

Homework Assignment #1

**Due: January 18, 2018, by 5:30 pm**

- You must submit your assignment as a PDF file of a typed (**not** handwritten) document through the MarkUs system by logging in with your CDF account at:

`markus.teach.cs.toronto.edu/csc263-2018-01`

To work with one or two partners, you and your partner(s) must form a group on MarkUs.

- The PDF file that you submit must be clearly legible. To this end, we encourage you to learn and use the  $\text{\LaTeX}$  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, the PDF file that you submit should contain for each assignment question:
  1. The name of the student 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.
- Your submission should be no more than 4 pages long in a 10pt font.

**Question 1.** (1 marks)

In the following procedure, the input is an array  $A[1..n]$  of arbitrary integers,  $A.size$  is a variable containing the size  $n$  of the array  $A$  (assume that  $A$  contains at least  $n \geq 2$  elements), and “return” means return from the procedure call.

```
0. Procedure strange(A) // A is indexed starting at 1 (not 0).
1.   A[1] := 0
2.   n := A.size
3.   for i = 2 to n do
4.       for j = 1 to i-1 do A[j] := A[j] - 2
5.       if A[i] is not equal to A[i-1] + 1 then return
6.   return
```

Let  $T(n)$  be the worst-case time complexity of executing procedure **strange()** on an array  $A$  of size  $n \geq 2$ . Assume that assignments, comparisons and arithmetic operations, like additions, take a constant amount of time each.

- a. State whether  $T(n)$  is  $O(n^2)$  and justify your answer.
- b. State whether  $T(n)$  is  $\Omega(n^2)$  and justify your answer.

Any answer without a *sound and clear* justification will receive no credit.

**Question 2.** (1 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  $\Theta$  and justify your answer.

[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) 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( $A$ )
   $A.heap-size := 1$ 
  for  $i := 2..n$  do
    MAX-HEAP-INSERT( $A, A[i]$ )
```

- 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.
- 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.)

**Question 4.** (0 marks)

Let  $I_n$  be the set of  $n$  integers  $\{1, 2, \dots, n\}$  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: Complete binary trees can be implemented without using pointers.

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