Midterm Exam, CS-250: Algorithms I, 2024

Do not turn the page before the start of the exam. This document is double-sided and has 8 pages. Do not unstaple.

- The exam consists of three parts. The first part consists of multiple-choice questions (Problem 1), the second part consists of a short open question (Problem 2), and the last part consists of three open-ended questions (Problems 3, 4, 5).
- For the open-ended questions, 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.

Good luck!

Problem 1: Multiple Choice Questions (35 points)

For each question, select the correct alternative. Note that each question has **exactly one** correct answer. Wrong answers are **not penalized** with negative points.

1a. Asymptotics *(7 points).* Let $f : \mathbb{N} \setminus \{0\} \to \mathbb{R}$ be a function from the positive integers to the reals. Which of the following implications does **not** hold?

A. If $f(n) = \log_2 n + \log_2(1 + \log_2 n)$ for all positive integers n, then $f(n) = O(\log_2^2 n)$.

B. If $f(n) = \log_2(n!)$ for all positive integers n, then $f(n) = \Theta(n \log_2 n)$.

C. If $f(n) = n$ for all positive integers n, then $f(n) = \Omega(1)$.

D. If $f(n) = n$ for all positive integers n, then $f(n) = O(1)$.

E. If $f(n) = \log_{10000} n$ for all positive integers n, then $f(n) = \Theta(\log_2 n)$.

Solution. The answer is D. □

1b. Heaps *(7 points).* Consider the array $A =$ | 7 | 10 | 6 | 9 | 4 | 1 | 2 | indexed from 1 to 7. Which of the following arrays is the result of calling BuildMaxHeap on A as shown in class? Recall that BuildMaxHeap works by sequentially calling MaxHeapify.

A. | 10 | 9 | 7 | 6 | 4 | 2 | 1 |

B. | 7 | 10 | 6 | 9 | 4 | 1 | 2 |

C. | 10 | 7 | 6 | 9 | 4 | 1 | 2 |

D. | 1 | 2 | 4 | 6 | 7 | 9 | 10 |

E. | 10 | 9 | 6 | 7 | 4 | 1 | 2 |

Solution. The answer is E. □

1c. Sorting *(7 points).* Consider the array $B =$ | 7 | 3 | 1 | 4 | indexed from 1 to 4. We sort it using insertion sort by calling InsertionSort(A, n) with $A = B$ and $n = 4$ (see below for pseudocode).

InsertionSort(A, n)

```
1: for j = 2, ..., n
2:     k ← A[j]
3:     i ← j − 1
4:     while i > 0 and A[i] > k
5:         A[i + 1] ← A[i]
6:         i ← i − 1
7:     A[i + 1] ← k
8:     print(A)
```

Which of the following sequence of arrays corresponds to the outputs printed in line 8?

A. | 3 | 7 | 1 | 4 | , | 1 | 3 | 7 | 4 | , | 1 | 3 | 4 | 7 |

B. | 1 | 7 | 4 | 3 | , | 1 | 7 | 3 | 4 | , | 1 | 3 | 4 | 7 |

C. | 7 | 3 | 4 | 1 | , | 7 | 4 | 3 | 1 | , | 7 | 4 | 3 | 1 |

D. | 3 | 7 | 1 | 4 | , | 1 | 7 | 3 | 4 | , | 1 | 3 | 4 | 7 |

E. | 7 | 4 | 3 | 1 | , | 7 | 4 | 3 | 1 | , | 7 | 4 | 3 | 1 |

Solution. The answer is A. □

1d. Time analysis *(7 points).* Consider the following algorithms.

Algorithm I: it takes as input a positive integer n and an array A of n integers, performs operations that run in $\Theta(1)$ time in total, computes $q = \lfloor n/3 \rfloor$, calls itself recursively on $A[1, \ldots, q]$, then performs other operations that run in $\Theta(1)$ time in total, and finally returns.

Algorithm II: it takes as input a positive integer n and an array A of n integers, performs operations that run in $\Theta(1)$ time in total, computes $q = \lfloor n/3 \rfloor$, calls itself recursively on $A[1, \ldots, q]$ and $A[n-q+1, \ldots, n]$, then performs other operations that run in $\Theta(1)$ time in total, and finally returns.

