CSE 100: Algorithm Design and Analysis
Midterm Exam 2

Spring 2018

Note: Please write down your name on *every* page. 10 pages in total including this cover. Time: 4:30-5:45pm

Problem	Points earned	Out of
1		14
2		6
3		5
4		10
5		10
6		10
7		10
8		10
9		10
10		10
11		10
Sum		Max 105

Name _____

1. **[14 points]** For each of the following claims, determine if it is true or false. *No* explanation is needed.

   (a) **[2 points]** Insertion sort is a stable sorting algorithm (here, we are considering the pseudocode of Insertion sort in the textbook).
   True **False Sol.** True

   (b) **[2 points]** The average running time of the randomized quicksort algorithm (that chooses the pivot randomly) is $O(n \log n)$ for all inputs.
   True **False Sol.** True

   (c) **[2 points]** Quick sort is an in-place sorting algorithm (here, we are considering the pseudocode of Quick sort in the textbook).
   True **False Sol.** True

   (d) **[2 points]** Suppose we resolve hash table collisions using chaining. Under the simple uniform hashing assumption, a successful search takes $\Theta(1/(1-\alpha))$ time in expectation where α is the load factor of the current hash table.
   True **False Sol.** False

   (e) **[2 points]** We are given a array $A[1 \cdots n]$ (not necessarily sorted) along with a key value v. We can find $1 \leq i \leq n$ such that $A[i] = v$ in $O(\log n)$ time (if such i exists).
   True **False Sol.** False

   (f) **[2 points]** $A = \langle 1, 2, 3, ..., n \rangle$ is a min-heap.
   True **False Sol.** True

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

2. **[6 points]** Explain...

   (a) **[3 points]** What is the main advantage of hash tables over direct-address tables? Please make your answer concise. **Sol.** Space-efficient; this answer is enough. More precisely, if the universe of possible keys is super large compared to the set of actual keys, then lots of space is wasted.

   (b) **[3 points]** Why the hash function $h(k) = k \bmod 2^L$ for some integer L is not desirable? Please make your answer concise. **Sol.** Only L least significant bits are used in computing the hash value.

3. **[5 points]** Show that $T(n) = T(\frac{9}{10}n) + n$ implies $T(n) = \Theta(n)$ (Hint: Iteratively expand the recurrence).

4. The following is a description 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, we analyzed the asymptotic running time of each step, set up a recurrence for the running time, and solved it. **State the running time of each step.** No explanation is needed. Use $T(n)$ to denote the algorithm's running time on inputs of size n. You can use the fact that $\frac{3}{10}n \leq x$'s order $\leq \frac{7}{10}n$.

   (a) Divide the n elements into groups of size 5 (so we will have $n/5$ groups). Time: **Sol.** $\Theta(n)$.

   (b) Find the median of each group (more precisely, the aggregate running time over all groups). Time: **Sol.** $\Theta(n)$.

   (c) Find the median x of the $n/5$ medians by a recursive call to Select. Time: **Sol.** $T(n/5)$.

   (d) Call Partition with x as the pivot. Time: **Sol.** $\Theta(n)$.

   (e) 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.** $T(\frac{7}{10}n)$.

   Sol. You can also use $O()$ instead of $\Theta()$.

5. **[10 points]** We learned the theorem that any comparison based sorting algorithm has a running time of $\Omega(n \log n)$. Your job is to **complete the following proof of the theorem.**

   (a) Consider any comparison-based algorithm and its decision tree T on n elements.

   (b) **[4 points]** Briefly explain why the decision tree has at least $n!$ leaves. **Sol.** There are $n!$ permutations of n elements and each permutation must appear at least once at a leaf node.

   (c) The tree's height h is defined as the maximum number of edges on any path from the root to a leaf. It is known that the number of leaves in the tree of height h is at most 2^h.

   (d) Therefore, we have $n! \leq 2^h$.

   (e) **[4 points]** Show that $h = \Omega(n \log n)$. **Sol.** See the lecture slides. We show that $(n/2)^{n/2} \leq n! \leq 2^h$. Taking the log on both sides gives the desired conclusion.

   (f) **[2 points]** Explain why $h = \Omega(n \log n)$ implies the sorting algorithm's running time is $\Omega(n \log n)$. **Sol.** In other words, there is a path of length h from the root to a leaf, which corresponds to a permutation. So, when the permutation is the right answer, the algorithm makes h comparisons, meaning that the running time is $\Omega(h)$.

6. **[10 points]** Consider a hash table with $m = 10$ slots and using the hash function $h(k) = k \bmod 10$. Say we insert (elements of) keys $k = 2, 5, 15, 12, 22, 32$ in this order. Show the final table in the following two cases. In both cases the same hash function $h(k)$ is used.

   (a) **[5 points]** When chaining is used to resolve collisions. Insert the element at the *beginning* of the linked list.

   Sol. slot 2 has a linked list storing 32, 22, 12, 2 in this order.
   slot 5 has a linked list storing 15, 5, in this order.
   All other slots have NIL.

   Any ordering of elements with the same hash value is acceptable. Rubric: if you has an integer to a wrong slot, -1.

   (b) **[5 points]** When open addressing and linear probing are used to resolve collisions, i.e. $h(k, i) = k + i \bmod 10$.

   Sol. 0: NIL
   1: NIL
   2: 2
   3: 12
   4: 22
   5: 5
   6: 15
   7: 32
   8: NIL
   9: NIL

   Rubric: for each integer in a wrong position, -1.

7. [**10 points**] Quicksort implementation. Give a pseudo-code of Quicksort(A, p, r) that sorts $A[p...r]$ via quick-sort. You can use the following helper function, Partition(A, p, r),

```
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
```

8. **[10 points]** Heap implementation. As in the textbook, we can represent a binary max-heap using array $A[1 \cdots A.heapsize]$. **Give a pseudo-code of Heap-Extract-Max(A) that removes the max element from $A[1 \cdots A.heapsize]$ and returns it.** You can use Max-Heapify(A, i) (Max-Heapify on node i). Note that

   - You can assume that $A.heap\text{-}size \geq 1$; and
   - The heap must remain a max-heap after executing this operation.

   Sol. See CLRS.

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 $A[q]$.

   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]** Show how to sort n integers in the range from 0 to $n^3 - 1$ in $O(n)$ time. Here you can assume that each bit operation takes $O(1)$ time, and so does comparing two integers. Explain your algorithm's running time. **Sol.** Express each given integer as an n-ary number, so it has 3 digits where each digit can have a value between 0 and $n - 1$. We apply Radix sort. The running time is $\Theta(3 * (n + n)) = \Theta(n)$.

11. **[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.