CSCE 221 Cover Page
Program Assignment # 4

First Name: Rong Last Name: Xu UIN: 928009312

User Name: Abby-xu E-mail address: rongx0915@tamu.edu

Please list all sources in the table below including web pages which you used to solve or implement the current homework. If you fail to cite sources you can get a lower number of points or even zero, read more in the Aggie Honor System Office http://aggiehonor.tamu.edu/

Type of sources					
People					
Web pages (provide URL)					
Printed material					
Other Sources					

I certify that I have listed all the sources that I used to develop the solutions/code to the submitted work.

"*On my honor as an Aggie, I have neither given nor received any unauthorized help on this academic work.*"

Your Name (signature) Rong Xu Date 2020/11/03 The Pro-

1. The description of an assignment problem.
   (a) This lab includes two parts. The first part is implements a minimum priority queue with a linked list as sorted MPQ and a vector as unsotred MPQ. The second part is implement a minimum priority queue with a binary heap and used for testing cpu.

2. The description of data structures and algorithms used to solve the problem.
   (a) Provide definitions of data structures for sorted MPQ
      i. The data structure I used is a Linkedlist for solving the sorted MPQ. In the class I have a node pointer named Min which just the header of the whole MPQ. For each node in the linkedlist is a struct including a T type data and a next pointer to point the next node in the linkedlist. The class is inherited by MPQ head file whihc inluding the basic vitural function of a basic MPQ, I will go over that part later.

   (b) Provide definitions of data structures for unsorted MPQ
      i. The data structure I used is a vector for solving the unsorted MPQ. It's basicallly just a class with a private member, the vector, and other basic function from the MPQ head file that I will go over that part later.

   (c) Provide definitions of data structures for binaryheap MPQ
      i. The data structure I used is a vector similar to the previous one. However, I also used the binary heap to access with my data stored in the vector. The other function just as same as the unctions provided by MPQ class file. Besides, I implement two more functions, up_heap() and down_heap() which help me to insert the data into the vector and remove the minimum value in the vector.

   (d) Describe algorithms used to solve the problem.
      i. MPQ (used by both sorted MPQ and unsorted MPQ, and binaryheap)
         A. remove_min(): remove the minimum value in the data structure (either vector or linkedlist in this assignment)
         B. min(): find the minimum element in the data structure
         C. is_empty(): boolean function, to see if the data structure stores any element. (helper function)
         D. insert(): insert the element into the data structure. Since we don't have the build function, this is the only function used to add data.
      ii. Binary heap
         A. down_heap(): healper function, be used in the remove_min() function, when we delete the min value and put the last element on the first position, (the biggest element in the data structure) we need to let the max element "go down"
         B. up_heap(): helper function, be used in the insert function, when we push back a element into the vector(for the binary heap, we used vector data structure) that data will be in the last position which is belong to the largest element, so we need to check and let the data "go up".

   (e) Analyze the algorithms according to assignment requirements.
      i. remove_min()
         A. Sorted MPQ(Linkedlist): O(1) —since the linkedlist has already been sorted when we insert the data, so we can just remove the first element in the linked list and chenge the header pointer to the second element.
         B. unsorted MPQ(Vector): O(n) — since the elements in the vector aren't sorted so that if we want to remove the min value, we need to find the min value first, which will need O(n) times to run. Then when we remove the element, we will need O(n) time. So the total will be O(n) + O(n) = O(n)

C. Binary heap: O(log n) — When we remove the smallest element in the vector, it has same time complexcity of sorted MPQ, so basically we jsut remove the first element. However, we have to rearrange the element since we are using binary method to implement this MPQ. The data structure will be similar to binary tree. So in order to let the max element (after we delete the first element we move the last element to the first position) "go odwn". Since we are using the binary method, so it will only take O(log n).

ii. min()

A. Sorted MPQ(Linkedlist): O(1) — Since the linkedlist is sorted when we insert the data, so we can just return the value of the first element.

B. unsorted MPQ(Vector): O(n) — Since the vector is unsotred, so if we wanna find the minmum value in the vector, we have to go over every elements in the vector, which will take O(n) time.

