CSE 100: Algorithm Design and Analysis: Midterm Exam 2

Spring 2023

Note: This is a **closed** book examination. You have 75 minutes to answer as many questions as possible. The number in the square bracket at the beginning of each question indicates the number of points given to the question. Write all of your answers directly on this paper.

Please make your answers concise as well as precise. If there is something in the question that you believe is open to interpretation, then please go ahead and interpret, but state your assumptions in your answer. Or you can ask the TAs proctoring the exam for clarification. There is no penalty for attempting to answer a question with a wrong answer, so please answer as many questions as you can. Partial credit will be given even if the answer is not fully correct. If you run out of space you can ask for extra papers. Please write down your name on *every* page as some papers may fall off although it rarely happens. 12 pages in total including this cover. The maximum points you can earn is 118. The one point above 100 will be considered as bonus.

Problem	Points earned	Out of
1		16
2		9
3		8
4		10
5		10
6		10
7		10
8		10
9		10
10		10
11 (bonus)		15
Sum		Max 118

Name _____

1. **[16 points]** For each of the following claims, determine if it is true or false. *No* explanation is needed.

   (a) **[2 points]** The heap-sort is an in-place sorting algorithm.
   True **False Sol.** True

   (b) **[2 points]** Suppose we want to sort n numbers where each number is chosen from $[0, 1]$ uniformly at random . Then the expected running time of the bucket sort is $\Theta(n \log n)$.
   True **False Sol.** False

   (c) **[2 points]** If we resolve collisions occurring in a hash table by chaining, a successful search takes average-case time $(1 + \alpha)$ time under the assumption of simple uniform hashing.
   True **False Sol.** True

   (d) **[2 points]** If $A[1...n]$ is *unsorted*, we can find the k th smallest element in $A[1...n]$ in $O(\log n)$ time.
   True **False Sol.** False

   (e) **[2 points]** One can build a max-heap of n elements in $O(n)$ time.
   True **False Sol.** True

   (f) **[2 points]** Consider the Merge sort algorithm on n elements. Then, every path from the root to a leaf node in the tree has at most $O(n \log n)$ edges.
   True **False Sol.** True

   (g) **[2 points]** We can sort n binary numbers of $10 \log_2 n$ bits in $O(n)$ time assuming a basic bit operation takes $O(1)$ time.
   True **False Sol.** True

   (h) **[2 points]** The running time of the deterministic Quick-sort is $\Theta(n^2)$.
   True **False Sol.** True

2. **[9 points]** Small problems...

   (a) **[3 points]** What is the main advantage of hash tables over direct-address tables? Please make your answer concise. **Sol.**

   It wastes too much memory. Or it's not memory-efficient. (Typically, the universe of possible keys is super large compared to the set of actual keys, so lots of space is wasted.)

   (b) **[3 points]** We have tuples of dates of the form $\langle xxxx(year), xx(month), xx(date)\rangle$. Which sorting algorithm would you use to sort them? **Sol.**

   Radix sort.

   (c) **[3 points]** If we sort n elements using randomized quicksort, what is the probability that the maximum element and the minimum element are compared? Explain why. **Sol.** $\frac{2}{n}$. Because it happens if and only if the maximum or the minimum element is chosen as the first pivot.

3. **[8 points]**

   (a) **[5 points]** You're given an array $A[1 \cdots 8] = \langle 2, 5, 8, 1, 4, 7, 3 \rangle$. Run Max-heapify on the root. What is $A[1 \cdots 8]$? **Sol.** 8, 5, 7, 1, 4, 2, 3

   (b) **[3 points]** Suppose we're given the following three vectors: $(6, 4), (3, 5), (1, 4)$. Stable-sort them in increasing order of the *second* coordinate values. **Sol.** (6, 4), (1, 4), (3, 5)

4. **[10 points]** Describe the Heapsort algorithm. You can use a pseudo-code, plain English, or a mixture of both. Assume that the input is $A[1 \dots n]$ and you want to sort them in non-decreasing order. If you want to use a pseudo-code, you can use Build-Max-Heap(A), $A.length$, A.heap-size, Max-Heapify(A, i).[1] But feel free to use your own notation.

[1] Recall that this function make the subtree rooted at node i a max-heap assuming that both subtrees rooted at i's children are max-heaps.

5. **[10 points]** The following is a *variant* of the $O(n)$ time deterministic algorithm for the Selection problem you learned in class. The algorithm consists of the following five steps. To analyze the algorithm's running time, state what is the running time of *each* step and state the *recurrence* for the running time. For example, the first step's run time is $O(n)$. Use $T(n)$ to denote the algorithm's running time on inputs of size n.

   (a) Divide the n elements into groups of size 7 (so we will have $n/7$ groups). Time: $O(n)$.

   (b) **[2 points]** Find the median of each group (more precisely, the aggregate running time over all groups). Time: **Sol.** $O(n)$.

   (c) **[2 points]** Find the median x of the $n/7$ medians by a recursive call to Select. Time: **Sol.** $T(n/7)$.

