

**Course:** CSC 220.02

**Student:** Kullathon “Mos” Sitthisarnwattanachai, **SFSU ID:** 921425216

**Teammate:** n/a, **SFSU ID:** n/a

**Assignment Number:** 04

**Assignment Due Date & Time:** 04-19-2020 at 11:55 PM

---

## **PART A – Introduction to Sorting, 9 points**

Please use this array of integers for the A.1, A.2, and A.3 problems:

**9 1 2 4 3 5 6 2 9 8 5 7**

- Use pen & paper to show our work and answers. We can scan/snapshot our work and include the images in our report.
- Use code to demonstrate our approaches and solutions. Submit code and include screenshots of output in our report.
- Each of these problems (A.1, A.2, and A.3) is worth **3 points**.

1. Show the contents of the array each time a **selection sort** changes it while sorting the array into **ascending order**.

- See assignment04PartA/A1 in the attached files for the code.

Below is the screenshot of the output demonstrating the contents of the array each time the algorithm sorts the array. Following it is the handwritten solution.

```
"C:\Program Files\Java\jdk-15.0.2\bin\java.exe"
```

```
[9, 1, 2, 4, 3, 5, 6, 2, 9, 8, 5, 7]
```

```
1: [1, 9, 2, 4, 3, 5, 6, 2, 9, 8, 5, 7]
```

```
2: [1, 2, 9, 4, 3, 5, 6, 2, 9, 8, 5, 7]
```

```
3: [1, 2, 2, 4, 3, 5, 6, 9, 9, 8, 5, 7]
```

```
4: [1, 2, 2, 3, 4, 5, 6, 9, 9, 8, 5, 7]
```

```
5: [1, 2, 2, 3, 4, 5, 6, 9, 9, 8, 5, 7]
```

```
6: [1, 2, 2, 3, 4, 5, 6, 9, 9, 8, 5, 7]
```

```
7: [1, 2, 2, 3, 4, 5, 5, 9, 9, 8, 6, 7]
```

```
8: [1, 2, 2, 3, 4, 5, 5, 6, 9, 8, 9, 7]
```

```
9: [1, 2, 2, 3, 4, 5, 5, 6, 7, 8, 9, 9]
```

```
10: [1, 2, 2, 3, 4, 5, 5, 6, 7, 8, 9, 9]
```

```
11: [1, 2, 2, 3, 4, 5, 5, 6, 7, 8, 9, 9]
```

```
[1, 2, 2, 3, 4, 5, 5, 6, 7, 8, 9, 9]
```

```
Process finished with exit code 0
```

```
9. 1. 2 4 3 5 6 2 9 8 5 7
① 1 9. 2. 4 3 5 6 2 9 8 5 7
② 2 9. 4 3 5 6 2. 9 8 5 7
③ 2 4. 3. 5 6 9 9 8 5 7
④ 3 4. 5. 6 9 9 8 5 7
⑤ 4 5. 6 9 9 8 5 7
⑥ 5 6. 9 9 8 5. 7
⑦ 5 9. 9 8 6. 7
⑧ 6 9. 8 9 7.
⑨ 7 8. 9 9
⑩ 8 9. 9
⑪ 9 9
1 2 2 3 4 5 5 6 7 8 9 9
```

2. Show the contents of the array each time an **insertion sort** changes it while sorting the array into **ascending order**.

- See assignment04PartA/A2 in the attached files for the code.