C. Binary heap: O(1) — similar to the linkedlist mentioned above, the vector in teh binary heap is sorted, so we can just return the first element in the vector.

iii. is_empty(): O(1) — for all three class, since we have a private member called size in the class to track the size of data structure. So inorder to check if the data struture has any element, we will just use the number in the size.

iv. insert():

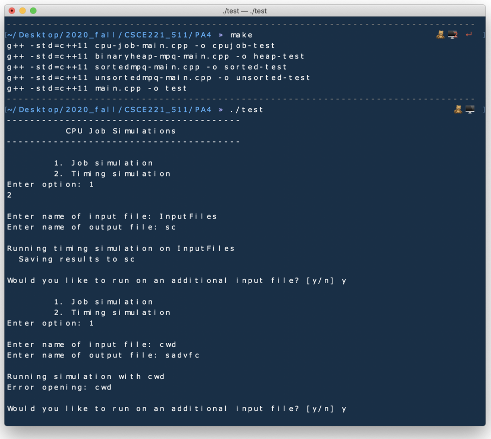
A. Sorted MPQ(Linkedlist): O(n) — Since we need to make sure when we insert the data into the linkedlist, we have to make sure the order. So everytime we insert a data, we will go over every element to check order and put the data at a right position.

B. Unsorted MPQ(vector): O(1) – We don't havr to check the order of elements so just oush the data at the last position of vector.

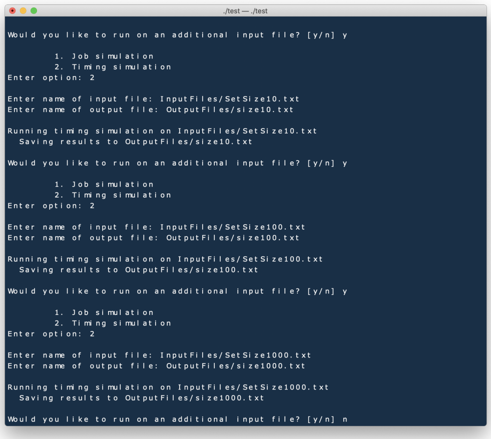
C. Binary heap: O(log n) — Since the order is matter in this data structure, we need to check every elements. However, since we use the binary heap method, so we can just check O(log n) times to make sure thr order. It's similar to the binary tree. So in this function, for insert the data into the vector, the running time id O(1), and we check the order will take O(log n) time. SO it's O(1) + O(log n) = O(log n).

v. For the other two function in the binary heap, the complexicity is O(log n) for both of them. These two functions will go over the whole vector by using the similar way of binary tree. So the running time would usually be the "height" of the tree which will be log n.

3. Give the best, worst, and average case input examples and runtime (bigO) for each implementation on sorting a given array.
   (a) Best case: O(n log n) — when the order of original array is in order, Ex: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
   (b) Worst case: O(n log n) — when the order of original array is reverse order, Ex: 10, 9, 8, 7, 6, 5, 4, 3, 2, 1
   (c) Average case: O(n log n) — when the order of original array is neither in order nor reverse order, Ex: 3, 2, 6, 4, 7, 5, 9, 1, 10, 8
   (d) Time complexity of heapify is O(Logn). Time complexity of createAndBuildHeap() is O(n). And in this case we are gonna use Minimum element as the "root".
4. C++ organization and implementation of the problem solution.
   (a) See the mimir submission.
5. Provide graphs and data tables of your CPU job simulation results. (Only input sizes of 4, 10, 100, and 1,000 required)
   (a) Screenshot of terminal:

