1 Lab Details

- Maximum Possible Points: 100
- Lab Timings:
  1. Monday Lab: Oct 9, 9:00 AM–10:50 AM
  2. Wednesday Lab: Oct 11, 12:00 PM–1:50 PM
  3. Friday Lab: Oct 13, 12:00 PM–1:50 PM
- Lab Due:
  1. Monday Lab: Oct 22, 5:00 PM
  2. Wednesday Lab: Oct 24, 5:00 PM
  3. Friday Lab: Oct 26, 5:00 PM

2 Assignment Instructions

In this lab you will implement a max 5-heap, minmax heap, and some basic operations on both the data structures. You will also do performance analysis on some of the operations you have implemented for the max 5-heap and minmax heap. The lab kit includes a main function that will be used to run and test your implementation of the data structure (in this case the max 5-heap and minmax heap). You will have to fill in the necessary code in function body marked as — IMPLEMENT_ME(). You should comment out or delete the line containing IMPLEMENT_ME() macros after you are done implementing the function.

Please use the Makefile given with the lab-kit to compile and test your code. The grader will be using the make file to compile and test your code. The sample test data will be read from the file data.txt. You may not use the standard template library (STL) or any container data structures from cpp standard library. Please use the principles of object oriented programming to design your program. You are free to add new private data members and private methods in your classes. You may not change the signature of the public methods (declared in *.hpp files), the implementations of the public methods declared in the template lab kit will be used to test your code.

2.1 Max5Heap operations

Implement the Max5Heap data structure and the associated operations on it as mentioned in §3

2.2 MinMaxHeap operations

Implement the MinMaxHeap data structure and the associated operations on it as mentioned in §4
2.3 Performance analysis of building max 5-heap
(void Max5Heap::buildHeap())

Generate a list of the following sizes: 10, 50, 100, 250, 500, 750, 1000, 2500, 5000, 7500, and 10000 containing random words. Call the method void Max5Heap::buildHeap() to heapify the word list of different sizes mentioned above and time this operation. For each of the sizes mentioned above repeat the experiment at-least 10 times and note the timings for each run. Plot a graph of the list size (x-axis) vs. the average time taken in milliseconds (or nanoseconds) (y-axis).

2.4 Performance analysis of deleting minimum element from max 5-heap
(std::string Max5Heap::deleteMinElem())

Generate a max 5-heap of the following sizes: 10, 50, 100, 250, 500, 750, 1000, 2500, 5000, 7500, and 10000 containing random words. Call the function std::string Max5Heap::deleteMinElem() to remove the minimum element from the heap and time it.

For each of the sizes mentioned above repeat the experiment at-least 10 times and note the timings for each run. Plot a graph of the list size (x-axis) vs. the average time taken in milliseconds (or nanoseconds) (y-axis).

2.5 Performance analysis of deleting maximum element from max 5-heap
(std::string Max5Heap::deleteMaxElem())

Generate a max 5-heap of the following sizes: 10, 50, 100, 250, 500, 750, 1000, 2500, 5000, 7500, and 10000 containing random words. Call the function std::string Max5Heap::deleteMaxElem() to remove the maximum element from the heap and time it.

For each of the sizes mentioned above repeat the experiment at-least 10 times and note the timings for each run. Plot a graph of the list size (x-axis) vs. the average time taken in milliseconds (or nanoseconds) (y-axis).

2.6 Performance analysis of building a minmax heap
(void MinMaxHeap::buildHeap())

Generate a list of the following sizes: 10, 50, 100, 250, 500, 750, 1000, 2500, 5000, 7500, and 10000 containing random words. Call the method void MinMaxHeap::buildHeap() to heapify the word list of different sizes mentioned above and time this operation. For each of the sizes mentioned above repeat the experiment at-least 10 times and note the timings for each run. Plot a graph of the list size (x-axis) vs. the average time taken in milliseconds (or nanoseconds) (y-axis).

2.7 Performance analysis of deleting minimum element from minmax heap
(std::string MinMaxHeap::deleteMinElem())

Generate a minmax heap of the following sizes: 10, 50, 100, 250, 500, 750, 1000, 2500, 5000, 7500, and 10000 containing random words. Call the function std::string MinMaxHeap::deleteMinElem() to remove the minimum element from the heap and time it.

For each of the sizes mentioned above repeat the experiment at-least 10 times and note the timings for each run. Plot a graph of the list size (x-axis) vs. the average time taken in milliseconds (or nanoseconds) (y-axis).

