CS 3358 Assignment 3

Due: 11:59pm Monday, Apr 14, 2025

Instructor: Tsz-Chiu Au (chiu.au@txstate.edu)
Doctoral Instructional Assistant: Minhyuk Park (dmz44@txstate.edu)
Grader: Dhruvil Thummar (kwt32@txstate.edu)

In this assignment, you will implement heapsort according to the description in our slides. We provide you with two C++ files: `heapsort.h` and `main.cpp`. The former file contains a partial implementation of the template class `HeapSort`, whereas the latter file contains the main function that allows users to test the heapsort. You will implement some methods in `heapsort.h`, and you should not touch `main.cpp`. However, you should read `main.cpp` to see how it uses the template class in `HeapSort`.

`heapsort.h` contains the following methods:

- `HeapSort(T* numbers, int size)`
  (10%) The constructor of the class. The parameter `numbers` is the sequence of numbers entered by the user, and the parameter `size` is the length of the sequence. The constructor should copy all the numbers in `numbers` to `arr[]` and set the member fields `size` and `heap_size` to `size`. Notice that the first element in `arr[]` is left empty, as discussed in the slides.

- `int getSize() const`
  This method returns `size`, the size of the number sequence you stored in this object. The size should be fixed after the creation of the `HeapSort` object. Notice that the first element in `arr[]` is left empty, as discussed in the slides, and hence `size` is not the size of `arr[]`. `size` cannot be larger than `MAX_HEAPSORT_SIZE`, the maximum length of the number sequence. We have already implemented this method for you.

- `int getHeapSize() const`
  This method returns `heap_size`, the size of the heap. In heapsort, the array is divided into two parts: The first part stores the heap, and the second part stores the sorted numbers. `heap_size` is the size of the first part. `heap_size` should be an integer smaller than or equal to `size`, and `heap_size` can be zero. Since heapsort is an in-place algorithm, it is okay that sometimes some edges in the heap in the first part violate the heap property, and we will still call the first part the heap even when some edges violate the heap property. We have already implemented this method for you.

- `bool isValidNodeId(int i) const`
  (5%) This method returns true if and only the index `i` is one of the indices of the heap in the array. In other words, this method will return false if `i` is zero (i.e., the empty element), an index in the second part of the array (which is not part of the heap), or an index that is out of bound. We have already implemented this method for you.

- `int getLeftChildIndex(int i) const`
  (5%) This method returns the index of the left children of the node of the given index `i` in the heap. This method does not check whether `i` is a valid node id in the heap or whether the returned index is a valid node id in the heap. The caller must call `isValidNodeId()` to check the validity of `i` and the returned index.

- `int getRightChildIndex(int i) const`
  (5%) This method returns the index of the right children of the node of the given index `i` in the heap. This method does not check whether `i` is a valid node id in the heap or whether the returned index is a valid node id in the heap. The caller must call `isValidNodeId()` to check the validity of `i` and the returned index.

- `int getParentIndex(int i) const`
  (5%) This method returns the index of the parent of the node of the given index `i` in the heap. This method does not check whether `i` is a valid node id in the heap or whether the returned index is a valid node id in the heap. The caller must call `isValidNodeId()` to check the validity of `i` and the returned index.

- `bool isLeaf(int i)`
  (5%) This method returns true if `i` is the index of a leaf node in the heap. This method does not check whether `i` is a valid node id in the heap. The caller must call `isValidNodeId()` to check the validity of `i`.

- `bool isInternalNode(int i)`
  (5%) This method returns true if `i` is the index of an internal node in the heap. This method does not check whether `i` is a valid node id in the heap. The caller must call `isValidNodeId()` to check the validity of `i`.

- `void buildMaxHeap()`
  (10%) This method builds a heap using all the numbers in `arr[]`. This method will set `heap_size` to `size` since the entire array has been a heap.

- `void swapNode(int i, int j)`
  This method swaps the value at the two nodes of the given indices. We have already implemented this method for you.

- `void reduceHeapSizeByOne()`
  This method reduces the size of the heap by 1. We have already implemented this method for you.

- `void maxHeapify(int i)`
  (10%) This method runs the max-heapify procedure at the node of the given index `i`.

- `void heapSort()`
  (10%) This method runs heapsort for the entire array. This method should ignore all the sorting efforts that have been done previously and start with `buildMaxHeap()` to build a heap for the entire array. In the end, it will set `heap_size` to zero since the numbers have been sorted.

- `void printArray()`
  This method prints the numbers in `arr[]`. We have already implemented this method for you.

- `void printUnsortedPartOfArray()`
  This method prints the numbers in the first part of `arr[]`, which stores the heap. We have already implemented this method for you.

- `void printSortedPartOfArray()`
  This method prints the numbers in the second part of `arr[]`, which stores the sorted numbers. We have already implemented this method for you.