Below is the screenshot of the output demonstrating the contents of the array each time the algorithm sorts the array. Following it is the handwritten solution.

```
"C:\Program Files\Java\jdk-15.0.2\bin\java.exe"  
[9, 1, 2, 4, 3, 5, 6, 2, 9, 8, 5, 7]  
 1: [1, 9, 2, 4, 3, 5, 6, 2, 9, 8, 5, 7]  
 2: [1, 2, 9, 4, 3, 5, 6, 2, 9, 8, 5, 7]  
 3: [1, 2, 4, 9, 3, 5, 6, 2, 9, 8, 5, 7]  
 4: [1, 2, 3, 4, 9, 5, 6, 2, 9, 8, 5, 7]  
 5: [1, 2, 3, 4, 5, 9, 6, 2, 9, 8, 5, 7]  
 6: [1, 2, 3, 4, 5, 6, 9, 2, 9, 8, 5, 7]  
 7: [1, 2, 2, 3, 4, 5, 6, 9, 9, 8, 5, 7]  
 8: [1, 2, 2, 3, 4, 5, 6, 9, 9, 8, 5, 7]  
 9: [1, 2, 2, 3, 4, 5, 6, 8, 9, 9, 5, 7]  
10: [1, 2, 2, 3, 4, 5, 5, 6, 8, 9, 9, 7]  
11: [1, 2, 2, 3, 4, 5, 5, 6, 7, 8, 9, 9]  
[1, 2, 2, 3, 4, 5, 5, 6, 7, 8, 9, 9]  
  
Process finished with exit code 0
```

	9	1	2	4	3	5	6	2	9	8	5	7
①	9											
②	1	9										
③	1	2	9									
④	1	2	4	9								
⑤	1	2	3	4	9							
⑥	1	2	3	4	5	9						
⑦	1	2	3	4	5	6	9					
⑧	1	2	2	3	4	5	6	9				
⑨	1	2	2	3	4	5	6	7	9			
⑩	1	2	2	3	4	5	6	8	7	9		
⑪	1	2	2	3	4	5	5	6	8	7	9	
	1	2	2	3	4	5	5	6	7	8	9	9

3. Show the contents of the array each time a **Shell sort** changes it while sorting the array into **ascending order**.

- See assignment04PartA/A3 in the attached files for the code.

Below is the screenshot of the output demonstrating the contents of the array

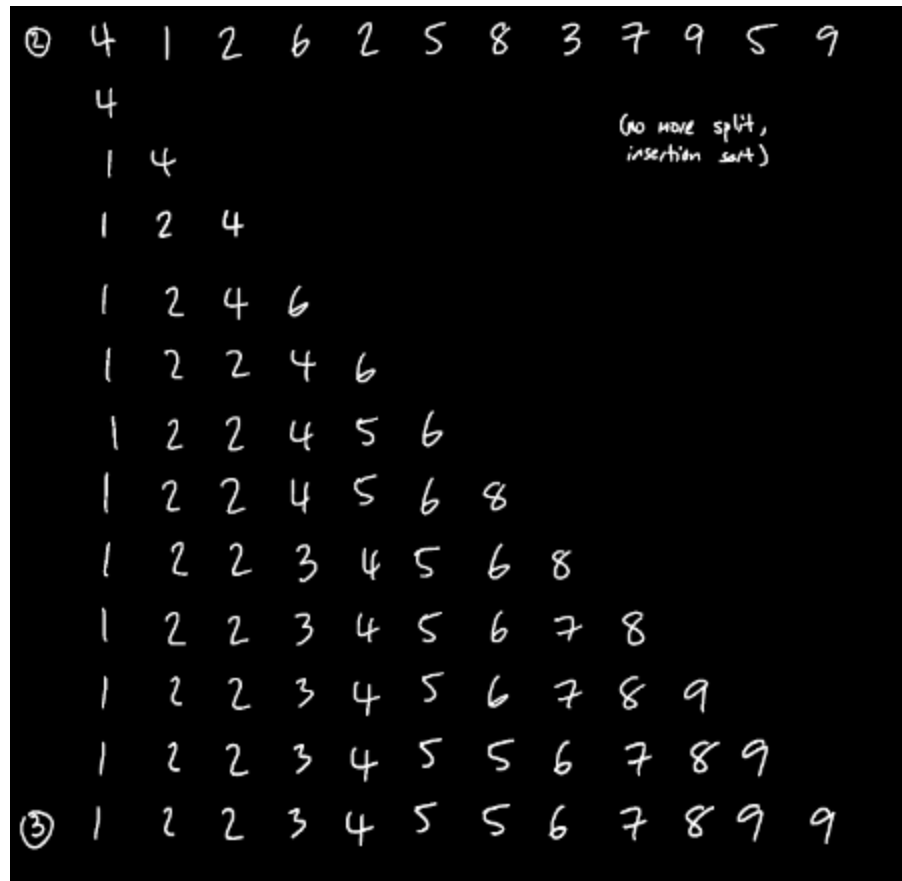
each time the algorithm sorts the array. Following it is the handwritten solution.

```
"C:\Program Files\Java\jdk-15.0.2\bin\java.exe"
[9, 1, 2, 4, 3, 5, 6, 2, 9, 8, 5, 7]
  1: [6, 1, 2, 4, 3, 5, 9, 2, 9, 8, 5, 7]
  2: [4, 1, 2, 6, 2, 5, 8, 3, 7, 9, 5, 9]
  3: [1, 2, 2, 3, 4, 5, 5, 6, 7, 8, 9, 9]
[1, 2, 2, 3, 4, 5, 5, 6, 7, 8, 9, 9]

Process finished with exit code 0
```

