Computer Science CSC263H — January 18, 2018
St. George Campus — University of Toronto

Homework Assignment #2

Due: February 1, 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:

  `https://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 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 3 pages long in a 10pt font.

Question 1. (1 marks) This question is about the cost of successively inserting k elements into a binomial heap of size n.

a. Prove that a binomial heap with n elements has exactly $n - \alpha(n)$ edges, where $\alpha(n)$ is the number of 1's in the binary representation of n.

b. Consider the worst-case total cost of successively inserting k new elements into a binomial heap H of size $|H| = n$. In this question, we measure the worst-case cost of inserting a new element into H as the maximum number of pairwise comparisons between the keys of the binomial heap that is required to do this insertion. It is clear that for $k = 1$ (i.e., inserting one element) the worst-case cost is $O(\log n)$. Show that when $k > \log n$, the *average* cost of an insertion, i.e., the worst-case total cost of the k successive insertions divided by k, is bounded above by constant.

Hint: Note that the cost of each one of the k consecutive insertions varies — some can be expensive, other are cheaper. Relate the cost of each insertion, i.e., the number of key comparisons that it requires, with the number of extra edges that it forms in H. Then use part (a).

Question 2. (1 marks) Recall that the height of a node u in a binary search tree (BST) T equals the number of edges in the *longest* path from u to a leaf of the subtree rooted at u. (Leaves have height 0.) Define the radius of u to equal the number of edges in the *shortest* path from u to a leaf of the subtree rooted at u. We define a new notion of a balanced BST: a BST T is 2-balanced if, for every node u of T, $\text{height}(u) \leq 2 \cdot \text{radius}(u)$, where $\text{height}(u)$ is the height of u in T, and $\text{radius}(u)$ is the radius of u in T.

Give a linear time algorithm that determines if a BST T is 2-balanced, according to the definition above. The algorithm's input is a pointer to the root of T, where each node u has the following fields: an integer $u.key$, and pointers $u.lchild$ and $u.rchild$ to the left and right child of u, respectively. (If u has no left or right child, the corresponding pointer is set to NIL.) **There is no height or radius information already stored in any node.** The algorithm's output is TRUE if T is 2-balanced, and FALSE otherwise. The worst-case running time of your algorithm ***must be*** $\Theta(n)$ where n is the number of nodes in T. Describe your algorithm by giving its ***pseudo-code***, and explain why its worst-case running time is $\Theta(n)$. You do not need to prove that the algorithm is correct.

[***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)

From CLRS: 12.1-1, 12.2-1, 12.2-4, 12.2-7

Question 4. (0 marks) We want an efficient algorithm for the following problem. The algorithm is given an integer $m \geq 2$, and then a (possibly infinite) sequence of distinct keys are input to the algorithm, ***one at a time***. A *query* operation can occur at any point between any two key inputs in the sequence. When a *query* occurs, the algorithm must return, ***in sorted order***, the m smallest keys among all the keys that were input before the *query*. Assume that at least m keys are input before the first *query* occurs.

For example, suppose $m = 3$, and the key inputs and query operations occur in the following order:

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

Then the first *query* should return 6, 13, 15; the second *query* should return 6, 10, 13; the third *query* should return 5, 6, 9.

Describe a simple algorithm that, for every $m \geq 2$, solves the above problem with the following worst-case time complexity:

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

Your algorithm must use a data structure that we learned in class without any modification to the data structure.

To answer this question, you must:

1. State which data structure you are using, and describe the items that it contains.
2. Explain your algorithm ***clearly*** and ***concisely***, in English.
3. Give the algorithm's ***pseudo-code***, including the code to process an input key k, and the code for *query*.
4. Explain why your algorithm achieves the required worst-case time complexity described above.

Question 5. (0 marks)

The task in this question is to compute the medians of all prefixes of an array. As input we are given the array $A[1\,..\,n]$ of arbitrary integers. Using a heap data structure, design an algorithm that outputs another array $M[1\,..\,n]$, so that $M[i]$ is equal to the median of the numbers in the subarray $A[1\,..\,i]$. Recall that when i is odd, the median of $A[1\,..\,i]$ is the element of rank $(i+1)/2$ in the subarray, and when i is even, the median is the average of the elements with ranks $i/2$ and $i/2+1$. Your algorithm should run in worst-case time $O(n \log n)$.

a. Describe your algorithm in clear and concise English, and also provide the corresponding pseudocode. Argue that your algorithm is correct.

b. Justify why your algorithm runs in time $O(n \log n)$.