CSCE 5150 – Analysis of Computer Algorithms

Assignment 2

Due: 3/8/2023 11:59PM on Canvas
(100 points)

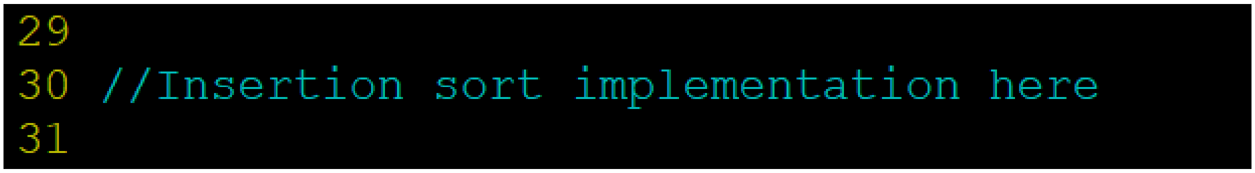
Question. Comparison-based sorting algorithm

In our class, we discussed four comparison-based sorting algorithms (insertion sort, selection sort, heapsort and mergesort), in this assignment, we are going to implement the algorithm we discussed and execute them to sort the given inputs in the ascending order and compare their efficiency by measuring their running time.

In `public.zip`, we are providing you with the following files:

```
public/
  Insertion/insertionSort.cpp    <-implement the insertion sort algorithm here
  Selection/selectionSort.cpp    <-implement the selection sort algorithm here
 Heapsort:
    demo.cpp   <-main method to try your implementation
    minheap.cpp<-implementation of minheap here
    minheap.hpp<-header file
 Mergesort:
    mergesort.cpp <- main method to try your implementation
    mergesort.hpp <- implementation of mergesort here
  inputs        <-folder with several sample inputs
[only highlighted file needs to be modified.]
```

Specifically, in the insertionSort.cpp, you need to implement the insertion sorting algorithm:



```

//Insertion sort implementation here

```

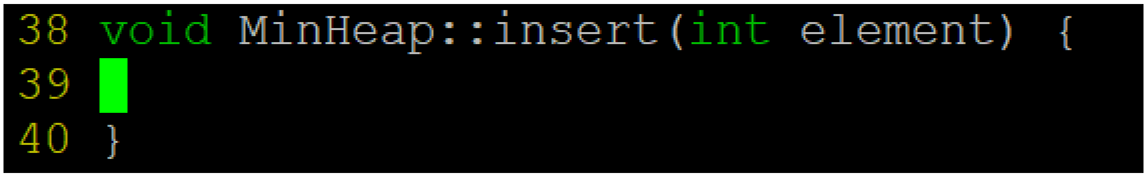
in the selectionSort.cpp file, you need to implement the selection sorting algorithm:

```

//selection sort algorithm implementation here

```

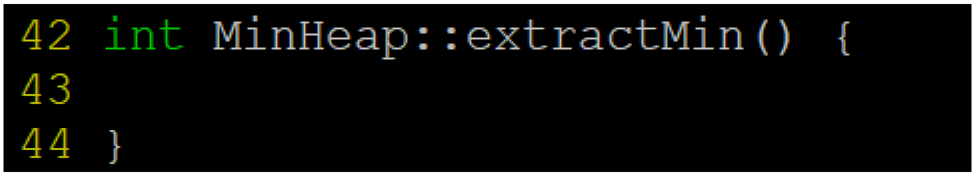
in the minheap.cpp file, you need to complete the **insert:**



```
void MinHeap::insert(int element) {

}
```

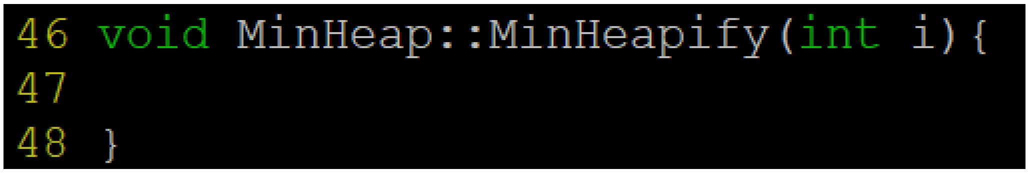
extractMin:



```
int MinHeap::extractMin() {

}
```

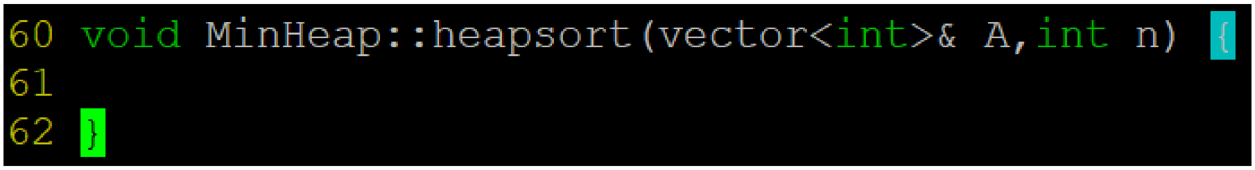
MinHeapify:



```
void MinHeap::MinHeapify(int i){

}
```