- `void printPreorderTraversal(int i=1)`
  (10%) This method prints the numbers in the preorder traversal of the heap. You do not print any number in the sorted part of `arr[]`.

- `void printPostorderTraversal(int i=1)`
  (10%) This method prints the numbers in the postorder traversal of the heap. You do not print any number in the sorted part of `arr[]`.

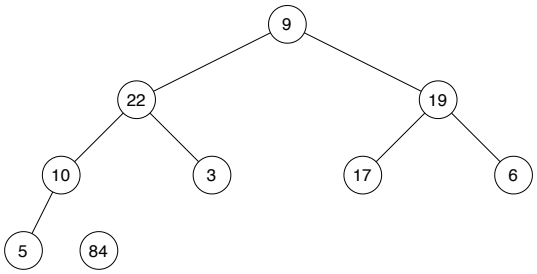
- `void printInorderTraversal(int i=1)`
  (10%) This method prints the numbers in the inorder traversal of the heap. You do not print any number in the sorted part of `arr[]`.

- `void drawHeapGraphically()`
  (20% bonus) This method draws the complete binary tree of the heap graphically. Please read the description of this method in the next section.

In most methods, we rely on the caller to call `isValidNodeId()` to check the validity of an index or the returned index. Therefore, there is no need to call `isValidNodeId()` in the method, except for the assert statements for debugging.

Bonus Task (20%):

We have a bonus task in this assignment, as requested by some students in the course evaluation. In this bonus task, you will implement a method in the template class `HeapSort` to draw the heap as a binary tree. For example, in the In-class Exercise 11, the second step of the heapsort is



Then, your program should replicate this step with the following commands:

```
Please enter your option: a
Please enter a sequence of nonnegative integers (-1 signals the end of the sequence):
5 3 17 10 84 19 6 22 9 -1

Please enter your option: c
The max-heap has been built.

Please enter your option: d
The root node and the last node have been swapped.

Please enter your option: e
The heap size has been reduced by one.

Please enter your option: s
The drawing of the heap is:
                     9
                     |
          -------------------------
         22                      19
          |                       |
    -------------           -------------
   10           3          17           6
    |
 ----
 5
```

This bonus task is to draw the binary tree of the heap, as shown in the above figure. The drawing of the binary tree should look exactly the same as the drawing in the above figure. However, partial points will be given if the drawing looks slightly different. You can assume all integers in the binary tree have at most two digits. Both ends of an edge must reach the rightmost digit of the numbers. Notice that *different numbers in the binary tree occupy different columns in the drawing*; no two numbers are drawn in the same column. Your code should be able to draw complete binary trees as small as an empty tree, as well as large complete binary trees with many levels.

You must put your code of this bonus task in the method `drawHeapGraphically()`, though you can also implement some private methods to help `drawHeapGraphically()`. Please add comments in `drawHeapGraphically()` to describe how your code works.

This bonus task is optional, and your grade will not be affected if you choose not to do this bonus task. If you choose *not* to do this task, you *must* leave `drawHeapGraphically()` empty except for the TODO comment.

Implementation:

In this assignment, you can modify any code in `heapsort.h` as long as you can compile

`main.cpp` with `heapsort.h`. In order words, you are not limited to adding code to the comments "`// TODO`" in `heapsort.h`. Therefore, you have the freedom to include more header files, add some private variables or methods, etc. However, it is likely you do not need to add new variables or methods to the template class `HeapSort` in this exercise.

Please do *not* modify `main.cpp` since you will not submit `main.cpp`. Please make sure `heapsort.h` contains a template class `HeapSort` with the methods used by `main.cpp`.

Please make sure that you add comments to `heapsort.h` to help the grader understand your code, especially the inputs and the outputs of the methods. If you add your own private variables or methods, please describe what they are.

Compilation:

You can compile the programs by the following commands on the departmental servers:

```
g++ -std=c++20 -O3 -o hsort main.cpp
```

`main.cpp` accepts an option `-q`, which stands for the quiet mode that suppresses some of the outputs in the program so that the grading script can be simplified. This option is used by our grader only and students can ignore this option.

Sample Program:

In this exercise, we provide you with a sample program called `hsort_sample` so that you can compare the outputs of your program with the outputs of the sample program. You can run the sample program directly on the departmental machine since the sample program was compiled on the departmental machine. Please make sure the outputs of your program are the same as the outputs of the sample program. This can help our grader avoid mistakes during grading. If you encounter any bugs in the sample program, please email the instructor.

Submission:

You should put your implementation of `heapsort.h` in a folder named `a3_yourNetID`, and then compress the folder into a zip file named `a3_yourNetID.zip`. For example, if your net id is `zz567`, the folder is `a3_zz567`, and the zip file is `a3_zz567.zip`. Finally, please upload the zip file to Canvas. Please do not put other files in the zip file. You are responsible for checking whether the zip file is not corrupted and contains your program before the submission. The deadline is strict, and we do not accept any submissions after the deadline.