Computer Science CSC263H
St. George Campus

March 1, 2018
University of Toronto

Homework Assignment #5

**Due: March 15, 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 5 pages long in a 10pt font.

**Question 1.** (1 marks)

Suppose we have an array $A[1, 2, \dots]$ supporting the following two operations (where $k$ is a *global variable* initially set to 0):

INSERT($x$)
  $k := k + 1$
  $A[k] := x$

OUTPUTANDREDUCE()
  **for** $i = 1$ **to** $k$ **do** print $A[i]$ **endfor**  *(this **for** loop **includes** $k$)*
  $k := \lfloor k/3 \rfloor$

The cost of each operation is *defined* as follows:

- The cost of INSERT($x$) is *exactly* 1.
- The cost of OUTPUTANDREDUCE() is the exact value of the global variable $k$ just before OUTPUTANDREDUCE() is executed (because this operation prints $k$ elements).

Let $T(n)$ be the worst-case total cost of executing any sequence of $n$ of the above operations, starting with $k = 0$. The *amortized* cost per operation is $T(n)/n$.

What is the *best (i.e., smallest) upper bound* for $T(n)/n$ in the sorted list $L$ below?

- $L = \frac{1}{4}, \frac{1}{2}, 1, \frac{4}{3}, \frac{3}{2}, 2, \frac{5}{2}, 3, \frac{7}{2}, 4, \frac{9}{2}, 5, \log_2 n, \sqrt{n}, \frac{n}{\log_2 n}, n.$

***Justify*** your answer (unjustified answers do not get credit).

HINT: Use the accounting method and charge each INSERT the ***smallest*** amount listed in $L$ such that the total amount charged always covers the total cost of operations.

**Question 2.** (1 marks)

We say that a node $u$ of a *directed* graph $G = (V, E)$ is a *supersource*, if there is an edge from $u$ to every other node in $G$ and there is no edge from any node into $u$, i.e., for all $v \neq u \in V$, $(u, v) \in E$ ***and*** for all $v \in V$, $(v, u) \notin E$.

**a.** Suppose the directed graph $G = (V, E)$ is given by its *adjacency matrix* $A$. Give an *efficient* algorithm to determine whether $G$ has a supersource node, and if so, to output it. Your algorithm should minimize the number of accesses to the adjacency matrix $A$ in the worst-case (e.g., reading an arbitrary element $A[i, j]$ of $A$ counts as one access to $A$). Describe your algorithm using pseudo-code and prove it correct (it is convenient to assume that the set of nodes of $G$ is $V = \{1, 2, \dots, n\}$).

**b.** How many times your algorithm accesses the adjacency matrix $A$ of the graph $G$ in the worst-case? Give the *exact* number (i.e., do not use the asymptotic notation) as a function of the number of nodes $n = |V|$ and the number of edges $m = |E|$ of $G$, and justify your answer.

**c.** Solve part (a) and (b) above under the assumption that the directed graph $G = (V, E)$ is given by its *adjacency lists* $L$ (instead of its adjacency matrix $A$).

**Question 3.** (1 marks)

Let $G = (V, E)$ be an undirected connected graph with $n$ nodes and $m$ edges. In this question you are asked to *orient* the graph, i.e. pick one direction for each edge: an edge $(u, v)$ can be directed either from $u$ to $v$, or from $v$ to $u$ (but not both ways). While $G$ is undirected, the oriented graph is directed.

**a.** Give an algorithm that orients $G$ such that in the oriented graph there is at most one node with no edges going into it. The worst-case running time of your algorithm should be $O(m + n)$ when $G$ is given by its adjacency list. Describe your algorithm in clear and precise English, analyze its running time, and justify its correctness. For each edge $(u, v)$ in $E$, your algorithm should output "$u$ to $v$" if the edge is directed from $u$ to $v$, or "$v$ to $u$" if it is directed from $v$ to $u$.

**b.** Prove that if there is a way to orient $G$ such that every vertex of $G$ has least one edge going into it, then $m \geq n$.

**c.** Give an algorithm that orients any connected undirected graph $G$ with $n$ vertices and $m \geq n$ edges such that every vertex of $G$ has least one edge going into it. The worst-case running time of your algorithm should be $O(m + n)$ when $G$ is given by its adjacency list. Describe your algorithm in clear and precise English, analyze its running time, and justify its correctness.

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

**Question 4.** (0 marks) Consider the following data structure for representing a set $I$ of integers. The elements of the set are stored in a doubly linked list of arrays such that: (1) Each element of $I$ occurs in exactly one of the arrays in this list, (2) each array is sorted in increasing order, (3) the number of elements in each array is a power of 2, (4) no two arrays in the list have the same size, (5) the arrays in the linked list are kept in order of increasing size, and (6) the first element of each array also contains a *size* field, which is the number of elements in this array. Note that although each array is sorted, there is no particular relationship between the elements in different arrays.

**a.** Draw two instances of this data structure: one for set $I = \{3, 5, 1, 17, 10\}$ and one for set $I = \{17, 8, 3, 10, 1, 12, 6\}$.

**b.** To do a SEARCH($x$) for an integer $x$ in this data structure, one performs a binary search separately on each array in the list until either $x$ is found in some array, or all arrays have been considered and $x$ is not found.

