CPSC 2150 – Algorithms and Data Structures II

Lab5: Binary Tree - naïve sort

Total - 40 Marks

Learning Outcomes

- Design and develop an appropriate binary tree
- Design and develop a naïve sorting algorithm using the binary tree structure
- Analyzing the naïve sort
- Program with C++

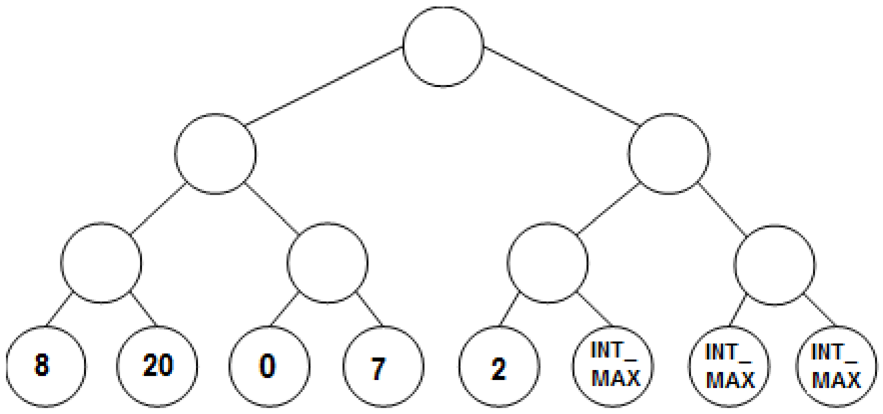
Resources

- Chapter 7 of the text book

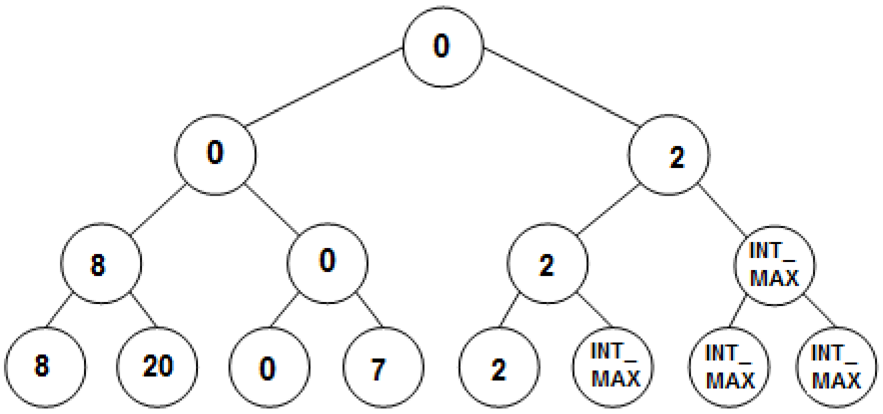
Description

Exercise 1: This is a practice on implementing of an almost complete binary tree. It is designed to implement a naïve sorting algorithm followed by analyzing it in terms of space and time complexity. For sake of simplicity consider an array of n integer elements, `data`, to be sorted in ascending order. The following steps explains the algorithm of naïve sort.

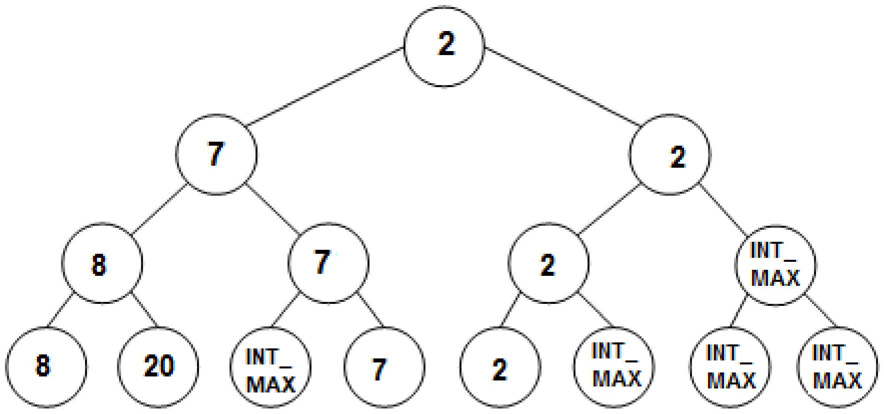
1. **Create Your Tree:** First, create a complete binary tree; a tree with all leaves at highest level with height, `h = ceil(log`$_2$`n) + 1` [for example, `ceil(2.3)` is 3]. Therefore, the number of leaves in the tree is always equal or more than n. Now, store all the elements of the array in the first n leaves of highest level of tree from left to right. Store a value (E) greater than any element in the array for every empty leaf; for example, you can use `INT_MAX` of `climits` library as the value of an empty leaf. The following tree shows an example where `Data={8, 20, 0, 7, 2}`, `h=ceil(log`$_2$`(5))+1=4` and `E=INT_MAX`.