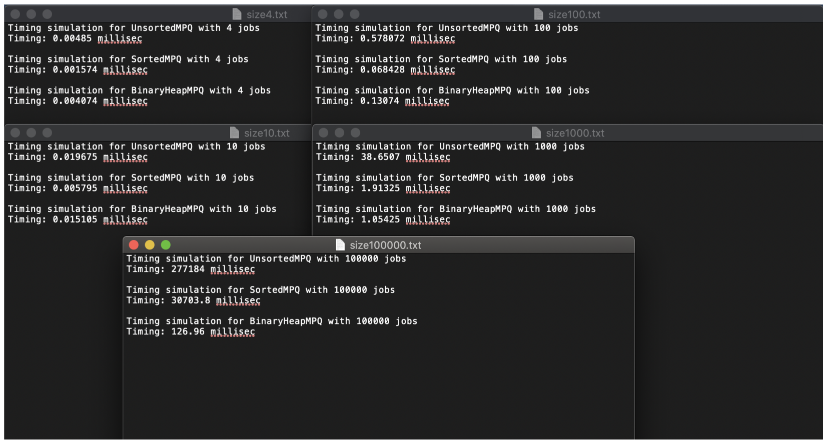


```
./test — ./test
--------------------------------------------------------------------------------
~/Desktop/2020_fall/CSCE221_511/PA4 » make
g++ -std=c++11 cpu-job-main.cpp -o cpujob-test
g++ -std=c++11 binaryheap-mpq-main.cpp -o heap-test
g++ -std=c++11 sortedmpq-main.cpp -o sorted-test
g++ -std=c++11 unsortedmpq-main.cpp -o unsorted-test
g++ -std=c++11 main.cpp -o test
--------------------------------------------------------------------------------
~/Desktop/2020_fall/CSCE221_511/PA4 » ./test
----------------------------------------
          CPU Job Simulations
----------------------------------------


        1. Job simulation
        2. Timing simulation
Enter option: 1
2

Enter name of input file: InputFiles
Enter name of output file: sc

Running timing simulation on InputFiles
  Saving results to sc

Would you like to run on an additional input file? [y/n] y

        1. Job simulation
        2. Timing simulation
Enter option: 1

Enter name of input file: cwd
Enter name of output file: sadvfc

Running simulation with cwd
Error opening: cwd

Would you like to run on an additional input file? [y/n] y
```



```
./test — ./test

Would you like to run on an additional input file? [y/n] y

        1. Job simulation
        2. Timing simulation
Enter option: 2

Enter name of input file: InputFiles/SetSize10.txt
Enter name of output file: OutputFiles/size10.txt

Running timing simulation on InputFiles/SetSize10.txt
  Saving results to OutputFiles/size10.txt

Would you like to run on an additional input file? [y/n] y

        1. Job simulation
        2. Timing simulation
Enter option: 2

Enter name of input file: InputFiles/SetSize100.txt
Enter name of output file: OutputFiles/size100.txt

Running timing simulation on InputFiles/SetSize100.txt
  Saving results to OutputFiles/size100.txt

Would you like to run on an additional input file? [y/n] y

        1. Job simulation
        2. Timing simulation
Enter option: 2

Enter name of input file: InputFiles/SetSize1000.txt
Enter name of output file: OutputFiles/size1000.txt

Running timing simulation on InputFiles/SetSize1000.txt
  Saving results to OutputFiles/size1000.txt

Would you like to run on an additional input file? [y/n] n
```

      i.
   (b) data table and plot of results:



```
size4.txt
Timing simulation for UnsortedMPQ with 4 jobs
Timing: 0.00485 millisec

Timing simulation for SortedMPQ with 4 jobs
Timing: 0.001574 millisec

Timing simulation for BinaryHeapMPQ with 4 jobs
Timing: 0.004074 millisec
```

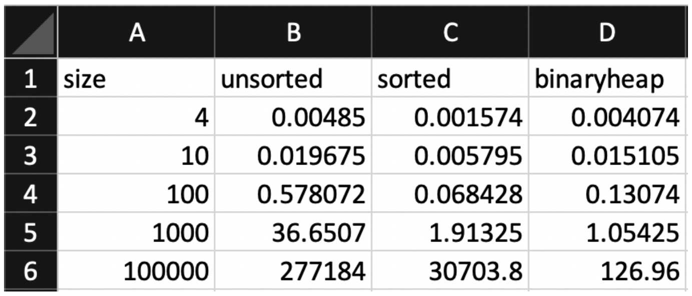
```
size100.txt
Timing simulation for UnsortedMPQ with 100 jobs
Timing: 0.578072 millisec

Timing simulation for SortedMPQ with 100 jobs
Timing: 0.068428 millisec

Timing simulation for BinaryHeapMPQ with 100 jobs
Timing: 0.13074 millisec
```

```
size10.txt
Timing simulation for UnsortedMPQ with 10 jobs
Timing: 0.019675 millisec

Timing simulation for SortedMPQ with 10 jobs
Timing: 0.005795 millisec

Timing simulation for BinaryHeapMPQ with 10 jobs
Timing: 0.015105 millisec
```

