Computer Science CSC263H — January 9, 2020
St. George Campus — University of Toronto

Homework Assignment #1

Due: January 16, 2020, by 5:30 pm

- **You must submit your assignment through the Crowdmark system.** You will receive by email an invitation through which you can submit your work. If you havent used Crowdmark before, give yourself plenty of time to figure it out!
- You must submit a **separate** PDF document with for **each** question of the assignment.
- To work with one or two partners, you and your partner(s) must form a **group** on Crowdmark (one submission only per group). We allow groups of up to three students, submissions by groups of more than three students will not be graded.
- The PDF file that you submit for each question must be typeset (**not** handwritten) and clearly legible. To this end, we encourage you to learn and use the LaTeX typesetting system, which is designed to produce high-quality documents that contain mathematical notation. You can use other typesetting systems if you prefer, but handwritten documents are not accepted.
- If this assignment is submitted by a group of two or three students, for each assignment question the PDF file that you submit should contain:
  1. The name(s) of the student(s) who *wrote* the solution to this question, and
  2. The name(s) of the student(s) who *read* this solution to verify its clarity and correctness.
- By virtue of submitting this assignment you (and your partners, if you have any) acknowledge that you are aware of the homework collaboration policy that is stated in the csc263 course web page: `http://www.cs.toronto.edu/~sam/teaching/263/#HomeworkCollaboration`.
- For any question, you may use data structures and algorithms previously described in class, or in prerequisites of this course, without describing them. You may also use any result that we covered in class, or is in the assigned sections of the official course textbook, by referring to it.
- Unless we explicitly state otherwise, you should justify your answers. Your paper will be marked based on the correctness and completeness of your answers, and the clarity, precision, and conciseness of your presentation.
- The total length of your pdf submission should be no more than 3.5 pages long in a 10pt font.

Question 1. (1 marks) The following procedure has an input array $A[1..n]$ with $n \geq 2$ arbitrary integers.

In the pseudo-code, "**return**" means ***immediately exit the procedure and then halt.*** Note that the indices of array A starts at 1.

```
NOTHING(A)
n = A.size
for i = 1 to n                    // i = 1, 2, ..., n (including n)
    for j = 1 to n                // j = 1, 2, ..., n (including n)
        if A[n - j + 1] ≠ j then return
    if (A[i] ≠ n - i) or (A[1] + A[2] = 2n - 3) then return
return
```

Assume that each assignment, comparison and arithmetic operation takes ***constant time***.

Let $T(n)$ be the worst-case time complexity of calling NOTHING(A) on an array A of size $n \geq 2$.

Give a tight asymptotic bound for $T(n)$, that is give a function $f(n)$ such that $T(n)$ is $\Theta(f(n))$.

Justify your answer by explaining why it is $O(f(n))$, and why it is $\Omega(f(n))$. Any answer without a sound and clear justification may receive no credit.

Question 2. (1 marks) Let I_n be the set of n integers $\{1, 2, \ldots, n\}$ where n is some power of 2.

Note that we can easily use an n-bit vector (i.e., an array of n bits) $B[1..n]$ to maintain a subset S of I_n and perform the following three operations (where j is any integer in I_n) in constant time each:

INSERT(j): insert integer j into S.

DELETE(j): delete integer j from S.

MEMBER(j): return **true** if $j \in S$, otherwise return **false**.

Describe a data structure that supports all the above operations ***and*** also the following operation

MAXIMUM: return the greatest integer in S such that:

- The worst-case time complexity of operations INSERT(j), DELETE(j), and MAXIMUM is $O(\log n)$ each. The worst-case time complexity of MEMBER(j) is $O(1)$.
- The data structure uses only $O(n)$ bits of storage.

  Note that the binary representation of an integer i where $1 \leq i \leq n$ takes $\Theta(\log n)$ bits. Assume that any pointer also takes $\Theta(\log n)$ bits.

A solution that does not meet ***all*** the above requirements may not get any credit.

HINT: Combine an n-bit vector with a complete binary tree, and avoid using pointers.

a. Describe your data structure by drawing it for $n = 8$ and $S = \{1, 2, 6\}$, and by explaining this drawing briefly and clearly. Your drawing must be very clear.

b. Explain how the operations INSERT(j), DELETE(j), and MAXIMUM are executed, and why they take $O(\log n)$ time in the worst-case. Be brief and precise.

c. Explain how the operation MEMBER(j) is executed, and why it takes $O(1)$ time in the worst-case. Be brief and precise.

