Assignment #4

Due: Friday Feb 18, 2022 (11:59 pm)

(Total 40 marks)

CMPUT 204
Department of Computing Science
University of Alberta

A number of questions in this assignment require you to study some problems and their solutions given in Problem Set #3. If a problem is related to some exercises of CLRS, possible solutions may also be found online. You may consult, adopt, or revise any of these solutions for your answers. When asked to show how an algorithm works, make sure you indicate clearly which algorithm you are applying. If you apply an algorithm different from what is given in Problem Set #3, please specify.

Problem 1. (16 marks)

a. (2 marks) Consider the following program:

```
Smallest (A, f, l)
  **Precondition: A[f..l] is an array of integers, f ≤ l and f, l ∈ ℕ.
  **Postcondition: Returns the smallest element of A[f..l].
  if f = l then
    return A[f]
  else
    m = ⌊(f+l)/2⌋
    return min(Smallest(A, f, m), Smallest(A, m + 1, l))
  end if
```

Write a recurrence relation that describes the time complexity of this program in terms of the size of array A and solve it.

b. (2 marks) Show the result after each operation below, starting on an NIL BST: `Insert(16)`, `Insert(4)`, `Insert(3)`, `Insert(9)`, `Insert(1)`, `Insert(44)`, `Insert(29)`, `Delete(1)`, `Delete(4)`, `Insert(34)`.

c. (2 marks) First, draw an AVL tree of height 4 that contains the minimum number of nodes. Then, delete a leaf node (any leaf node) from your AVL tree above so that the resulting tree violates the AVL-property. Then, apply tree-rotation to turn it to an AVL tree.

d. (2 marks) Demonstrate what happens when the following keys are inserted to a hash table with collisions resolved by chaining: 18, 22, 32, 8, 6, 10, 20, 16, 2, 29. Let the table have 9 slots and the hash function be $h(k) = k \mod 9$.

e. (2 marks) CLRS Exercise 6.5-9 (which can also be found in Problem Set #3) asks you to give an $O(n \lg k)$-time algorithm to merge k sorted lists into one sorted list, where n is the total number of elements in all the input lists. Use the following example to explain how your algorithm works for three sorted lists: L1 = [2,5], L2 = [4,8], and L3 = [3,7].

f. (3 marks) Consider Problem 7 of Problem Set #3

- Suppose the root of a tree T with 7 nodes is nearly balanced (note that the subtrees rooted at other nodes of the tree may not be nearly balanced). What is the maximum height of such a T?
- To calculate the maximum possible height of a nearly balanced tree with n nodes, we start from

$$H_{max}(n) = 1 + H_{max}(\tfrac{2}{3}(n-1))$$

Use $n = 8$ as an example to explain how this equation helps derive the maximum height of a nearly balanced tree. To support your answer, draw a nearly balanced tree that consists of 8 nodes. (Hint: The above equation can be viewed as a recursive definition, with some base cases added.)

- If a BST is nearly balanced, what would be the running time for searching an element in it? Briefly justify your answer.

g. (3 marks) Consider Problem 9 of Problem Set #3. Explain, using n = 10, how your solution works. In particular, explain how you identify which bottle is poisoned.

Problem 2. (8 marks)

A d-ary heap is like a binary heap except that non-leaf nodes have d children instead of 2 children (with one possible exception for the parent of the last leaf). Also see CLRS Problem 6.1, page 167. In this question, we consider 3-ary max-heaps.

a. Given an array A and index i, how do you compute its left child, middle child, and right child, and how do you compute the parent of $A[i]$?

b. What is the height of a 3-ary heap of n elements? Justify your answer briefly.

c. Write a version of Max-Heapify for a 3-ary max-heap and analyze its running time.

d. Write a version of Build-Heap for a 3-ary max-heap and analyze its running time.

e. Let array $A = [2, 8, 5, 4, 7, 10, 12, 6]$. Invoke your Build-Heap(A) to build a 3-ary max-heap. Just show the resulting array.

Problem 3. (8 marks) Given a max-heap A containing n keys and two element x and y such that $x < y$. Propose an efficient algorithm for reporting (say printing) all the keys z in A satisfying $x \leq z \leq y$ (note that x and y may not necessarily be in A). The running time of your algorithm should be $O(k)$ where k is the number of elements that are greater than or equal to x and less than or equal to y.

Give your algorithm in pseudocode and analyse its running time. In particular, comment on why your algorithm is more efficient than a linear search over A: check each element a in A to see whether it is the case that $x \leq a \leq y$.

Problem 4. (8 marks) We have a set J of n jars and a set L of n lids such that each lid in L has a unique matching jar in J. Unfortunately all the jars look the same (and all the lids look the same). We can only try fitting a lid ℓ to a jar j for any choice of $\ell \in L$ and $j \in J$; the outcome of such an experiment is either a) the lid is smaller, b) the lid is larger, or c) the lid fits the jar. We cannot compare two lids or two jars directly. Describe an algorithm to match all the lids and jars. The expected number of tries (experiments) of fitting a lid to a jar performed by your algorithm must be $O(n \log n)$.

We describe an algorithm and then you will be asked to answer some questions.

The idea of the algorithm is very similar to that of Quicksort.

1. If $n = 1$ (i.e. only one jar and one lid) then they must be matching; return this as the solution.
2. If $n > 1$ we do the following:
   (a) Take anyone of the lids, say $L[\ell]$ as the pivot (for example take one of them randomly)

(b) Compare this lid against all the jars (using n comparison) to partition the jars into three sets: 1) J_s are all those jars that are smaller than $L[\ell]$, 2) J_l are all those jars that are larger than $L[\ell]$, 3) the 3rd set contains the unique jar, say j, that fits into $L[\ell]$.

(c) Use jar j (that is matching lid ℓ) and compare against all the lids in L (except ℓ) to partition L into a set of smallers, denoted by L_s, and those that are largers, denoted by L_l; this takes $n-1$ comparisons.

(d) Note J_s are all the jars smaller than ℓ and L_s are all the lids smaller than j, and therefore the lids of jars in J_s are exactly those in L_s; similarly the lids of jars in J_l are exactly those in J_l. Solve the following two subproblems recursively: jars and lids in J_s and L_s as one subproblem, and jars and lids in J_l and L_l as another subproblem.

Here are the questions that you are asked to answer.

- Provide a running time analysis.
- Suppose it is required that your algorithm always select the last lid as the pivot in the given line up. Consider a problem with 7 lids $l_1, \ldots, l_7$ and 7 jars $j_1, \ldots, j_7$, in the increasing order of their sizes, where all jars have distinct sizes and l_i uniquely fits j_i for all $1 \leq i \leq 7$. Given the sequence of lids, $\langle l_5, l_1, l_6, l_7, l_2, l_3, l_4 \rangle$, and an arbitrary sequence of jars, explain how your algorithm works. In your answer you can assume that any subproblem of the given problem is solved correctly. That is, you don't need to show how subproblems are solved, just give the result.
- Suppose your algorithm must take the last lid as the pivot in the given line up. What would be the worst-case running time of your algorithm and when it is guaranteed to happen? What is the best-case running time? You do not need to present your algorithm, but it must be an efficient one under the requirement.