

CSC220.01 Duc Ta

Edreece Afridi 920159263

Kayvaun Khoshkhou 920357344

Assignment Number: 4

Assignment Due: 7/26/20 at : 11:55 P.M.

Part A

Handwritten Parts:

Part A

Array of Integers: 9 4 9 8 3 5 6 2 9 7 5 1

Smallest → Final Position

| | | | | | | | | | | | | |
|----|----|----|----|----|----|----|---|----|----|----|----|----|
| 1) | 9. | 4 | 9 | 8 | 3 | 5 | 6 | 2 | 9 | 7 | 5 | 1. |
| | 1 | 4. | 9 | 8 | 3 | 5 | 6 | 2. | 9 | 7 | 5 | 9 |
| | ✓ | 2 | 9. | 8 | 3. | 5 | 6 | 4 | 9 | 7 | 5 | 9 |
| | | ✓ | 3 | 8. | 9 | 5 | 6 | 4. | 9 | 7 | 5 | 9 |
| | | | ✓ | 4 | 9. | 5. | 6 | 8 | 9 | 7 | 5 | 9 |
| | | | | ✓ | 5 | 9. | 6 | 8 | 9 | 7 | 5. | 9 |
| | | | | | ✓ | 5 | 6 | 8. | 9 | 7. | 9 | 9 |
| | | | | | | ✓ | 6 | 7 | 9. | 8. | 9 | 9 |
| | | | | | | | ✓ | 7 | 8 | 9 | 9 | 9 |
| | | | | | | | | ✓ | 8 | 9 | 9 | 9 |
| ✓ | 1 | 2 | 3 | 4 | 5 | 5 | 6 | 7 | 8 | 9 | 9 | 9 |

Key: .. to swap, ✓ done

Sorted [sublist]

| | | | | | | | | | | | | |
|----|-----|-----|----|-----|-----|-----|-----|-----|---|-----|-----|-----|
| 2) | [9] | 4 | 9 | 8 | 3 | 5 | 6 | 2 | 9 | 7 | 5 | 1 |
| | [9. | 4.] | | | | | | | | | | |
| | [4 | 9 | 9] | | | | | | | | | |
| | [4 | 9. | 9 | 8.] | | | | | | | | |
| | [4. | 8 | 9 | 9 | 3.] | | | | | | | |
| | [3 | 4 | 8. | 9 | 9 | 5.] | | | | | | |
| | [3 | 4. | 5 | 8. | 9 | 9 | 6.] | | | | | |
| | [3. | 4 | 5 | 6 | 8 | 9 | 9 | 2.] | | | | |
| | [2 | 3 | 4 | 5 | 6 | 8. | 9 | 9 | 9 | 7.] | | |
| | [2 | 3 | 4 | 5 | 6. | 7 | 8 | 9 | 9 | 9 | 5.] | |
| | [2. | 3 | 4 | 5 | 5 | 6 | 7 | 8 | 9 | 9 | 9 | 1.] |
| ✓ | [1 | 2 | 3 | 4 | 5 | 5 | 6 | 7 | 8 | 9 | 9 | 9] |

