CSCE 221 Cover Page

First Name Aaron Last Name Sanchez UIN 228004762

User Name aaronsanchez01 E-mail address aaronsanchez01@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 on Aggie Honor System Office website: 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/codes 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	Aaron		Sanchez	Date	11/05/2020

Report (20 points)

1. Describe your approach for each MPQ implementation.

In order to implement each of the different MPQ's, we must look at the data structure that we are using to hold each MPQ. For the Unsorted MPQ we used vectors, so to insert we will simply push the data to the back of vector, but we will have to completely search through the vector in order to find the minimum value. For the sorted MPQ we will use linked lists and the MPQ is actually sorted so in order to insert into the MPQ, we need to check the value to insert across the list and once there is a value that is bigger than the value to insert we will insert this value before the bigger value. For removal of the minimum and finding the minimum the linked list is sorted so we simply need to give the front value or take out the front value. Finally for Binary Heap MPQ this is a combination of a sorted and unsorted MPQ as it has the values in a Binary Heap, however, this Binary Heap makes sure that the top root value is the smallest, and that the parent nodes are simply smaller than the child nodes. To implement insert we need to simply insert the value at the back of the binary heap, and continue to upheap, which is checking if the node is less than its parent node, and switching the node and parent node until the node is no longer smaller than the parent node. For min we simply return the first element of binary heap, and to remove min we switch the first and last element then delete the last element, then we will down heap, which is switching the first node with its child node until this node gets to the point where it is greater than all the nodes above it.

Screenshots of Code running correctly:

```
Enter option: 2

Enter name of input file: SetSize4.txt
Enter name of output file: 4Time.txt

Running timing simulation on SetSize4.txt
  Saving results to 4Time.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: SetSize10.txt
Enter name of output file: 10out.txt

Running timing simulation on SetSize10.txt
  Saving results to 10out.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: SetSize100.txt
Enter name of output file: 100out.txt

Running timing simulation on SetSize100.txt
  Saving results to 100out.txt

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:
1
Enter name of output file: 1

Running simulation with 1
Error opening: 1

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: SetSize1000.txt
Enter name of output file: 1000out.txt

Running timing simulation on SetSize1000.txt
  Saving results to 1000out.txt

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

[aaronsanchez01]@compute ~/CSCE_221/PA/PA4> (01:43:44 11/05/20)
::
```

4Time.txt | 10out.txt | 100out.txt | 1000out.txt | 1000.txt

```
Running simulation for UnsortedMPQ with 1000 jobs
Job 337 with length 1 and priority -20
Job 732 with length 2 and priority -20
Job 221 with length 3 and priority -20
Job 808 with length 4 and priority -20
Job 977 with length 4 and priority -20
Job 729 with length 5 and priority -20
Job 750 with length 5 and priority -20
Job 792 with length 5 and priority -20
Job 441 with length 6 and priority -20
Job 469 with length 6 and priority -20
Job 767 with length 6 and priority -20
Job 719 with length 7 and priority -20
Job 785 with length 7 and priority -20
Job 697 with length 8 and priority -20
Job 929 with length 8 and priority -20
Job 150 with length 9 and priority -20
Job 913 with length 9 and priority -20
Job 290 with length 10 and priority -20
Job 382 with length 10 and priority -20
Job 485 with length 10 and priority -20
Job 517 with length 10 and priority -20
Job 534 with length 10 and priority -20
Job 414 with length 1 and priority -19
Job 653 with length 2 and priority -19
Job 852 with length 2 and priority -19
Job 932 with length 2 and priority -19
Job 457 with length 3 and priority -19
Job 764 with length 3 and priority -19
Job 79 with length 4 and priority -19
Job 287 with length 4 and priority -19
Job 621 with length 4 and priority -19
Job 369 with length 5 and priority -19
Job 967 with length 5 and priority -19
Job 991 with length 5 and priority -19
Job 77 with length 6 and priority -19
Job 173 with length 7 and priority -19
Job 238 with length 7 and priority -19
Job 395 with length 7 and priority -19
Job 677 with length 7 and priority -19
Job 694 with length 7 and priority -19
Job 987 with length 7 and priority -19
Job 139 with length 9 and priority -19
Job 215 with length 9 and priority -19
Job 385 with length 9 and priority -19
Job 486 with length 10 and priority -19
Job 520 with length 10 and priority -19
Job 986 with length 10 and priority -19
Job 775 with length 1 and priority -18
Job 869 with length 1 and priority -18
Job 133 with length 2 and priority -18
Job 629 with length 2 and priority -18
Job 734 with length 2 and priority -18
Job 301 with length 3 and priority -18
Job 419 with length 3 and priority -18
Job 754 with length 3 and priority -18
Job 321 with length 4 and priority -18
Job 434 with length 5 and priority -18
Job 928 with length 5 and priority -18
Job 798 with length 6 and priority -18
Job 879 with length 6 and priority -18
Job 587 with length 7 and priority -18
Job 744 with length 7 and priority -18
Job 59 with length 8 and priority -18
Job 280 with length 8 and priority -18
Job 686 with length 9 and priority -18
Job 699 with length 9 and priority -18
```

