, do not only give pseudocode without explanations. A good guideline is that a description of an algorithm should be such that a fellow student can easily implement the algorithm following the description.
- You are allowed to refer to material covered in the lectures including algorithms and theorems (without reproving them). You are however *not* allowed to simply refer to material covered in exercises.
- Please answer all questions within the designated boxes (otherwise your answer may not be accurately scanned). At the end of the exam there are five extra pages if you need additional space for your answers.
- Leave the check-boxes empty, they are used for the grading.

Question 8: Recurrences. *(22 pts)*

☐0 ☐1 ☐2 ☐3 ☐4 ☐5 ☐6 ☐7 ☐8 ☐9 ☐10 ☐11
☐12 ☐13 ☐14 ☐15 ☐16 ☐17 ☐18 ☐19 ☐20 ☐21 ■22

Do not write here.

Consider the recurrence $T(n) = 4T(n/4) + 3T(n/2) + n, T(1) = \Theta(1)$. Prove that $T(n) = \Theta(n^2)$.

Solution.
We use the substitution method. First, we show that $T(n) = O(n^2)$. In particular, we show that for all n it holds that $T(n) \leq an^2 - bn$ for some positive constants a, b.
Base case. $T(1) = \Theta(1)$ from the problem statement for any b it should always be possible to find an a such that $T(1) \leq a - b$.
Induction step. Assume $T(k) \leq ak^2 - bk$ for all $k < n$. Then

$$\begin{aligned} T(n) &= 4T(n/4) + 3T(n/2) + n && (1)\\ &\leq 4(an^2/16 - bn/4) + 3(an^2/4 - bn/2) + n && \text{(By induction hypothesis)}\\ &= an^2 - 5bn/2 + n && (2)\end{aligned}$$

and our goal is to show that there exist constants a, b such that $T(n) \leq an^2 - bn$. Using the above it suffices to pick a, b such that

$$an^2 - 5bn/2 + n \leq an^2 - bn, \tag{3}$$

solving which leads to $b \geq 2/3$. Thus, $T(n) = O(n^2)$.
Next, we show that $T(n) = \Omega(n^2)$ which together with the above implies the desired result. In particular, we show that for all n it holds that $T(n) \geq cn^2$ for some positive constant c.
Base case. $T(1) = \Theta(1)$ implying there exists a constant c such that $T(1) \geq c$.
Induction step. Assume $T(k) \geq ck^2$ for all $k < n$. Then

$$\begin{aligned} T(n) &\geq 4cn^2/16 + 3n^2/4 + n && \text{(By induction hypothesis)}\\ &\geq cn^2, && (4)\end{aligned}$$

which concludes the proof.

CORRECTED

Question 9: Running a race. *(22 pts)*

0 1 2 3 4 5 6 7 8 9 10 11
12 13 14 15 16 17 18 19 20 21 22

Do not write here.

You are managing a team of three runners (Alice, Bob and Charlie) who must run a race consisting of n segments. You have determined a preferred runner for every segment $i = 1, \ldots, n$. However, according to the rules of the race the runners can only be swapped in a round-robin fashion, i.e. Bob can only run after Alice, Charlie can only run after Bob and Alice can only run after Charlie (but any given runner can run multiple segments in a row). For example, if $n = 3$, the feasible sequences of runners are ABC, AAB, AAA, ABB, BCA, BBC, BBB, BCC, CAB, CCA, CCC, CAA, where A stands for Alice, B for Bob and C for Charlie. For $n = 4$ feasible sequences include $ABCA$ (i.e. note that a runner can run multiple disjoint segments). Formally, a sequence of runners is feasible if A's are followed only by other A's or by B's, B's are followed by B's or C's, and C's are followed by A's or C's. You need to find a feasible sequence of runners such that the maximum possible number of segments is run by your preferred runner for that segment.

Design and **analyze** an efficient algorithm for the following problem:

> **Input:** An array $P[1, \ldots, n]$ of preferences for the segments, where for every $i \in \{1, 2, \ldots, n\}$ one has $P[i] \in \{\mathrm{A}, \mathrm{B}, \mathrm{C}\}$.
>
> **Output:** A feasible sequence matching P in the largest possible number of segments.

