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:

```
29
30 //Insertion sort implementation here
31
```

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

```
29
30 //selection sort algorithm implementation here
31
```

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

```
38 void MinHeap::insert(int element) {
39 
40 }
```

extractMin:

```
42 int MinHeap::extractMin() {
43 
44 }
```

MinHeapify:

```
46 void MinHeap::MinHeapify(int i){
47 
48 }
```

and **heapsort** method:

```
60 void MinHeap::heapsort(vector<int>& A,int n) {
61 
62 }
```

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

```
 7 
 8 //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	
Selection sort	0m0.003s	0m0.004s	0m0.003s	0m0.003s	0m0.003s	
Heapsort	0m0.002s	0m0.002s	0m0.001s	0m0.002s	0m0.001s	
Mergesort	0m0.003s	0m0.003s	0m0.003s	0m0.003s	0m0.003s	

This is input size 10

	100.1	100.2	100.3	100.4	100.5	Average
Insertion sort	0m0.004s	0m0.003s	0m0.003s	0m0.003s	0m0.003s	
Selection sort	0m0.003s	0m0.003s	0m0.003s	0m0.003s	0m0.003s	
Heapsort	0m0.001s	0m0.001s	0m0.001s	0m0.001s	0m0.001s	
Mergesort	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	0m0.006s	0m0.005s	0m0.005s	0m0.006s	0m0.005s	
Selection sort	0m0.005s	0m0.004s	0m0.005s	0m0.004s	0m0.007s	
Heapsort	0m0.001s	0m0.001s	0m0.001s	0m0.001s	0m0.001s	
Mergesort	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	0m0.187s	0m0.190s	0m0.188s	0m0.183s	0m0.210s	
Selection sort	0m0.063s	0m0.064s	0m0.042s	0m0.040s	0m0.068s	
Heapsort	0m0.001s	0m0.001s	0m0.002s	0m0.001s	0m0.001s	
Mergesort	0m0.011s	0m0.023s	0m0.027s	0m0.027s	0m0.024s	

This is input size 10000

	100000.1	100000.2	100000.3	100000.4	100000.5	Average
Insertion sort	0m18.314s	0m18.302s	0m18.197s	0m18.452s	0m18.361s	
Selection sort	0m3.224s	0m3.371s	0m3.424s	0m3.260s	0m2.970s	
Heapsort	0m0.002s	0m0.001s	0m0.002s	0m0.001s	0m0.001s	
Mergesort	0m0.260s	0m0.259s	0m0.262s	0m0.264s	0m0.275s	

This is input size 10,0000

	1000000.1
Insertion sort	
Selection sort	
Heapsort	
Mergesort	

This is input size 100,0000

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