2. Show the time complexity analysis for each of the MPQ functions (remove min(), min(), and insert()).

	min	Remove min	insert
Unsorted MPQ	O(n)	O(n)	O(1)
Sorted MPQ	O(1)	O(1)	O(n)
Binary Heap MPQ	O(1)	O(logn)	O(logn)

Unsorted MPQ min and remove min have to search through the vector for the smallest element, insert simply pushes to the back.

Sorted MPQ, insert searches through the linked list to find the correct spot to insert the value. Min and remove min simply take out the first element of the list.

BinaryHeap MPQ min simply returns the first element. Insert puts the value into the back of the BinaryHeap, but will go through the parent nodes, constantly dividing the size by 2 to check the parent for swapping. Remove min takes the first element out, then moves the last element up to the front divides the list by 2 checking if the node needs to go to the left or right and if it is smaller than these.

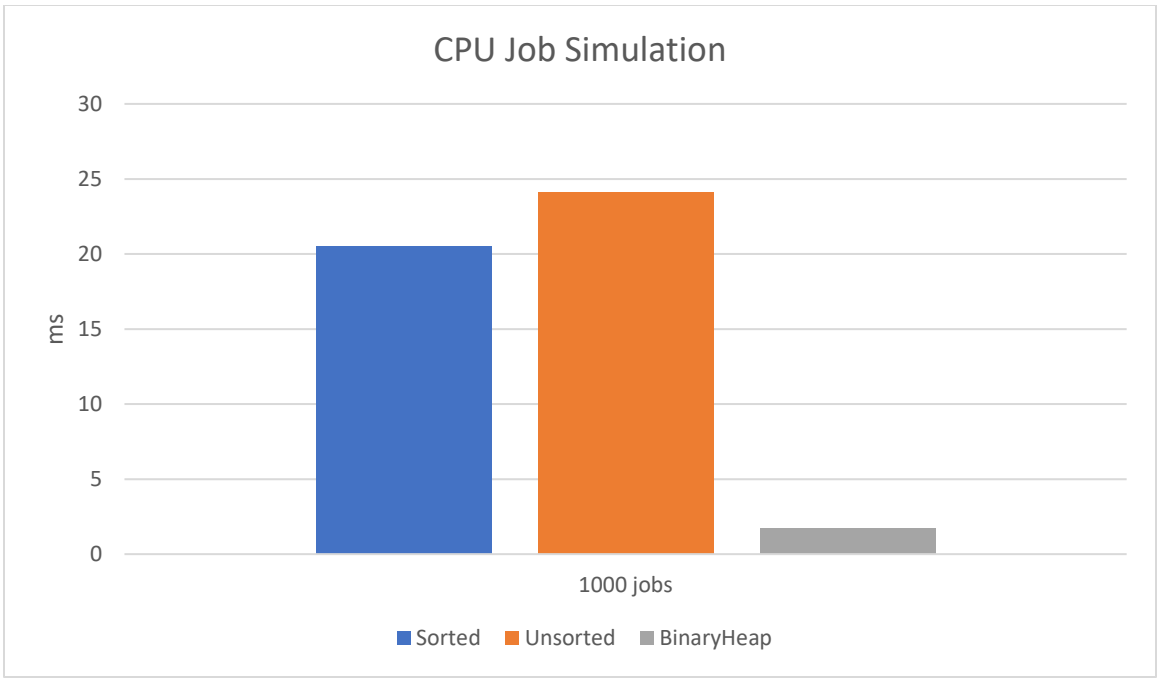
3. Give the best, worst, and average case input examples and runtime (bigO) for each implementation on sorting a given array.