Hint. Use dynamic programming. For every $i \in \{1, 2, \ldots, n\}$ and $R \in \{A, B, C\}$ let $m[i, R]$ denote the largest possible overlap that a feasible sequence of runners of length i that finishes with R can have with $P[1 \ldots i]$, and show how to compute $m[i, R]$ using a bottom-up approach.

Solution.

Let $m[i, R]$ be defined as in the hint. Let $Prev : \{A, B, C\} \to \{A, B, C\}$ denote the function returning the previous runner, so $Prev(A) = C$, $Prev(B) = A$, $Prev(C) = B$. The numbers $m[i, R]$ can be computed by the bottom-up approach using the following recurrence:

$$m[0, R] = 0 \qquad \forall R$$

$$m[i, R] = \begin{cases} max\{m[i-1, R], m[i-1, Prev(R)]\}, & if R \neq P[i], i > 0 \\ max\{m[i-1, R], m[i-1, Prev(R)]\} + 1, & if R = P[i], i > 0 \end{cases}$$

Note that the m[i,R] is just a number. To output a feasible sequence of runners, track which of the runners your choose for each segment in a separate array as you go along.

Return the sequence of runners corresponding to $max\{m[n, A], m[n, B], m[n, C]\}$.

The running time of this algorithm is $\Theta(n)$.

CORRECTED

Question 10: Reconstructing a binary search tree. *(22 pts)*

0 1 2 3 4 5 6 7 8 9 10 11
12 13 14 15 16 17 18 19 20 21 22

Do not write here.

In this problem you will show that a Binary Search Tree T with n keys can be reconstructed uniquely given its in-order traversal and its pre-order traversal. We assume for simplicity that the keys in T are distinct.

Design and analyze an algorithm for the following problem:

> **Input:** An integer n, two length n sequences σ and τ of distinct integers.
>
> **Output:** A binary search tree T such that σ is its in-order traversal and τ is its pre-order traversal, if such a tree exists, and `NONE` otherwise.

The running time of your algorithm should be polynomial in n. While your algorithm should run in polynomial time, it is not important that you achieve an optimal running time in this question.

We recall the pseudocode of the in-order tree walk of a tree T rooted at a node x:

```
IN-ORDER-TREE-WALK(x)
 If x ≠ NIL then:
      IN-ORDER-TREE-WALK(x.left)
      print(key[x])
      IN-ORDER-TREE-WALK(x.right)
```

The pseudocode of the pre-order tree walk of a tree T rooted at a node x is:

```
PRE-ORDER-TREE-WALK(x)
 If x ≠ NIL then:
      print(key[x])
      PRE-ORDER-TREE-WALK(x.left)
      PRE-ORDER-TREE-WALK(x.right)
```

Solution. We present one of the possible solutions. The procedure build_tree takes as inputs two listings of nodes, called "inorder" and "preorder", and returns a tree such that its inorder listing is inorder and its preorder listing is preorder. If such a tree does not exist then it returns "none".

If the length of inorder is different from the length of preorder return none.

Otherwise

1. Set the root of the tree to the first element of the preorder list.
2. Find the index i of the root in the inorder list. If there is no such index return none.
3. Set

$$\begin{aligned} left\_preorder &= preorder[1:i] \\ right\_preorder &= preorder[i+1:] \\ left\_inorder &= preorder[i+1:] \\ right\_inorder &= inorder[pos+1:] \\ root\_node.left &= build\_tree(left\_inorder, left\_preorder) \\ root\_node.right &= build\_tree(right\_inorder, right\_preorder). \end{aligned}$$

5. Return root_node.

Finding the root and spliting the lists can be done in linear time. This operation is done once for each node and thus the time complexity is in $O(n^2)$.

Corrected

Corrected

Corrected

Corrected

Corrected

Corrected

Corrected

Corrected

Corrected