2.8 Performance analysis of deleting maximum element from minmax heap
(std::string MinMaxHeap::deleteMaxElem())

Generate a minmax heap of the following sizes: 10, 50, 100, 250, 500, 750, 1000, 2500, 5000, 7500, and 10000 containing random words. Call the function std::string MinMaxHeap::deleteMaxElem() to remove the maximum element from the heap and time it.

For each of the sizes mentioned above repeat the experiment at-least 10 times and note the timings for each run. Plot a graph of the list size (x-axis) vs. the average time taken in milliseconds (or nanoseconds) (y-axis).
3 Operations on max 5-heap

You may use the array based design for max 5-heap implementation. The array size should be 15000. You may use lexicographic ordering to define <, = or > relations on the string elements.

1. Build heap: signature void Max5Heap::buildHeap()
   This method restructures input data into a proper heap format. This method may be called inside the constructor after instantiation of private member variables is completed. You must use the bottom-up technique to build the heap. Fig. 1 is transformed in to heap as shown in fig. 2 after the operation is performed successfully.

   ![Figure 1: Tree structure after reading data.txt](image1)
   ![Figure 2: Max5Heap structure after buildHeap](image2)

2. Add element: signature void Max5Heap::addElem(std::string);
   This function adds an element to max 5-heap. Eg. consider the max 5-heap generated from data.txt (Refer fig. 3). Adding an element, say “zum”, will change the heap structure to item 2

   ![Figure 3: Max5Heap structure after reading from data.txt](image3)
   ![Figure 4: Max5Heap structure after adding element](image4)

3. Delete element: signature void Max5Heap::deleteElem(std::string elem);
   Deletes the all the elements which are equal to the given argument elem. from the heap. If the node to be deleted is not at lowest level, choose the right most node from the lowest level to replace the node to be deleted and then call buildHeap to re-instantiate the heap property. If the element does not exist, there should be no change in the heap structure.

4. Delete minimum element: signature std::string Max5Heap::deleteMinElem()
   Deletes the minimum element from the heap and if necessary, restructures it to maintain the heap property. It should return the element deleted from the heap. Eg. executing the delete min function on item 4 will result in a heap as shown in fig. 6 and return “beggar”.

5. Delete maximum element: signature std::string Max5Heap::deleteMaxElem()
   Deletes the maximum element from the heap and if necessary, restructures it to maintain the heap property. It should return the element deleted from the heap.

6. Exists: signature bool Max5Heap::exists(std::string elem);
   Should return true if the element exists in the heap, return false otherwise
7. Levelorder print: signature void Max5Heap::levelorderPrint();
Prints the max 5-heap structure in a level order fashion. For example, the heap structure created with data.txt should be printed as shown below:

```
work
spell end west search chisel
men beggar spell owen presidential catch
```

4 Operations on minmax heap

You may use the array based design for minmax heap implementation. The array size should be 15000. You may use lexicographic ordering to define <, = or > relations on the string elements.

1. Build heap: signature void MinMaxHeap::buildHeap();
This method restructures input data into a proper heap format. This method may be called inside the constructor after instantiation of private member variables is completed. You must use the bottom-up technique to build the heap. fig. 7 is transformed in to heap as shown in fig. 8 after the operation is performed successfully.

2. Add element: signature void MinMaxHeap::addElem(std::string);
This function adds an element to minmax heap. Eg. consider the minmax heap generated from data.txt (Refer fig. 9). Adding an element, say “ability”, will change the heap structure to fig. 10.

3. Delete element: signature void MinMaxHeap::deleteElem(std::string elem);
Deletes the all the elements which are equal to the given argument elem. from the heap. If the node to be deleted is not at lowest level, choose the right most node from the lowest level to replace the node to be deleted and then call buildHeap to re-instantiate the heap property. If the element does not exist, there should be no change in the heap structure.
4. Delete minimum element: signature `std::string MinMaxHeap::deleteMinElem()`
   Deletes the minimum element from the heap and if necessary, restructures it to maintain the heap property.
   It should return the element deleted from the heap. E.g., executing the delete min function on fig. 11 will
   result in a heap as shown in fig. 12 and return “beggar”

5. Delete maximum element: signature `std::string MinMax::deleteMaxElem()`
   Deletes the maximum element from the heap and if necessary, restructures it to maintain the heap property.
   It should return the element deleted from the heap.

6. Exists: signature `bool MinMaxHeap::exists(std::string elem);`
   Should return `true` if the element exists in the heap, return `false` otherwise.

