CSE 100: Algorithm Design and Analysis: Midterm Exam 2

Spring 2024

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 103. Any points you earn above 100 will be considered as bonus.

Problem	Points earned	Out of
1		22
2		9
3		6
4		6
5		10
6		10
7		10
8		10
9		10
10		10
Sum		Max 103

Name _____

1. **[22 points]** For each of the following claims, determine if it is true or false. *No* explanation is needed.

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

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

   (c) **[2 points]** If we solve $T(n) = T(\frac{9}{10}n) + T(\frac{1}{10}n) + \Theta(n)$, then we obtain $T(n) = \Theta(n)$.
   True **False Sol.** False

   (d) **[2 points]** One can sort n integers in the range between 0 and n^{100} in $O(n)$ time, assuming that each binary operation takes $O(1)$ time.
   True **False Sol.** True

   (e) **[2 points]** The *expected* running time of the randomized quicksort algorithm (that chooses the pivot uniformly at random) is $O(n \log n)$ for all inputs.
   True **False Sol.** True

   (f) **[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, where α is the load factor.
   True **False Sol.** True

   (g) **[2 points]** In Max-Priority Queue, we can *extract* the max element in $O(1)$ time.
   True **False Sol.** False

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

   (i) **[2 points]** Consider the decision tree of Insertion-sort on n elements. Then, the tree has a path from the root to a leaf node that has $\Omega(n^2)$ edges.
   True **False Sol.** True

   (j) **[2 points]** We're given a *sorted* array $A[1 \cdots n]$ and a target number x. We want to find i such that $A[i] = x$ if such i exists. We can do this in $O(1)$ time.
   True **False Sol.** False

   (k) **[2 points]** A red black tree consisting of n nodes has a height of $\Theta(\log n)$.
   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]** We can use Bucket-sort to sort n fractional numbers ranging between 0 and 1. Under a certain assumption, it only takes $O(n)$ time in expectation. State the assumption. **Sol.** The assumption that each number is sampled uniformly at random from $[0, 1)$ (independent of other numbers).

3. **[6 points]**

   (a) **[3 points]** You have implemented a max-priority queue using a binary heap and you currently have $A[1 \cdots 8] = \langle 8, 4, 6, 2, 1, 3, 5 \rangle$. Suppose you extracted the max. Then, the priority queue will convert the heap into a max-heap again. What is the array you obtain as a result? You must use the same algorithm you learned from the textbook or lecture notes.

   Sol. 6, 4, 5, 2, 1, 3

   (b) **[3 points]** Suppose we're given the following four vectors: $(6, 2), (3, 2), (1, 1), (3, 1)$. Stable-sort them in increasing order of the *second* coordinate values. **Sol.** (1,1), (3, 1), (6, 2), (3, 2)

4. **[6 points]** Explain how you can implement Build-Max-Heap(A). You can use a pseudo-code, plain English, or a mixture of both. You're allowed to use Max-Heapify(A, i). [1] You can use the notations A.length and A.heap-size.

[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 11 (so we will have $n/11$ 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 $n/11$ medians by a recursive call to Select. Time: **Sol.** $T(n/11)$.

   (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 than the pivot (depending on where the element we are looking for lies); if the pivot is the element we are done. Time: **Sol.** $T(\frac{8}{11}n)$.

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

   Sol. $T(n) = T(\frac{1}{11}n) + T(\frac{8}{11}n) + O(n)$.

6. **[10 points]** Prove that the randomized quicksort has an expected run time of $O(n \log n)$ for any inputs. For simplicity, you can assume that the input is a permutation of $1, 2, \ldots, n$ and can use the fact that $1 + \frac{1}{2} + \ldots + \frac{1}{n} = O(\log n)$.

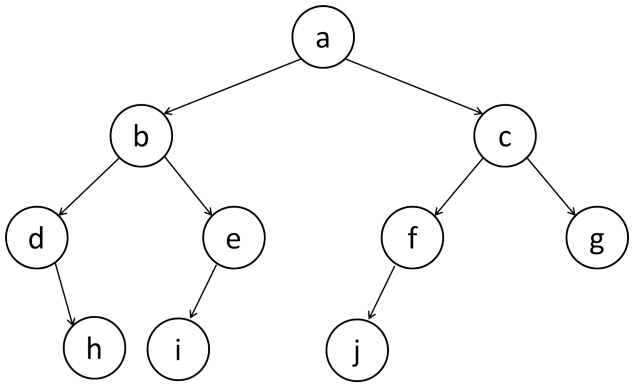
   Sol. See the lecture slides.

7. **[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 $\mathcal{H} = \{k \mod t, 2k \mod t, 3k \mod t, \ldots, tk \mod t\}$. 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.

8. **[10 points]** Consider the following binary search tree where nodes are labeled by alphabets. Here the keys are *not* shown in the picture. Assume that all key values are distinct.



(a) **[2 points]** Do the inorder tree walk printing the labels of nodes. **Sol.** d h b i e a j f c g

(b) **[1 points]** What is the node with the maximum key? **Sol.** g

(c) **[1 points]** What is the node with the minimum key? **Sol.** d

(d) **[2 points]** What is f's successor? **Sol.** c

(e) **[2 points]** What is a's predecessor? **Sol.** e

(f) **[2 points]** What is the node with the second smallest key value? **Sol.** h

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]** You are given two sequences, $\langle a_1, a_2, ..., a_n \rangle$ and $\langle b_1, b_2, ..., b_n \rangle$, where each sequence consists of distinct integers. Describe a linear time algorithm (in the average case) that tests if a sequence is a permutation of the other. Assume that the simple uniform hashing assumption holds.

    Sol. We create a hash table of size $\Theta(n)$ and use chaining to resolve collisions. Recall that under the simple uniform hashing assumption, a search, either successful or unsuccessful, take $O(1)$ time on average. We first insert $a_1, a_2, ..., a_n$ into the hash table. This is done in $O(n)$ time in the average case. Then, we search each b_i in the hash table, which is done in $O(1)$ time. we can find every b_i in the hash table if and only if one is a permutation of the other, due to the fact that each of the two sequences consists of distinct integers. No need to explain the running time.