Key: .. to be sorted, ✓ done

shell sort

| | | | | | | | | | | | | | |
|----|-----|----|----|----|----|----|----|----|----|----|----|----|--------|
| 3) | 9. | 4 | 7 | 8 | 3 | 5 | 6. | 2 | 9 | 7 | 5 | 1 | 12/2=6 |
| | 6 | | | | | | 9 | | | | | | |
| | | 2. | | | | | | 4. | | | | | |
| | | | 9. | | | | | | 9. | | | | |
| | | | | 7. | | | | | | 8. | | | |
| | | | | | 3. | 2 | | | | | 5. | | |
| | | | | | | 1. | | | | | | 5. | |
| | 6. | 2 | 9 | 7 | 3 | 1 | 9 | 4 | 9 | 8 | 5 | 5 | 6/2=3 |
| | 6. | | | 7. | | | 8. | | | 9. | | | |
| | | 2. | | | 3. | | | 4. | | | 5. | | |
| | | | 1. | | | 5. | | | 9. | | | 9. | |
| | 6 | 2 | 1 | 7 | 3 | 5 | 8 | 4 | 9 | 9 | 5 | 9 | 3/2=1 |
| | [6] | | | | | | | | | | | | |
| | [2 | 6] | | | | | | | | | | | |
| | [1 | 2 | 6] | | | | | | | | | | |
| | [1 | 2 | 6 | 7] | | | | | | | | | |
| | [1 | 2 | 3 | 6 | 7] | | | | | | | | |
| | [1 | 2 | 3 | 5 | 6 | 7] | | | | | | | |
| | [1 | 2 | 3 | 5 | 6 | 7 | 8] | | | | | | |
| | [1 | 2 | 3 | 4 | 5 | 6 | 7 | 8] | | | | | |
| | [1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9] | | | | |
| | [1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 9] | | | |
| | [1 | 2 | 3 | 4 | 5 | 5 | 6 | 7 | 8 | 9 | 9] | | |
| ✓ | [1 | 2 | 3 | 4 | 5 | 5 | 6 | 7 | 8 | 9 | 9 | 9] | |

Code is included in Zip file. Screenshots of output:

```
Original Array : 9 4 9 8 3 5 6 2 9 7 5 1
Selection sort :
1 4 9 8 3 5 6 2 9 7 5 9
1 2 9 8 3 5 6 4 9 7 5 9
1 2 3 8 9 5 6 4 9 7 5 9
1 2 3 4 9 5 6 8 9 7 5 9
1 2 3 4 5 9 6 8 9 7 5 9
1 2 3 4 5 5 6 8 9 7 9 9
1 2 3 4 5 5 6 8 9 7 9 9
1 2 3 4 5 5 6 7 9 8 9 9
1 2 3 4 5 5 6 7 8 9 9 9
1 2 3 4 5 5 6 7 8 9 9 9
1 2 3 4 5 5 6 7 8 9 9 9
Original Array : 9 4 9 8 3 5 6 2 9 7 5 1
Bubble sort :
4 9 8 3 5 6 2 9 7 5 1 9
4 8 3 5 6 2 9 7 5 1 9 9
4 3 5 6 2 8 7 5 1 9 9 9
3 4 5 2 6 7 5 1 8 9 9 9
3 4 2 5 6 5 1 7 8 9 9 9
3 2 4 5 5 1 6 7 8 9 9 9
2 3 4 5 1 5 6 7 8 9 9 9
2 3 4 1 5 5 6 7 8 9 9 9
2 3 1 4 5 5 6 7 8 9 9 9
2 1 3 4 5 5 6 7 8 9 9 9
1 2 3 4 5 5 6 7 8 9 9 9
1 2 3 4 5 5 6 7 8 9 9 9
```

```
Original Array : 9 4 9 8 3 5 6 2 9 7 5 1
Shell sort :
6 4 9 8 3 5 9 2 9 7 5 1
6 2 9 8 3 5 9 4 9 7 5 1
6 2 9 8 3 5 9 4 9 7 5 1
6 2 9 7 3 5 9 4 9 8 5 1
6 2 9 7 3 5 9 4 9 8 5 1
6 2 9 7 3 1 9 4 9 8 5 5
6 2 9 7 3 1 9 4 9 8 5 5
6 2 9 7 3 1 9 4 9 8 5 5
6 2 1 7 3 9 9 4 9 8 5 5
6 2 1 7 3 9 9 4 9 8 5 5
6 2 1 7 3 9 9 4 9 8 5 5
6 2 1 7 3 9 9 4 9 8 5 5
6 2 1 7 3 9 8 4 9 9 5 5
6 2 1 7 3 5 8 4 9 9 5 9
2 6 1 7 3 5 8 4 9 9 5 9
1 2 6 7 3 5 8 4 9 9 5 9
1 2 6 7 3 5 8 4 9 9 5 9
1 2 3 6 7 5 8 4 9 9 5 9
1 2 3 5 6 7 8 4 9 9 5 9
1 2 3 5 6 7 8 4 9 9 5 9
1 2 3 4 5 6 7 8 9 9 5 9
1 2 3 4 5 6 7 8 9 9 5 9
1 2 3 4 5 5 6 7 8 9 9 9
1 2 3 4 5 5 6 7 8 9 9 9
BUILD SUCCESSFUL (total time: 1 second)
```


Part B

Part B

1. Let's say we have an array of size n .
In order to find the largest entry in the unsorted array of size n as efficiently as possible then we would want to opt towards using algorithm A over algorithm B. This is because algorithm A searches the array linearly and records the highest value entry seen.