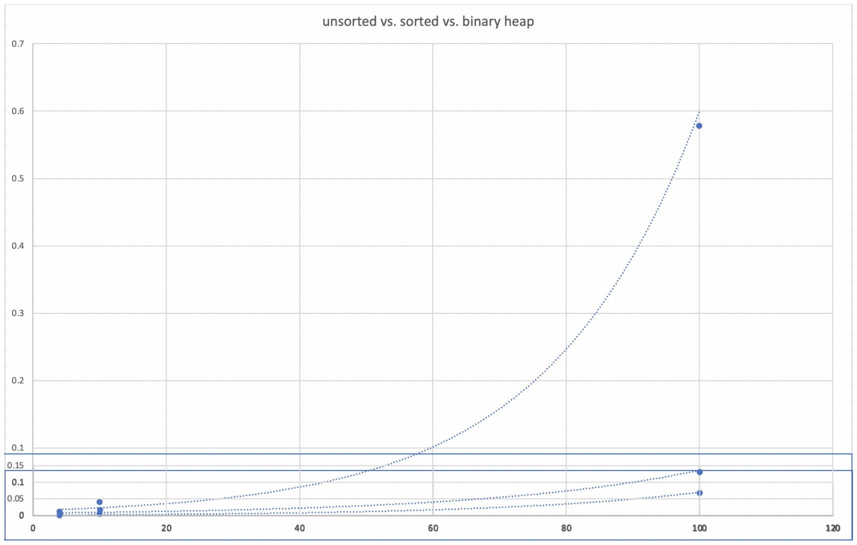
```
size1000.txt
Timing simulation for UnsortedMPQ with 1000 jobs
Timing: 38.6507 millisec

Timing simulation for SortedMPQ with 1000 jobs
Timing: 1.91325 millisec

Timing simulation for BinaryHeapMPQ with 1000 jobs
Timing: 1.05425 millisec
```

```
size100000.txt
Timing simulation for UnsortedMPQ with 100000 jobs
Timing: 277184 millisec

Timing simulation for SortedMPQ with 100000 jobs
Timing: 30703.8 millisec

Timing simulation for BinaryHeapMPQ with 100000 jobs
Timing: 126.96 millisec
```

      i.

ii. 

	A	B	C	D
1	size	unsorted	sorted	binaryheap
2	4	0.00485	0.001574	0.004074
3	10	0.019675	0.005795	0.015105
4	100	0.578072	0.068428	0.13074
5	1000	36.6507	1.91325	1.05425
6	100000	277184	30703.8	126.96

iii. 

iv. I only used the data set of 4, 10, and 100 to plot since that would be more obvious.

v. From the plot, it is easy to show that the binary heap will be more efficiency than sorted than unsorted.

6. A user guide description how to navigate your program with the instructions how to:

(a) compile the program: makefile file

```
all: cpujob-test heap-test sorted-test unsorted-test test

cpujob-test: cpu-job.h
        g++ -std=c++11 cpu-job-main.cpp -o cpujob-test
heap-test: MPQ.h BinaryHeapMPQ.h BinaryHeap.h
        g++ -std=c++11 binaryheap-mpq-main.cpp -o heap-test
sorted-test: MPQ.h SortedMPQ.h
        g++ -std=c++11 sortedmpq-main.cpp -o sorted-test
unsorted-test: MPQ.h UnsortedMPQ.h
        g++ -std=c++11 unsortedmpq-main.cpp -o unsorted-test
test: MPQ.h UnsortedMPQ.h SortedMPQ.h BinaryHeapMPQ.h BinaryHeap.h cpu-job.h
        g++ -std=c++11 main.cpp -o test
clean:
        rm *test OutputFiles/*
run: test
        ./test
```

(b) compile the program: specify the directory and file names, etc.

```
» cd ./PA_4
» make clean
» make
» ./test
```

7. Specifications and description of input and output formats and files
   (a) The type of files: keyboard, text files, etc (if applicable).
      i. Input file: a list of data has three columns which are CPU ID, length and prior.
   (b) A file input format: when a program requires a sequence of input items, specify the number of items per line or a line termination. Provide a sample of a required input format.
      i. Input file: input file should be an .txt file which have the format like this:
         A. Test file for "SetSize10"
         B.
```
7       7       8
1       2       11
6       2       15
3       8       3
2       10      -16
8       3       -3
4       7       -16
9       9       -3
10      9       -3
5       9       11
```
   (c) Discuss possible cases when your program could crash because of incorrect input (a wrong file name, strings instead of a number, or such cases when the program expects 10 items to read and it finds only 9.)
      i. none.
      ii. no special case.

8. Provide types of exceptions and their purpose in your program.
   (a) logical exceptions (such as deletion of an item from an empty container, etc.).
      i. No logical error has been found in testing
   (b) runtime exception (such as division by 0, etc.)
      i. Empty MPQ will be excuted in a different way: throw an exception.

9. Test your program for correctness using valid, invalid, and random inputs (e.g., insertion of an item at the beginning, at the end, or at a random place into a sorted vector). Include evidence of your testing, such as an output file or screen shots with an input and the corresponding output
   (a) no implementation of invalid input.
   (b) All tests passes.