Computer Science B63
University of Toronto Scarborough

Homework Assignment # 2

Handing in and marking

For this assignment, you need to submit your solutions to the pencil-and-paper exercises on crowdmark and your solutions to the programming question on MarkUs. Your pencil-and-paper solutions will be marked with respect to correctness, clarity, brevity, and readability. Your code will be marked with respect to correctness, efficiency, program design and coding style, clarity, and readability. This assignment counts for 15% of the course grade.

Question 1. Weight-Balanced Trees [20 MARKS]

In this question we prove correctness of the algorithm for rebalancing weight-balanced trees we saw in class. For this question we will consider weight balanced trees with the following balancing equations:

$$\frac{1}{4} \leq \frac{weight(n.left)}{weight(n.right)} \leq 4$$

where,

$$weight(n) = size(n) + 1$$

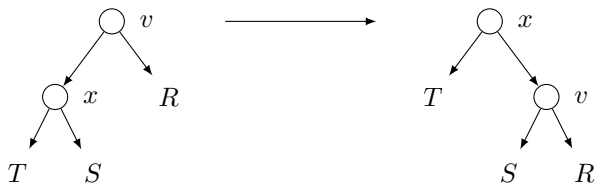
The algorithm works as follows, for each node v on the path from newly inserted/deleted node up to the root:

```
if weight(v.right) > weight(v.left) * 4:
  let x = v.right
  if weight(x.left) < weight(x.right) * (5/3):
    single rotation: left
  else:
    double rotation: right then left
else if weight(v.left) > weight(v.right) * 4:
  let x = v.left
  if weight(x.right) < weight(x.left) * (5/3):
    single rotation: right
  else:
    double rotation: left then right
else:
  no rotation
```

We will focus on the case

```
else if weight(v.left) > weight(v.right) * 4:
  let x = v.left
  if weight(x.right) < weight(x.left) * (5/3):
    single rotation: clockwise
```

Your task is to prove that in this case the single right rotation restores the balance (similar to how we did in the lecture). You do not have to prove all cases of insertion/deletion. Pick any case that you like to show and proof that.

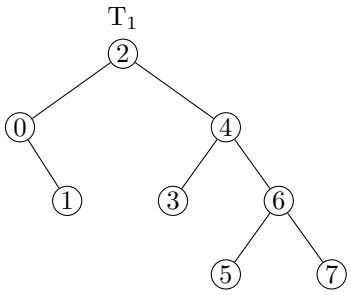


- Begin by clearly stating which case you are proving for (insertion or deletion from which subtree).
- Then state what you know about the original tree, together with their justifications.

- Then state your assumption under which a single right rotation balances the tree. You are essentially proving that this condition is sufficient for a single right rotation to balance the tree.
- State clearly what it means for the rotation to restore the balance. You are trying to prove these hold true for above assumptions.
- Complete the proof!

Question 2. Union of AVL Trees [5 MARKS]

Consider the AVL trees T_1 and T_2 as follows. We discussed an efficient algorithm to find `union` of two AVL trees.



Is `union(T1,T2) =` T_1 ?
Justify?

Question 3. Implementing Heaps [30 MARKS]

We provided you with the starter code for implementing a Minimum Heap. Your task is to implement the functions declared in `minheap.h` that are not already implemented in `minheap.c`. Note that `minheap.c` is the only file you will submit, so make sure your implementation works with the original provided `minheap.h`. Make sure to **carefully study the starter code** before you begin to add your own. The marking scheme for this question is as follows:

- Correctness (includes complexity requirement):
  - `HeapNode getMin(MinHeap* heap)`: 1 mark
  - `HeapNode extractMin(MinHeap* heap)`: 5 marks
  - `bool insert(MinHeap* heap, int priority, int id)`: 8 marks
  - `void changePriority(MinHeap* heap, int nodeIndex, int newPriority)`: 10 marks
- Program design (modular implementation, self-explanatory code, clear logic, no repeated code, no unnecessary code): 4 marks
- Readability and coding style (good use of white space, good naming, etc.): 2 marks
  - You are encouraged (but not required) to use a `linter` to help check and/or auto-format your code.