7. Levelorder print: signature `void MinMaxHeap::levelorderPrint();`
   Prints the minmax heap structure in a level order fashion. For example, the heap structure created with
   data.txt should be printed as shown below:

   beggar
   work end
   earth owen catch spell
   west site men presidential chisel

5 Questions

Please answer the following questions in not more than 5 lines each and submit it with your implemented code in
the PDF format (document should be named *-writeup.pdf)
1. What is the worst case algorithmic asymptotic complexity i.e. $O(\cdot)$ of each of the operations that you have implemented. (10 points)
   a. Build a max 5-heap ($\text{void Max5Heap::buildHeap()}$).
   b. Delete min element from the max 5-heap ($\text{void Max5Heap::deleteMinElem()}$).
   c. Delete max element from the max 5-heap ($\text{void Max5Heap::deleteMaxElem()}$).
   d. Build a minmax heap ($\text{void MinMaxHeap::addElem(std::string)}$).
   e. Delete min element from the minmax heap ($\text{void MinMaxHeap::deleteMinElem()}$).
   f. Delete max element from the minmax heap ($\text{void MinMaxHeap::deleteMaxElem()}$).

2. Is the average case complexity equal to the worst case complexity in each of the 6 cases above? If not, mention the average case complexity for each of the above operations in a., b., c., d., e. and f.

6 Report

Your report should consist of the following sections:

1. Overall organization of the experiment in not more than 10 lines.
2. Tabulated data obtained from the 6 analysis tasks. You should include all the 10 timing values for each size and also the average value you calculate for plotting the graph.
3. 6 Graphs generated from the tabulated data (1 for each operation).

7 Grading Scheme

- Your submitted code should compile and run on the EECS unix machines. (Please use cycle2.eecs.ku.edu/cycle3.eecs.ku.edu/EECS lab machines. g++ v6.2.1) There should be no memory leaks and compilation warnings. (20 points)
- Your code will be tested against the example flow given at the end of the instruction document. (20 points)
- Hidden test case suite run by the grader on your implementation. (20 points)
- Write up answering questions. (10 points)
- Report on performance analysis with graphs and generated data. (30 points)
- All function implementations are equally weighted.

8 Deliverables

1. Code that compiles and runs with Makefile. Your submitted code archive should compile $\text{make instal-all}$ and run seamlessly after un-archiving on the EECS lab machines.
2. Write up answering questions from §5
3. Report on performance analysis as per §2.3, §2.4, §2.5, §2.6, §2.7 and §2.8
4. Code archive and write-up should have correct naming conventions as described in §9

9 Submission and Miscellaneous Hints

1. Please add the grader’s (Dravid Joseph) email id in the To section of the mail (dravidjoseph@ku.edu) and my (Apoorv Ingle) email id in CC (apoorv.ingle@ku.edu)
2. Your subject line for the submission should be of the form [EECS 560] Lab-<Lab #> <Lab Day> <Your KU username> eg. [EECS 560] Lab-7 M j543h898
3. Your reports should be named as `<your KU username>-minmax-vs-max5-heap-lab7-report.pdf` and your writeups should be named as `<your KU username>-minmax-vs-max5-heap-lab7-writeup.pdf`

4. Your code tar archive will be automatically named in correct format by running `make tar`

5. You may use the pre-existing random number generation helper functions `rand` and `srand` to generate random values for your analysis tasks.

6. You may use `random-word-generator-demo.cpp` to generate random words. It uses `word-list.txt` as its data store.

7. You may use `Timer` class from `timer.hpp` to time the operation given in the lab-kit.

8. Expand the tar ball: `$ tar xvf <filename>.tar.gz`

9. Make cheat-sheet:
   - compiling and linking your program: `make -B clean install`
   - testing your program: `make max5h-test` or `make minmaxh-test`
   - bundle your code in a tar archive: `make tar`
   
   *Note: Please change XXXXX in first line of the Makefile into your KU username of the format (j052h567) before running `make tar`*

10. Counting always starts from 0

10 Sample Test Cases

Please note this is just for illustration using `max5h-data.txt` file as input

10.1 Test cases for Max5 Heap

$ make max5h-install
$ ./out/max5h-main

No input file given, using default data.txt
Please choose one of the following commands:
1: add
2: delete
3: exists
4: delete min
5: delete max
6: print level order
7: exit
>> 6
Printing heap:
work
spell end west earth chisel
men beggar site owen presidential catch

Please choose one of the following commands:
1: add
2: delete
3: exists
4: delete min
5: delete max
6: print level order
7: exit
>> 1
Enter element to be added: zum
Please choose one of the following commands:
1: add
2: delete
3: exists
4: delete min
5: delete max
6: print level order
7: exit