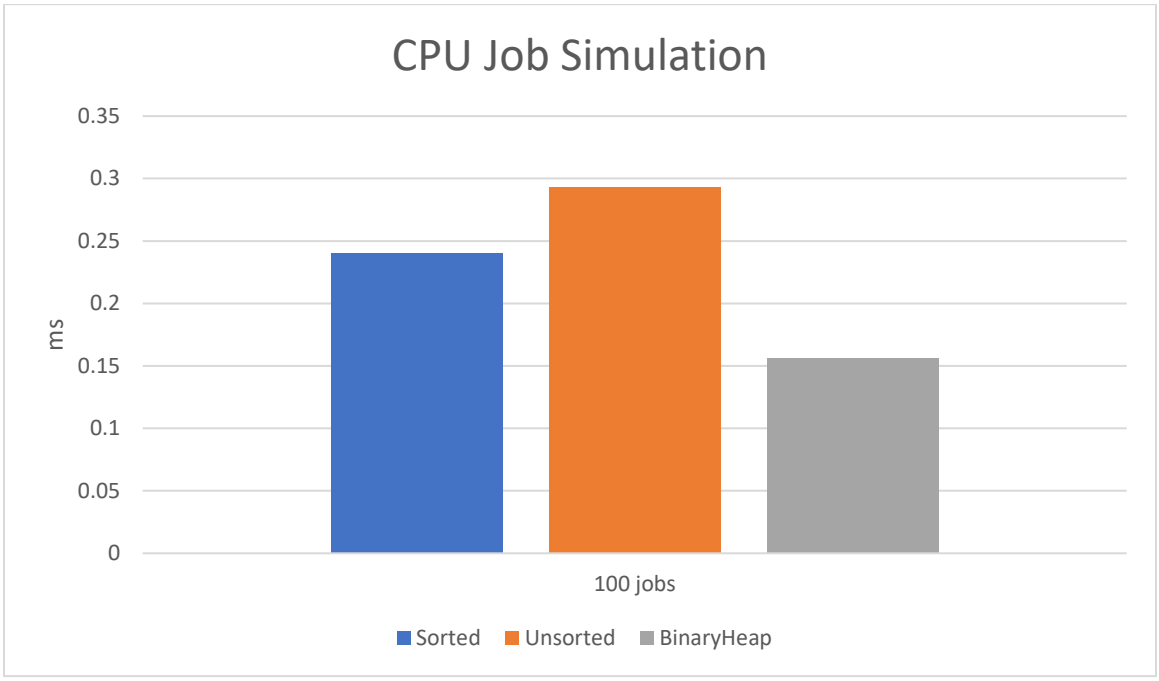
For the best, worst and average case for Sorted MPQ min and remove min bigO(1) constantly, as these are simply taking the first element of the vector. Insert best case happens when either the vector is empty or the element to be inserted is the smallest making it bigO(1), average case is whenever inserted value is around the middle of the vector, making bigO(n), and worst case is whenever the value is bigger than any other in the vector which is still bigO(n).

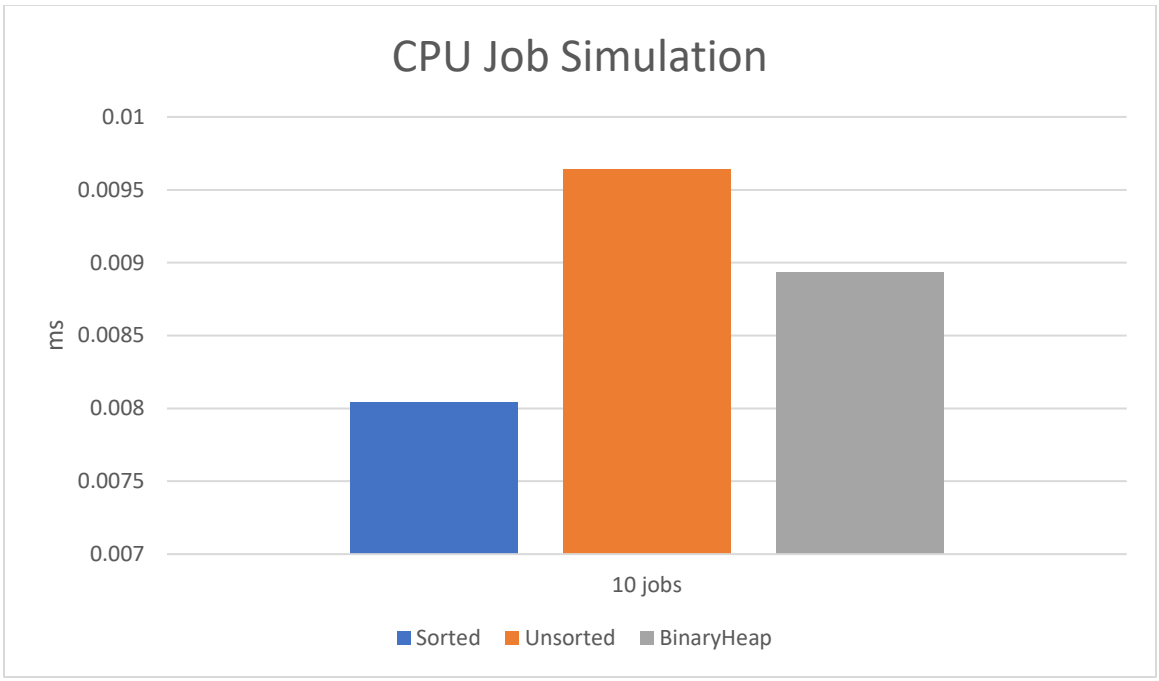
For unsorted MPQ the best case for min and remover min happens whenever the list is 1 element long, making the bigO(1). Worst and average case is whenever the minimum is anywhere in the list, as you have to compare all the values anyways, making bigO(n). Insert best, average, and worst are all bigO(1), as you simply push to the back of the vector.