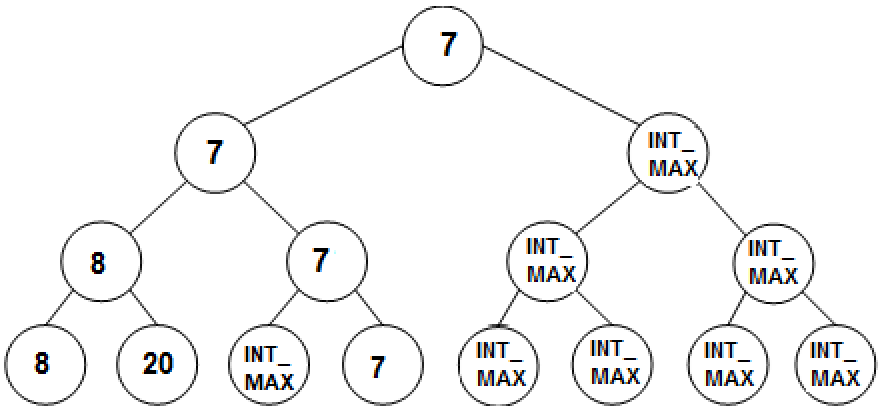
2. **Populate Your Tree:** As you can see in the above tree, all the internal nods have no value. To populate and initialize the internal nodes of tree, start from the bottom of the tree, and assign each internal node with the minimum value of its two children, as shown in the following tree. This ensures to assign the smallest value in the list, `min`, to the root. You must then store this minimum value in its right place in the original list, ie. `data`. After this step, the first item of `data` is updated like: *`data={0, 20, 0, 7, 2}`*. Note that valid elements in the original list are shown in red.



3. **Finding Next Minimum:** First stores E (or INT_MAX) in the leaf that contains `min` value (is shown in following tree). Then starting from the bottom and assign each internal nodes to the minimum of its two children. Again the root carries the next minimum value of the original list, ie: *data= {0, 2, 0, 7, 2}*. The following figure displays the tree after one iteration of finding the next minimum value.



4. **Sorting Data:** repeat step 3 until the root becomes E (or INT_MAX). The following tree shows another iteration of the algorithm (finding next minimum is considered as an iteration), and `data` is updated to *{0, 2, 7, 7, 2}*.



a. **[5 marks]** Choose the most efficient data structure to implement the naïve sort algorithm. Justify your choice (**answers.pdf**).

b. **[15 marks]** Write a function named **naiveSort()** that receives the original list, and then sorts it in ascending order using the algorithm described in above 4 steps (**naiveSort.cpp**).

c. **[5 marks]** Calculate the time complexity of naiveSort() function (**answers.pdf**).

d. **[5 marks]** Calculate the space complexity of naiveSort() function (**answers.pdf**).

Exercise 2: Consider binary trees that have single characters stored at each internal node.

[10 marks] Draw one binary tree that will spell out the phrase: `ANUPSIDEDOWNTREE` when nodes are visited in preorder, and `UNPADIESDNWTOERE` when visited in inorder. (Draw only one tree!) (**answers.pdf**).

Submit to D2L

Make a **zip file** named **StudentNumber-lab5.zip** including all related files by the end of the lab time**.** For example, if your student number is 10023449, the submitted file must be named as **10023449-lab5.zip**.