>> 6
Printing heap:
zum
spell work west earth chisel
men beggar site owen presidential catch end

Please choose one of the following commands:
1: add
2: delete
3: exists
4: delete min
5: delete max
6: print level order
7: exit

>> 4
Deleted Min element: beggar

Please choose one of the following commands:
1: add
2: delete
3: exists
4: delete min
5: delete max
6: print level order
7: exit

>> 6
Printing heap:
zum
spell work west earth chisel
men end site owen presidential catch

Please choose one of the following commands:
1: add
2: delete
3: exists
4: delete min
5: delete max
6: print level order
7: exit

>> 5
Deleted max element: zum

Please choose one of the following commands:
1: add
2: delete
3: exists
4: delete min
5: delete max
6: print level order
7: exit

Printing heap:
work
spell catch west earth chisel
men end site owen presidential

Please choose one of the following commands:
1: add
2: delete
3: exists
4: delete min
5: delete max
6: print level order
7: exit

> 5
Deleted max element: work

Please choose one of the following commands:
1: add
2: delete
3: exists
4: delete min
5: delete max
6: print level order
7: exit

> 6
Printing heap:
west
spell catch presidential earth chisel
men end site owen

Please choose one of the following commands:
1: add
2: delete
3: exists
4: delete min
5: delete max
6: print level order
7: exit

> 3
Enter element to check existence: beggar
Element does not exist

Please choose one of the following commands:
1: add
2: delete
3: exists
4: delete min
5: delete max
6: print level order
7: exit

> 2
Enter element to be deleted: chisel

Please choose one of the following commands:
1: add
2: delete
3: exists
4: delete min
5: delete max
6: print level order
7: exit

>> 6
Printing heap:
west
spell catch presidential earth owen
men end site

Please choose one of the following commands:
1: add
2: delete
3: exists
4: delete min
5: delete max
6: print level order
7: exit

>> 7

10.2 Test cases for MinMax Heap

Please note this is just for illustration using minmaxh-data.txt file as input

$ make minmaxh-install
$ ./out/minmaxh-main

No input file given, using default minmaxh-data.txt
Please choose one of the following commands:
1: add
2: delete
3: exists
4: delete min
5: delete max
6: print level order
7: exit

>> 6
Printing heap:
beggar
work end
earth owen catch spell
west site men presidential chisel

Please choose one of the following commands:
1: add
2: delete
3: exists
4: delete min
5: delete max
6: print level order
7: exit
Enter element to be added: ability

Please choose one of the following commands:
1: add
2: delete
3: exists
4: delete min
5: delete max
6: print level order
7: exit

Printing heap:
ability
work end
earth oven beggar spell
west site men presidential chisel catch

Please choose one of the following commands:
1: add
2: delete
3: exists
4: delete min
5: delete max
6: print level order
7: exit

Deleted Min element: ability

Please choose one of the following commands:
1: add
2: delete
3: exists
4: delete min
5: delete max
6: print level order
7: exit

Printing heap:
beggar
work end
earth oven catch spell
west site men presidential chisel

Please choose one of the following commands:
1: add
2: delete
3: exists
4: delete min
5: delete max
6: print level order
7: exit

Deleted max element: work

Please choose one of the following commands:
1: add
2: delete
3: exists
4: delete min
5: delete max
6: print level order
7: exit

>> 6
Printing heap:
    beggar
    west end
    earth owen catch spell
    chisel site men presidential

Please choose one of the following commands:
1: add
2: delete
3: exists
4: delete min
5: delete max
6: print level order
7: exit

>> 5
Deleted max element: west

Please choose one of the following commands:
1: add
2: delete
3: exists
4: delete min
5: delete max
6: print level order
7: exit

>> 6
Printing heap:
    beggar
    site end
    earth owen catch spell
    chisel presidential men

Please choose one of the following commands:
1: add
2: delete
3: exists
4: delete min
5: delete max
6: print level order
7: exit

>> 3
Enter element to check existence: catch
Element exists

Please choose one of the following commands:
1: add
2: delete
3: exists
4: delete min
5: delete max
6: print level order
7: exit

>> 2
Enter element to be deleted: Owen

Please choose one of the following commands:
1: add
2: delete
3: exists
4: delete min
5: delete max
6: print level order
7: exit

>> 6
Printing heap:
beggar
site end
earth men catch spell
chisel presidential

Please choose one of the following commands:
1: add
2: delete
3: exists
4: delete min
5: delete max
6: print level order
7: exit

>> 7