The runtime complexity requirements for your functions are as follows (here n is the size of the heap):

- `getMax`: $\mathcal{O}(1)$ time.
- `extractMax`: $\mathcal{O}(\log n)$ time.
- `insert`: $\mathcal{O}(\log n)$ time.
- `changePriority`: $\mathcal{O}(\log n)$ time.

We will not be testing `heapify(MinHeap* heap, int nodeIndex)` function separately, but it will be used in the next question so you should still implement it.

Question 4. Heaps [20 MARKS]

There are 2 main methodologies when doing research, top-down and bottom-up. Top-down approach starts with a theoretical hypothesis and researchers then simulate experiments to show validity of the hypothesis. Bottom-up on the other hand starts with some observations and then researchers try to formulate a theoretical hypothesis to justify them. In this question you will get some brief idea about how the bottom-up approach works. You will be running some experiments in this question to get some results. And then try to justify what you observe.

Before you begin working on this question, make sure you atleast have `newHeap`, `insert` and `heapify` working for your heap implementation.

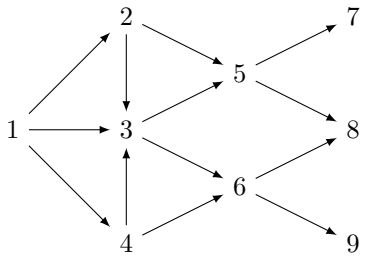
We wanted to create a heap to autotest your heap implementation so I asked your TAs to write a function for it. Both of your TAs gave me a version which you can find in `minheap.c`. Your task is to find out if I should prefer one

over the other. They both take the exact same arguments as input and return a pointer to the minheap. You can assume that both of them are correct implementations. However, we will need to find out if they behave similar with respect to their running time. For this question we will focus on the worst-case analysis of these functions. While you will be writing code for this question, you will not be submitting one. With that in mind, do the following:

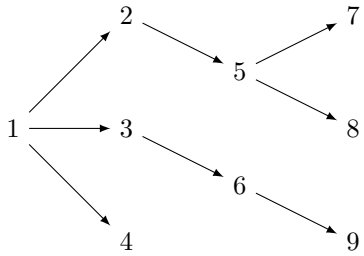
- Identify what is the worst case for both of these functions.
- Write a function (can be in a new file as it's not submitted) that does the following:
  1. Create an input array of values (of a specified size) that will be used as the worst-case input for our `buildHeap` functions.
  2. Run each version of `buildHeap` separately for this input array around 100 times (the more you increase this number, your graph will look smoother) and measure how much time it takes for the function to finish (explore `time.h` to ssee how to measure time of a C function.)
- Run this function for multiple values (at least 10 different values between 10 and 1000000) of size parameter for your input array.
- Based on what values you see, plot a line curve for both versions of `buildHeap` with input size on x-axis and running time on y-axis (just a simple curve made in excel or google sheets is fine).
- What can you say about the worst-case time complexity of these functions based on your observations.
- Finally, prove these bounds theoretically.

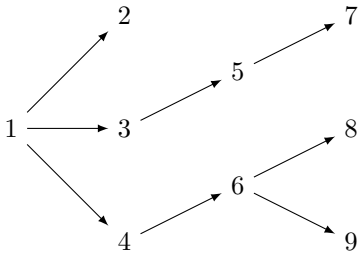
Question 5. Breadth-First Search [15 MARKS]

The breadth-first tree found by breadth-first search depends on how vertices are ordered in the adjaceny lists. Of the following directed graph (call it G):



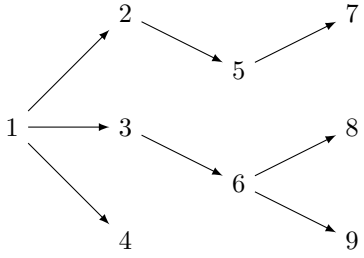
- Both trees below are possible breadth-first trees, both starting at (1).





  Give an adjacency-list representation of G such that breadth-first search starting at (1) results in the left tree, and another representation that results in the right tree.

- Explain why the following **cannot** be a breadth-first tree of G starting at (1).



Question 6. Graph properties [10 MARKS]

Prove or disprove.

- Maximum number of edges in an undirected graph G (no self edges) with n vertices are given by $n(n-1)/2$
- For every undirected graph G (no self edges) with n vertices and $n-1$ edges, G contains no cycle.