Algorithm III: it takes as input a positive integer n and an array A of n integers, performs operations that run in $\Theta(n)$ time in total, computes $q = \lfloor n/3 \rfloor$, calls itself recursively on $A[1, \ldots, q]$, $A[q+1, \ldots, 2q]$, $A[n-q+1, \ldots, n]$, then performs other operations that run in $\Theta(n)$ time in total, and finally returns.

Which of the following statements holds? *Hint:* $\log_3 2 \approx 0.63$.

A. Algorithms I, II, III run in time $\Omega(n^{1/3})$, $O(n^{2/3})$, $\Theta(n^{3/4})$ respectively.

B. Algorithms I, II, III run in time $\Theta(\log n)$, $\Omega(\sqrt{n})$, $O(n \log n)$ respectively.

C. Algorithms I, II, III run in time $\Theta(\log n)$, $\Theta(\log n)$, $\Theta(n \log n)$ respectively.

D. Algorithms I, II, III all run in time $\Theta(n \log n)$.

E. Algorithms I, II, III run in time $\Theta(\log n)$, $\Theta(n \log n)$, $\Omega(n^{3/2})$ respectively.

Solution. The answer is B. □

1e. Data Structures *(7 points).* Let S be a stack and Q be a queue, initially empty. Consider the following sequence of operations on S and Q:

$\textsc{Push}(S, 1)$,
$\textsc{Enqueue}(Q, 4)$,
$\textsc{Push}(S, 1)$,
$\textsc{Push}(S, 4)$,
$\textsc{Enqueue}(Q, \textsc{Pop}(S))$,
$\textsc{Enqueue}(Q, \textsc{Pop}(S))$,
$\textsc{Push}(S, \textsc{Dequeue}(Q))$.

We recall that Pop and Dequeue also return the item that they removed from the stack and queue respectively. Which of the following statements about S and Q holds true after having run the sequence of operations above?

A. The output of $\textsc{Dequeue}(Q), \textsc{Pop}(S)$ is $4, 1$ (in order).

B. The output of $\textsc{Dequeue}(Q), \textsc{Dequeue}(Q)$ is the same as $\textsc{Pop}(S), \textsc{Pop}(S)$ (in order).

C. The output of $\textsc{Pop}(S)$ is 1.

D. The stack S contains only 1's and the queue Q contains 1's and 4's.

E. The stack S contains 1's and 4's and the queue Q contains only 4's.

Solution. The answer is B. □

Problem 2: Magical Computation (10 points)

Consider the following procedure UNKNOWN that takes as input an array $A[\ell \dots r]$ of $n = r-\ell+1$ numbers with the left-index ℓ and the right-index r:

UNKNOWN(A, ℓ, r)

1. **if** $\ell > r$
2. **return** 0
3. **elseif** $\ell = r$
4. **return** $A[\ell]$
3. **else**
4. $p \leftarrow \ell + \lfloor \frac{r-\ell}{4} \rfloor$
5. $q \leftarrow r - \lceil \frac{r-\ell}{4} \rceil$
6. $Term1 \leftarrow$ UNKNOWN(A, ℓ, p)
7. $Term2 \leftarrow$ UNKNOWN$(A, p+1, q)$
8. $Term3 \leftarrow$ UNKNOWN$(A, q+1, r)$
9. **return** $Term1 + Term2 + Term3$

2a. *(5 points)* Let $A[1 \dots 8] =$ | 3 | 7 | 5 | 5 | 2 | 3 | 1 | 9 |. What does a call to UNKNOWN(A, 1, 8) return?

Solution. 35 (the sum of elements in A) □

2b. *(5 points)* Let $T(n)$ be the time it takes to execute UNKNOWN(A, ℓ, r) where $n = r - \ell + 1$ is the number of elements in the array. Give the recurrence relation of $T(n)$.

Solution. $T(n) = T(\frac{n}{2}) + 2T(\frac{n}{4}) + \Theta(1)$, $T(1) = \Theta(1)$. □

Problem 3: Searching in a mountain (20 points)

Consider an array A containing n distinct integers. It is indexed starting at 1, i.e. its first element is $A[1]$ and last element is $A[n]$.

The array has the following structure: there is an unknown index k ($1 < k < n$) such that the array is increasing until its k^{th} element and decreasing afterwards. More precisely $A[1] < A[2] \ldots < A[k]$ and $A[k] > A[k+1] > \ldots > A[n]$. For example, the array $[8, 9, 13, 10, 4]$ satisfies the property for $k = 3$.

