Computer Science B63
University of Toronto Scarborough

Homework Assignment # 1

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 10% of the course grade.

Question 1. Asymptotic Bounds [12 MARKS]

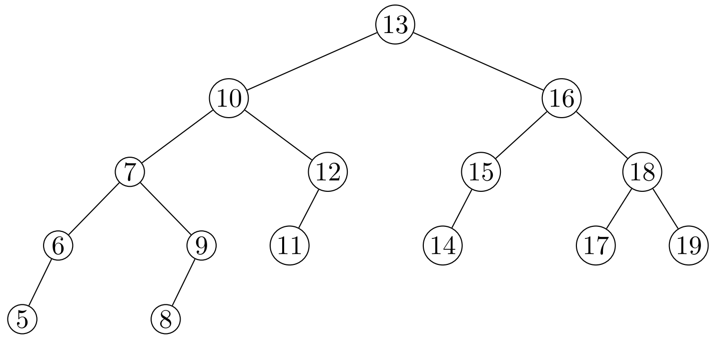
Prove or disprove each of the following *using the definitions of* Big-oh / Big-omega / Big-theta.

- (3 marks) If $f \in \mathcal{O}(h)$ and $g \in \mathcal{O}(h)$ then $f(n) \cdot g(n) \in \mathcal{O}(h)$, for all functions f, g, h in $\mathbb{N} \to \mathbb{R}^+$.
- (3 marks) If $f \in \Theta(g)$ then $g \in \Theta(f)$, for all functions f, g, in $\mathbb{N} \to \mathbb{R}^+$.
- (3 marks) If $f \notin \mathcal{O}(g)$ then $g \in \mathcal{O}(f)$, for all functions f, g in $\mathbb{N} \to \mathbb{R}^+$.
- (3 marks) If $f \in \mathcal{O}(h)$ and $g \in \mathcal{O}(k)$ then $f(n) + g(n) \in \mathcal{O}(max(h(n), k(n)))$, for all functions f, g, h, k in $\mathbb{N} \to \mathbb{R}^+$.

Question 2. AVL Trees Basic Operations [16 MARKS]

Show the AVL trees resulting from performing the requested sequence of operations. Whenever a rotation is required, indicate the type of rotation used and what node it applies to. Whenever a double rotation is required, show the intermediate step (i.e., show the result after each of the single rotations).

- On an initially empty tree, show each step of inserting the keys `29, 30, 32, 34, 23, 54, 39, 57, 40`, in this order.
- On the tree shown below, show each step of deleting the keys `7`, `16`, `17`, `18`, in this order.



Question 3. Implementing Augmented AVL Trees [50 MARKS]

We provided you with the starter code for implementing augmented (with closest-pair) AVL Trees. Your task is to implement the functions declared in `closest_AVL_tree.h` that are not already implemented in `closest_AVL_tree.c`. **Each function must be implemented in either $\mathcal{O}(1)$ or $\mathcal{O}(\log n)$ running time**. Note that `closest_AVL_tree.c` is the only file you will submit, so make sure your implementation works with the original provided `closest_AVL_tree.h`.

Make sure to **carefully study the starter code**, including the file `sample_session.txt`, before you begin to add your own. Making an erroneous assumption can lead to your function(s) earning 0 correctness marks, as your code will be autotested, so please study all requirements stated in the starter code and in this handout carefully.

The marking scheme for this question is as follows:

- Correctness and run time bounds (includes proper memory management):
  - `closest_AVL_Node* search(closest_AVL_Node* node, int key)`: 4 marks
  - `closest_AVL_Node* insert(closest_AVL_Node* node, int key, void* value)`: 8 marks
  - `closest_AVL_Node* delete(closest_AVL_Node* node, int key)`: 18 marks
  - `pair* getClosestPair(closest_AVL_Node* node)`: 8 marks
- Program design (modular implementation, self-explanatory code, clear logic, no repeated code, no unnecessary code): 8 marks
- Readability and coding style (good use of white space, good naming, etc.): 4 marks
  - You are strongly encouraged to use a `linter` to help check and/or auto-format your code. See C-programming resources linked from quercus.

Question 4. Augmenting AVL Trees [22 MARKS]

A teacher wants to keep track of the students that require the most help in their course at any given time. They decide to maintain a list of k students who most recently asked for help. The list does not change unless someone asks for help. When any student asks for help, they use the following rules to update the list:

- If the list has less than k students, add this student to the list.
- If the list already has k students and the student is not in the list then, we remove the student from the list who asked for help furthest in time and add current student in the list.
- If the student is already on the list, the list does not change.

This calls for an ADT consisting of a set H, of at-most k students out of a total of n students in the course that supports the following operations:

- `help(H, x)`: Indicate that student x was recently helped. The student x needs to be added to list H appropriately (based on above rules).
- `drop(H, x)`: Indicate that student x has dropped the course and thus needs to be removed from the list. We want to delete student x from H. This should have no effect if the student was not on the list H to begin with.
- `needsMoreHelp(H, x)`: Find out if student x needs more help in the course at this time. Find out if student x is present in list H. Returns a boolean $True$ or $False$.

For simplicity we consider every student to be associated with a number in $\mathbb{N}$. So, you can assume that the class consists of students from 1 to n. You will pass this student number as the argument to your functions above for x. You will need some way to identify which student asked for help furthest in time.

Your task is to design a data structure to implement this ADT, such that all operations are performed in $\mathcal{O}(\log k)$ time. Provide the following:

1. Describe all information that is to be stored and how it's structured.
2. Provide pseudo-code for the 3 required operations listed above.
3. Justify why your algorithms are correct and why they achieve the required time bound.
4. Assess the space complexity for your data structure.