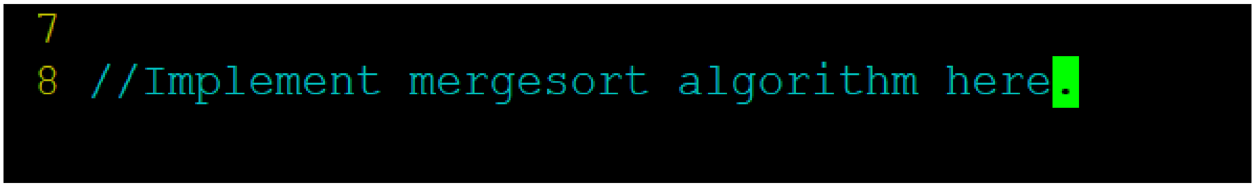
and **heapsort** method:



```
void MinHeap::heapsort(vector<int>& A,int n) {

}
```

And in the mergesort.hpp file, you need to implement the mergesort sorting algorithm:



```

//Implement mergesort algorithm here.
```

Test you implementations on the CELL machine (detailed instructions: please refer to CELL machine, CELL machine on MAC).

- To compile, run

```
$ g++ *.cpp
```

- To test your implementation with a sample input file, run

```
$ ./a.out ../inputs/input.10.1
```

This will test one sample file.

- To get the running time, run

```
$ time for f in ../inputs/input.10.1; do echo $f; ./a.out $f; done
```

This will test one sample file and give you running time. The command time will let you know how long it took to run the code.

Submission Instructions:

A zip file contains all the following information:

- **All files** that are needed to compile and run your code. Include a brief **README file** explaining how to run your code. Making sure that based on your readme file, we should be able to compile and execute your code.
- A WORD/PDF file with a brief explanation of the four sorting algorithms (including their running time analysis [see lecture slides]). Additionally, create tables to document the running times (real time) for all input files using various sorting algorithms. Examples of these tables are provided below:

	10.1	10.2	10.3	10.4	10.5	Average
Insertion sort	0m0.004s	0m0.003s	0m0.003s	0m0.003s	0m0.004s	0m0.0034s
Selection sort	0m0.005s	0m0.004s	0m0.004s	0m0.004s	0m0.004s	0m0.0042s
Heapsort	0m0.004s	0m0.003s	0m0.003s	0m0.003s	0m0.003s	0m0.0032s
Mergesort	0m0.003s	0m0.004s	0m0.003s	0m0.003s	0m0.003s	0m0.0032s

This is input size 10

	100.1	100.2	100.3	100.4	100.5	Average
Insertion sort	0m0.003s	0m0.003s	0m0.004s	0m0.003s	0m0.003s	0m0.0032s
Selection sort	0m0.004s	0m0.003s	0m0.003s	0m0.004s	0m0.003s	0m0.0034s
Heapsort	0m0.003s	0m0.003s	0m0.003s	0m0.003s	0m0.003s	0m0.003s
Mergesort	0m0.003s	0m0.003s	0m0.003s	0m0.003s	0m0.003s	0m0.003s

This is input size 100

	1000.1	1000.2	1000.3	1000.4	1000.5	Average
Insertion sort	0m0.005s	0m0.005s	0m0.006s	0m0.006s	0m0.007s	0m0.0058s
Selection sort	0m0.005s	0m0.009s	0m0.006s	0m0.005s	0m0.005s	0m0.006s
Heapsort	0m0.004s	0m0.003s	0m0.005s	0m0.004s	0m0.004s	0m0.004s
Mergesort	0m0.004s	0m0.004s	0m0.004s	0m0.004s	0m0.004s	0m0.004s

This is input size 1000

	10000.1	10000.2	10000.3	10000.4	10000.5	Average
Insertion sort	0m0.232s	0m0.237s	0m0.184s	0m0.185s	0m0.276s	0m0.2228s
Selection sort	0m0.186s	0m0.187s	0m0.198s	0m0.186s	0m0.186s	0m0.1886s
Heapsort	0m0.018s	0m0.018s	0m0.016s	0m0.008s	0m0.010s	0m0.014s
Mergesort	0m0.018s	0m0.024s	0m0.027s	0m0.025s	0m0.019s	0m.0.0226s

This is input size 10000

	100000.1	100000.2	100000.3	100000.4	100000.5	Average
Insertion sort	0m18.277s	0m18.375s	0m18.157s	0m23.955s	0m20.999s	0m19.952s
Selection sort	0m18.217s	0m18.265s	0m18.101s	0m18.233s	0m18.098s	0m18.382s
Heapsort	0m0.132s	0m0.121s	0m0.161s	0m0.171s	0m0.167s	0m0.150s
Mergesort	0m0.201s	0m0.185s	0m0.176s	0m0.179s	0m0.176s	0m0.183s

This is input size 10,0000

	1000000.1
Insertion sort	40m4.539s
Selection sort	30m36.622s
Heapsort	0m0.729s
Mergesort	0m1.464s

This is input size 100,0000

Utilizing the table data, summarize what you’ve found in your experiment with a one-or-two paragraph.