[***The questions below will not be corrected/graded. They are given here as interesting problems that use material that you learned in class.***]

Question 3. (0 marks)

Design an efficient algorithm for the following problem. The algorithm is given an integer $m \geq 1$, and then a (possibly infinite) sequence of distinct integer keys are input to the algorithm, ***one at a time***. A *print* operation can occur at any point between keys in the input sequence. When a *print* occurs, the algorithm must print (in any order) the m smallest keys among all the keys that were input before the *print*.

For example, suppose $m = 3$, and the keys and *print* operations occur in the following order:

$$20, 15, 31, 6, 13, 24, \textit{print}, 10, 17, 9, 16, 5, 11, \textit{print}, 14, \ldots$$

Then the first *print* should print 15, 6, 13 (in any order), and the second *print* should print 6, 9, 5 (in any order).

Assume that at least m keys are input before the first *print* occurs and that m does not change during an execution of the algorithm.

Describe a ***simple*** algorithm that solves the above problem with the following worst-case time complexity:

- $O(\log m)$ to process each input key.
- $O(m)$ to perform each *print* operation.

Your algorithm must use a data structure that we learned in class.

- State which data structure you are using and describe the items that it contains.
- Explain your algorithm for each operation ***clearly*** and ***concisely***, in English.
- Explain why your algorithm achieves the required worst-case time complexity described above.
- Prove that your algorithm is correct (Hint: use induction. What is your induction hypothesis?)

Question 4. (0 marks) In class we studied *binary* heaps, i.e., heaps that store the elements in complete *binary* trees. This question is about *ternary* heaps, i.e., heaps that store the elements in complete *ternary* trees (where each node has at most *three* children, every level is full except for the bottom level, and all the nodes at the bottom level are as far to the left as possible). Here we focus on MAX heaps, where the priority of each node in the ternary tree is greater or equal to the priority of its children (if any).

a. Explain how to implement a ternary heap as an array A with an associated $Heapsize$ variable (assume that the first index of the array A is 1). Specifically, explain how to map each element of the tree into the array, and how to go from a node to its parent and to each of its children (if any).

b. Suppose that the heap contains n elements.
(1) What elements of array A represent internal nodes of the tree? Justify your answer.
(2) What is the height of the tree? Justify your answer.

c. Consider the following operations on a ternary heap represented as an array A.

- INSERT(A, key): Insert key into A.
- EXTRACT_MAX(A): Remove a key with highest priority from A.

- UPDATE(A, i, key), where $1 \leq i \leq A.Heapsize$: Change the priority of $A[i]$ to *key* and restore the heap ordering property.

- REMOVE(A, i), where $1 \leq i \leq A.Heapsize$: Delete $A[i]$ from the heap.

For each one of these four operations, describe an efficient algorithm to implement the operation, and give the worst-case time complexity of your algorithm for a heap of size n. Describe your algorithm using high-level pseudo-code similar to that used in your textbook, with clear explanations in English. Express the worst-case time complexity of your algorithm in terms of Θ and justify your answer.

Question 5. (0 marks) Let A be an array containing n integers. Section 6.3 of our textbook (CLRS) describes a procedure, called BUILD-MAX-HEAP(A), that transforms array A into a max-heap in $O(n)$ time. That procedure works "bottom-up", using MAX-HEAPIFY repeatedly.
Another way of transforming A into a max-heap is to insert the elements of A into the heap one at a time. Specifically, the algorithm is as follows:

```
BUILD-BY-INSERTS(A)
    A.heapsize := 1
    for i := 2..n do
        MAX-HEAP-INSERT(A, A[i])
```

a. Give an example of an input array A for which the two procedures BUILD-MAX-HEAP and BUILD-BY-INSERTS produce different outputs. Keep your example as small as possible.

b. Let $T(n)$ be the worst-case time complexity of BUILD-BY-INSERTS for an input array A of size n. Prove that $T(n)$ is $\Theta(n \log n)$. (Recall that the worst-case time complexity of BUILD-MAX-HEAP is $O(n)$, and therefore BUILD-MAX-HEAP is more efficient than BUILD-BY-INSERTS.)