	9	1	2	4	3	5	6	2	9	8	5	7
	6	.	.	.	.	.	9	.	.	.	.	.
	.	1	.	.	.	.	.	2	.	.	.	.
	.	.	2	.	.	.	.	.	9	.	.	.
	.	.	.	4	.	.	.	.	.	8	.	.
	.	.	.	.	3	.	.	.	.	.	5	.
	.	.	.	.	.	5	.	.	.	.	.	7
④	6	1	2	4	3	5	9	2	9	8	5	7
	4	.	.	6	.	.	8	.	.	9	.	.
	.	1	.	.	2	.	.	3	.	.	5	.
	.	.	2	.	.	5	.	.	7	.	.	9
⑫	4	1	2	6	2	5	8	3	7	9	5	9

**The sorting continues below.** After the second split, the array is partially sorted. It is sorted once more using insertion sort to obtain the final order.



## PART B – Sorting, 11 points

1. --- 3 points --- Suppose we want to find the largest entry in an unsorted array of  $n$  entries. Algorithm A searches the entire array sequentially and records the largest entry seen so far. Algorithm B sorts the array into descending order and then reports the first entry as the largest. Compare the time efficiency of the two approaches. *Coding is not required but highly recommended.*

- See assignment04PartA/B1 in the attached files for the time comparison.

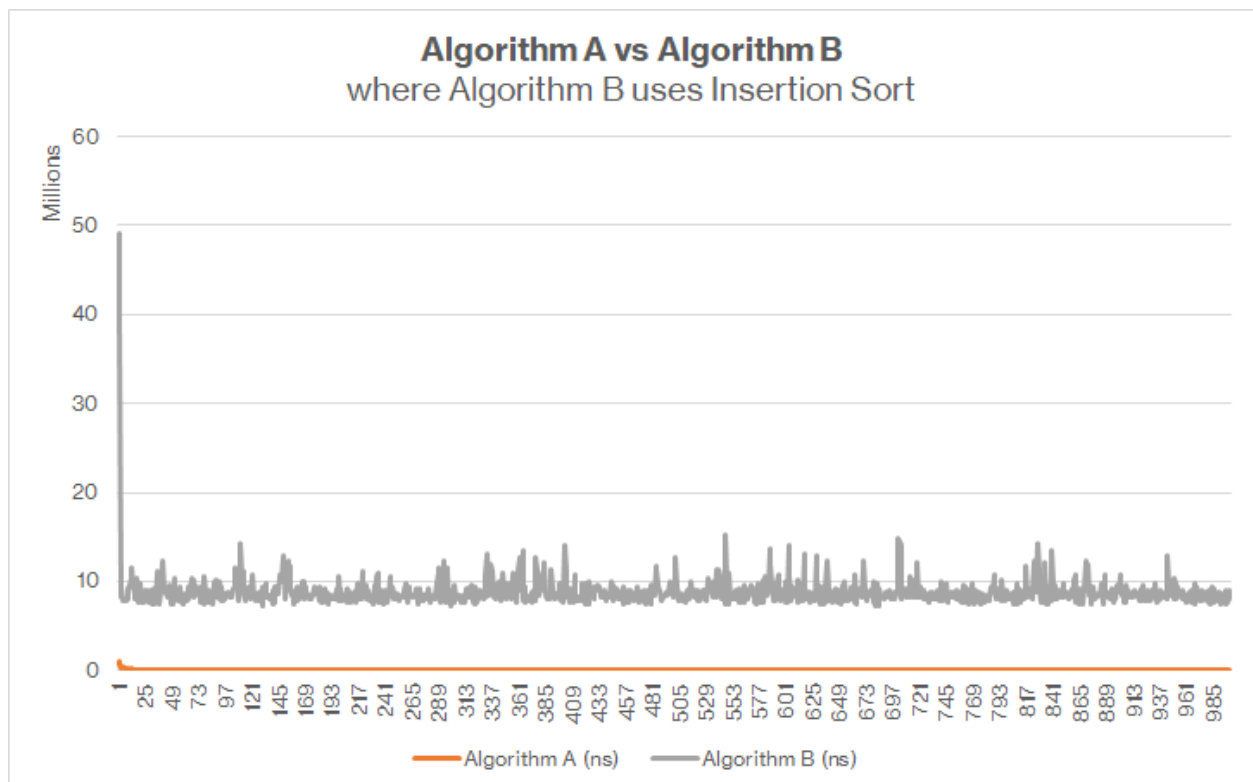
By observation, we are able to see that Algorithm A is much faster than those of Algorithm B. This is because Algorithm B makes sure that all of the elements in