Algorithm A Time Complexity = $O(n)$

We're told algorithm B sorts in descending order and records the first value stored as the highest value.

Algorithm B Time Complexity = $O(n \log(n))$

Algorithm B is not as efficient as A because it works backwards down the list compared to the linear analysis that A sorts in.

Part C: 1. --- 3 points --- After each of the following statements executes, what are the contents of the queue? Please explain.

```
QueueInterface myQueue = new LinkedList<>();
```

```
myQueue.enqueue("Jane");    //myQueue = [Jane]
```

```
myQueue.enqueue("Jess");    //myQueue = [Jane, Jess]
```

```
myQueue.enqueue("Jon");    //myQueue = [Jane, Jess, Jon]
```

```

myQueue.enqueue(myQueue.dequeue()); //myQueue = [Jess, Jon, Jane]

myQueue.enqueue(myQueue.getFront()); // [Jess] //myQueue = [Jess, Jon, Jane, Jess]

myQueue.enqueue("Jim"); //myQueue = [Jess, Jon, Jane, Jess, Jim]

String name = myQueue.dequeue(); // name = Jess

myQueue.enqueue(myQueue.getFront()); //myQueue = [Jon, Jim, Jane, Jess, Jon]

```

2. --- 3 points --- After each of the following statements executes, what are the contents of the deque? Please explain.

```

DequeInterface myDeque = new LinkedDeque<>();

myDeque.addToFront("Jim"); //myDeque = [Jim]

myDeque.addToFront("Jess"); //myDeque = [Jess, Jim]

myDeque.addToBack("Jen"); //myDeque = [Jess, Jim, Jen]

myDeque.addToBack("Josh"); //myDeque = [Jess, Jim, Jen, Josh]

String name = myDeque.removeFront(); //name = Jess //myDeque = [Jim, Jen, Josh]

myDeque.addToBack(name); //myDeque = [Jim, Jen, Josh, Jess]

myDeque.addToBack(myDeque.getFront()); //myDeque = [Jim, Jen, Josh, Jess, Jim]

myDeque.addToFront(myDeque.removeBack()); //myDeque = [Jim, Jim, Jen, Josh, Jess]

myDeque.addToFront(myDeque.getBack()); //myDeque = [Jess, Jim, Jim, Jen, Josh, Jess]

```

3. --- 3 points --- After each of the following statements executes, what are the contents of the priority queue? Please explain.

```
PriorityQueueInterface myPriorityQueue = new LinkedPriorityQueue<>();  
myPriorityQueue.add("Jim"); //myPriorityQueue = [Jim]  
myPriorityQueue.add("Josh"); //myPriorityQueue = [Jim, Josh]  
myPriorityQueue.add("Jon"); //myPriorityQueue = [Jim, Jon, Josh]  
myPriorityQueue.add("Jane"); //myPriorityQueue = [Jane, Jim, Jon, Josh]  
String name = myPriorityQueue.remove(); //name = Jane ////myPriorityQueue = [Jim, Jon,  
Josh]  
myPriorityQueue.add(name); //myPriorityQueue = [Jane, Jim, Jon, Josh]  
myPriorityQueue.add(myPriorityQueue.peek()); //Jane //myPriorityQueue = [Jane, Jane, Jim,  
Jon, Josh]  
myPriorityQueue.add("Jose"); //myPriorityQueue = [Jane, Jane, Jim, Jon, Jose, Josh]  
myPriorityQueue.remove(); //myPriorityQueue = [Jane, Jim, Jon, Jose, Josh]
```