Give a good upper bound on the worst-case time complexity of this SEARCH algorithm (using the $O$ notation) and justify your answer.

**c.** To do INSERT($x$), i.e, to insert a new integer $x$ into $I$ (assume that $x$ is not already in $I$), one performs following algorithm:

```
(a) create a new array of size 1 containing x
(b) insert this new array at the beginning of the linked list
(c) while the linked list contains 2 arrays of the same size:
        merge the 2 sorted arrays into one sorted array of twice the size.
        To do each merging use a procedure similar to the one used in Mergesort.
```

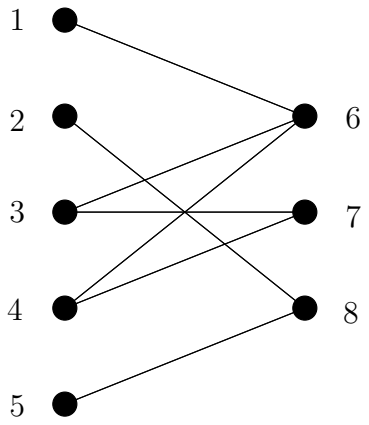
Give a good upper bound on the worst-case time complexity of this INSERT algorithm (using the $O$ notation) and justify your answer.

**d.** Suppose we execute a sequence of $n$ INSERTs starting from an empty set $I$. Determine a good upper bound on the *amortized* time of an INSERT (i.e., the worst-case total time to execute these $n$ INSERTs divided by $n$). Justify your answer in two different ways, i.e., give two separate proofs, each proof using a different argument (e.g., use aggregate analysis and the accounting method).

**e.** Describe an algorithm to perform a DELETE($x$) operation (i.e., given a pointer to integer $x$ in one of the arrays of the list, remove $x$) in $O(n)$ time in the worst case. There is no need to use pseudocode, a clear and concise description of the algorithm in English is sufficient. Explain why the worst-case time complexity of your algorithm is $O(n)$.

**Question 5.** (0 marks) Prove that every connected undirected graph $G$ contains at least one vertex whose removal (along with all incident edges) does *not* disconnect the graph. Give an algorithm that for any connected graph $G = (V, E)$, given by its adjacency list, finds such a vertex in $O(|V| + |E|)$ time in the worst case. Justify your algorithm's correctness and worst-case time complexity.

**Question 6.** (0 marks) Let $G = (V, E)$ be an undirected graph. $G$ is ***bipartite*** if the set of nodes $V$ can be partitioned into two subsets $V_0$ and $V_1$ (i.e., $V_0 \cup V_1 = V$ and $V_0 \cap V_1 = \varnothing$), so that every edge in $E$




connects a node in $V_0$ and a node in $V_1$. For example, the graph shown below is bipartite; this can be seen by taking $V_0$ to be the nodes on the left and $V_1$ to be the nodes on the right.
Note that $G$ is bipartite if and only if every connected component of $G$ is bipartite.

**a.** Prove that if $G$ is bipartite then it has no *simple cycle* of odd length.[1] Hint: Give a proof by contradiction.

**b.** Prove that if $G$ has no simple cycle of odd length (i.e., every simple cycle of $G$ has even length) then $G$ is bipartite. (Hint: Suppose every simple cycle of $G$ has even length. Perform a BFS starting at any node $s$. Assume for now that $G$ is connected, so that the BFS reaches all nodes; you can remove this assumption later on. Use the distance of each node from $s$ to partition the set of nodes into two sets, and prove that no edge connects two nodes placed in the same set.)

**c.** Describe an algorithm that takes as input an undirected graph $G = (V, E)$ in adjacency list form. If $G$ is bipartite, then the algorithm returns a pair of sets of nodes $(V_0, V_1)$ so that $V_0 \cup V_1 = V$, $V_0 \cap V_1 = \varnothing$, and every edge in $E$ connects a node in $V_0$ and a node in $V_1$; if $G$ is not bipartite, then the algorithm returns the string "not bipartite." Your algorithm should run in $O(n + m)$ time, where $n = |V|$ and $m = |E|$. Explain why your algorithm is correct and justify its running time. (Hint: Use parts (a) and (b).)

**Question 7.** (0 marks) You are riding your bike and you have an aerial map of the routes you could take. This map contains the $(x, y)$-coordinates of $n$ bike pump stations. There is a straight bikeway that goes directly between any two bike pump stations. You want to travel from station $s$ to station $t$. Since your tires aren't very good, the best route from $s$ to $t$ is one that minimizes the distance that you have to travel between any two successive stations on this route. Your task is to design an $O(n^2 \log n)$-time algorithm for finding a best route.

**a.** Restate this task as a graph problem. To do so: (i) describe the graph: its vertices, edges, and edge weights, and (ii) restate the task as a graph problem on this graph.

**b.** We learned several algorithms that construct trees from an input graph. One these trees can be used to find the desired path efficiently. Explain how and prove it.

**c.** Describe a corresponding algorithm for solving this problem in plain and clear English.

**d.** Explain why the worst-case running time of your algorithm is $O(n^2 \log n)$.

NOTE: your algorithm can use any algorithms that we studied in the course as "black-boxes," and you can refer to their worst-case time complexity as stated in lecture or in the textbook.

[1] Recall that a cycle is simple if it contains each node at most once.