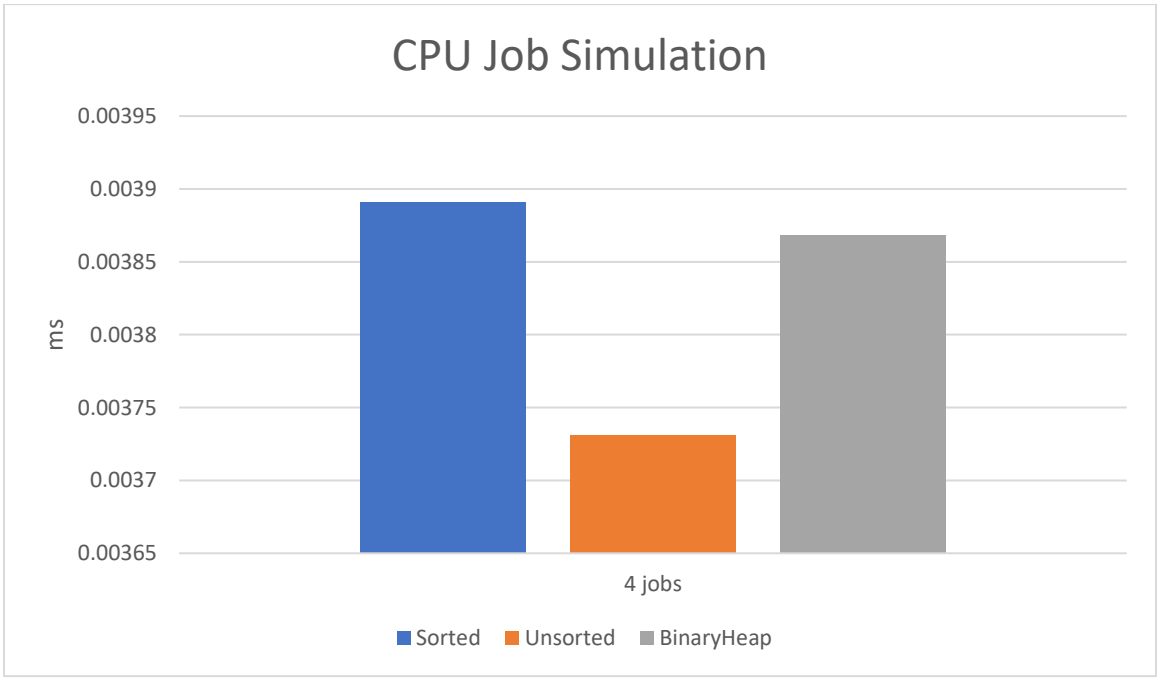
For BinaryHeap MPQ min best, average, and worst cases all are bigO(1), as this is simply returning the first element. Remove min best case is whenever the list is all the same value and we simply switch a value and it is bigO(1). Average case is switching first and last element, with last element not being the largest value, then removing and going through the downheap, which leads to a bigO(logn). Worst is whenever we switch first and last element, but last is the largest element, this results in downheap all the way to the bottom, resulting in bigO(logn). Insert best case is inserting into a list that is all the same value or inserting a larger value than the parent values, which results in no upheap, bigO(1). Average case is inserting a value that is not the minimum and we end up upheaping a couple times, resulting in bigO(logn). Worst case is inserting the minimum, where we have to upheap all the way up the BinaryHeap, resulting in bigO(logn).

4. Provide graphs and data tables of your CPU job simulation results. (Only input sizes of 4, 10, 100, and 1,000 required).


CPU Job Simulation
30
25
20
15
10
5
0
ms
1000 jobs
Sorted
Unsorted
BinaryHeap


CPU Job Simulation
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
ms
100 jobs
Sorted
Unsorted
BinaryHeap


CPU Job Simulation
0.01
0.0095
0.009
0.0085
0.008
0.0075
0.007
ms
10 jobs
Sorted
Unsorted
BinaryHeap


CPU Job Simulation
0.00395
0.0039
0.00385
0.0038
0.00375
0.0037
0.00365
ms
4 jobs
Sorted
Unsorted
BinaryHeap