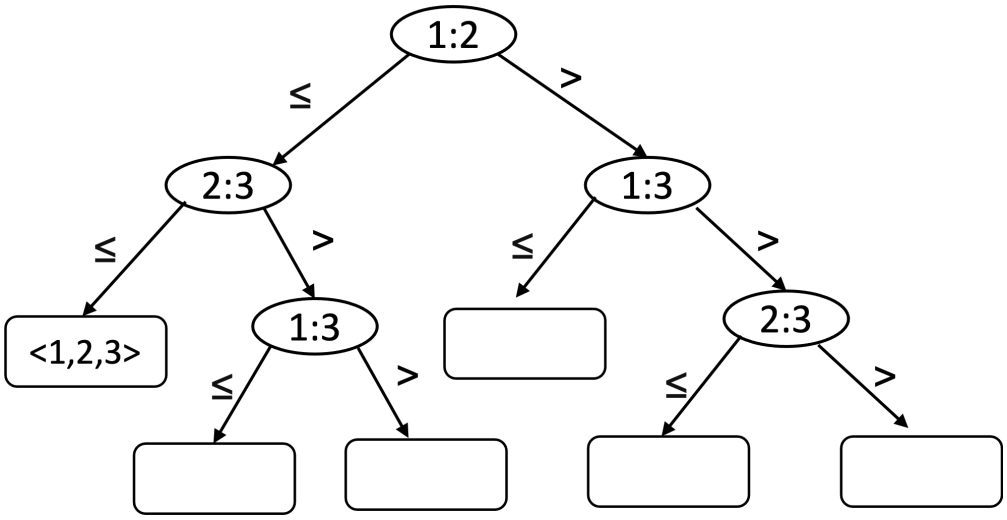
   (d) **[2 points]** Call Partition with x as the pivot. Time: **Sol.** $O(n)$.

   (e) **[3 points]** Make a recursive call to Select either on the smaller elements or larger elements (depending on where the element we are looking for lies); if the pivot is the element we are done. Time: **Sol.** $T(\frac{2}{7}n)$.

   [1 points] Recurrence for the running time $T(n) =$

   Sol. $T(n) = T(\frac{1}{7}n) + T(\frac{2}{7}n) + O(n)$.

6. **[10 points]** The following is the decision tree of Insertion sort on three elements ($n = 3$). In the following, $1 : 2$ is a short hand for comparing a_1 and a_2, assuming that the input is $\langle a_1, a_2, a_3 \rangle$. Note that $\langle 1, 2, 3 \rangle$ implies that we have $\langle a_1, a_2, a_3 \rangle$ after sorting. Fill out all the blanks.



7. **[6 points]** Consider a hash table with $m = 10$ slots. Suppose we use the hash function $h(k) = k \bmod 10$. Say we insert (elements of) keys $k = 25, 8, 6, 4, 15, 16, 5$ in this order. Show the final table when chaining is used to resolve collisions. Insert the element at the *beginning* of the linked list.

   Sol. slot 4 has a linked list storing 4.
   slot 5 has a linked list storing 5, 15, 25 in this order.
   slot 6 has a linked list storing 16, 6 in this order.
   slot 7 has a linked list storing 8.
   All other slots have NIL.

   Any ordering of elements with the same hash value is acceptable. Rubric: For each key value assigned to a wrong slot, -2.

8. **[10 points]** Universal Hash Family. Here, we're considering the following scenario. The universe U of all possible key values is super large, compared to n, the number of actual keys we have. Recall that t denotes the hash table size, i.e., the number of slots.

   (a) **[5 points]** If you choose a hash function from all possible functions from U to $\{0, 1, ..., t-1\}$, what goes wrong? **Sol.** See lecture slides.

   (b) **[5 points]** Suppose $\mathcal{H} = \{k \mod t, k+1 \mod t, k+2 \mod t, \ldots, k+t-1 \mod t\}$. Is this a universal hash family? **Sol.** No. For any function h chosen from $\mathcal{H}$, we have $h(0) = h(t)$. So, 0 and t collide with probability 1.

9. **[10 points]** Randomized-Select implementation. The following is the pseudo-code of Randomized-Select(A, p, r, i) from the textbook where missing parts are underlined. Recall that

   - Randomized-Select(A, p, r, i) is supposed to return the ith smallest element in the sub-array $A[p...r]$.
   - Randomized-Partition(A, p, r) randomly chooses a pivot from $A[p...r]$ and returns the pivot index q such that all elements in $A[p...q-1]$ are smaller than $A[q]$ and all elements in $A[q+1...r]$ are greater than $A[q]$.

```
Partition(A, p, r)
1. x = A[r]
2. i =  p - 1
3. for j = p to r-1
4.    if A[j] <= x
5.       i = i+1
6.       exchange A[i] with A[j]
7. exchange A[i+1] with A[r]
8. return i +1
```

Complete the following pseudocode. You can assume that all elements have distinct values.

```
Randomized-Select(A, p, r, i)
1. if p == r
2.   return _____________________________________________________
3. q = Randomized-Partition(A, p, r)
4. k = q - p + 1
5. if i == k
6.   return _____________________________________________________
7. elseif i < k
8.    return _____________________________________________________

9. else return _____________________________________________________
```

Sol. See CLRS. 2, 2, 3, 3 pts for the four blanks, in this order.

10. **[10 points]** We are given two arrays $A[1 \dots m]$ and $B[1 \dots n]$. We want to test if the set of numbers in $B[1 \dots n]$ is a subset of numbers found in $A[1 \dots m]$. For simplicity, assume that all numbers in each array are distinct. Give an algorithm that has a running time $O(m + n)$ in expectation. **Sol.** Insert all numbers in A into a hash table. Then, we just need to search each element in B in the hash table. Since each operation takes $O(1)$ time in expectation, all take $O(m + n)$ in expectation.

11. **[15 points]** Suppose we have a sorted array of n integers of distinct values, $A[1 \ldots n]$. Suppose we would like to find a target integer t—if it appears in the array. Prove that any comparison based algorithm requires $\Omega(\log n)$ time *using a decision tree based argument*. Here, by comparison based algorithm, we mean that the outcome of the algorithm is completely determined by a sequence of comparisons of t to a number in A. In each comparison, you can only know $<, >$, or $=$.

    (Note: This is a bonus problem. This problem is for students who challenged themselves hard, so they could solve new problems they have never seen before. So, this problem will be graded rigorously.)