You need to design and analyze an algorithm that searches for an integer x in the array A. If x is in A then you should return the index of x in A, otherwise return -1.

Your algorithm must run in $O(\log n)$ time.

(You will receive partial points (10 out of 20) if you design an algorithm with $O(\log n)$ running time assuming the index k is given to you.)

Solution. The first part of the algorithm will be to use binary search to find the pivot index k. We do a binary search with the following modification - we first take the middle element of arr. If the elements to its right and left are smaller than itself, then this is the index k. If the element to its left is bigger then k lies in the left half of arr, otherwise it lies in the right half. Thus we recursively repeat this strategy on either the left or right half to find k, and thus this procedure runs in $O(\log n)$ time.

After we know k, then we just search for x using a standard binary search in both the sub arrays $[A[1], \ldots, A[k]]$ and $[A[k], \ldots, A[n]]$. Both of these searches can be done in $O(\log n)$ time. Thus the total runtime of both the phases of the algorithm is $O(\log n)$. □

Problem 4: Finding the highest peaks (15 points)

Alice, who recently relocated to Switzerland, thrives on adventure and hiking challenges. Eager to take on the hardest challenges, she sets out to identify highest mountains of Switzerland. Armed with an array containing the heights of all N Swiss mountains, she faces a daunting task due to Switzerland's abundance of peaks. Alice seeks an efficient algorithm to determine the heights of the $\sqrt{N}$ highest mountains. As an experienced algorithm designer, she looks for a solution that runs in $O(N)$ time complexity. Can you assist her in this endeavor? Your task is to design and analyze an algorithm which, given an array of N positive integers, outputs the $\sqrt{N}$ highest numbers and runs in $O(N)$ time.

For your convenience you can assume that $\sqrt{N}$ is an integer.

Solution. The algorithm that we will run is the following. First, we will create a max-heap that contains all of the input array's numbers. As we know from the class, this procedure can be implemented in $O(N)$ time. Then, we can simply repeadetly extract the maximum height from the heap and fix it so that it remains a heap. Extracting one number and fixing the heap can be implemented in $O(\log N)$ time. Since we only need to do $\sqrt{N}$ extractions, the total complexity of our algorithm will be $O(N + \sqrt{N} \log N) = O(N)$.

The pseudocode of our algorithm is the following:

Algorithm 1 Find-Highest-Peaks

```
1:  procedure FindHighestPeaks(heights, n)
2:      highest ← empty array of size √n
3:      heap ← empty array of size n

4:      Build-Heap(heap, heights, n);
5:      heapSize = n;

6:      for i = n; i ≥ n − √n; i = i − 1 do
7:          highest[n − i + 1] = heap[1]                ▷ keeping the max height
8:          swap(heap[1],heap[heapSize])                ▷ swap the first element with the last
9:          heapSize = heapSize − 1                     ▷ decrease the heap size by 1
10:         Max-Heapify(heap, 1, heapSize)              ▷ fix the heap
11:     end for

12:     return highest
13: end procedure
```

□

Problem 5: Mirror mirror on the wall (20 points)

For a binary tree T, the mirror tree $M(T)$ is another binary tree where the children of all non-leaf nodes are interchanged. In other words, the left child of every node becomes its right child and the right child becomes its left child. An example of a tree and its mirror tree can be seen in Figure 1.

Figure 1.

Your task is to design and analyze an algorithm that, given a binary tree of n nodes, outputs its mirror image in time $O(n)$.

Solution. The idea is to use is a simple recursive algorithm, as follows. If the tree is empty the algorithm does nothing. Otherwise it recursively assigns the left sub-tree of the current node in the new tree to be the mirror of the right sub-tree of the current root in the original tree and similarly the right sub-tree of the current node of the new tree as the mirror of the left sub-tree of the current root in the original tree. The procedure is described formally in the following pseudocode.

```
Procedure MirrorTree(node)
    If node is not null
        // Swap the left and right children of node
        temp = node.left
        node.left = node.right
        node.right = temp

        // Call MirrorTree on the left child of node
        If node.left is not null
            MirrorTree(node.left)

        // Call MirrorTree on the right child of node
        If node.right is not null
            MirrorTree(node.right)
```

□