an array are sorted, which means more comparisons are being made. While Algorithm A compares the value of each element exactly once against its highest recorded value.

Below is the graph of the time taken by Algorithm A and Algorithm B to sort an array of 10,000 randomly-generated integers over 10,000 trials, where the arrays are generated on each trial.

We can see clearly, even with varying generated arrays, that the time taken by Algorithm A is consistently lower than those of Algorithm B.

The scale of the time taken by Algorithm A is so insignificant compared to those of Algorithm B such that the thickness of its plot is larger than the value represented.



2. --- 8 points --- Consider an **n by n** array of integer values. Write an algorithm to sort the rows of the array by their first value.

The starter code for this problem is provided in the **Assignment-04-Code.zip** archive.

Our output must be **identical** to the output to the right.

- See assignment04PartB2 in the attached files.

### **PART C – Queues, Deques, and Priority Queues, 15 points**

1. --- 5 points --- After **each of the following statements executes**, what are the contents of the **queue**? Please explain.

- See assignment04PartC1 in the attached files for the code.

- **QueueInterface<String> myQueue = new LinkedList<>();**

[ ]

Nothing, instantiating queue.

- **myQueue.enqueue("Jane");**

[ Jane ]

Adds "Jane" to the end of the queue.

- **myQueue.enqueue("Jess");**

[ Jane, Jess ]

Adds "Jess" to the end of the queue.

- **myQueue.enqueue("Jon");**



[Jane, Jess, Jon]

Adds "Jon" to the end of the queue.

- **`myQueue.enqueue(myQueue.dequeue());`**

[Jess, Jon, Jane]

Removes "Jane" from the front of the queue. Then, add it to the end.

- **`myQueue.enqueue(myQueue.getFront());`**

[Jess, Jon, Jane, Jess]

Gets "Jess" from the front of the queue. Then, add it to the end.

- **`myQueue.enqueue("Jim");`**

[Jess, Jon, Jane, Jess, Jim]

Adds "Jim" to the end of the queue.

- **`String name = myQueue.dequeue();`**

[Jon, Jane, Jess, Jim]

Removes "Jess" from the front of the queue.

- **`myQueue.enqueue(myQueue.getFront());`**

[Jon, Jane, Jess, Jim, Jon]

Gets "Jon" from the front of the queue. Then, add it to the end.

2. - 5 pts - After each of the following statements executes, what are the contents of the deque? Please explain.

- See assignment04PartC2 in the attached files for the code.

- **DequeInterface<String> myDeque = new LinkedDeque<>();**

[ ]

Nothing, instantiating deque.

- **myDeque.addToFront("Jim");**

[Jim]

Adds "Jim" to the front of the deque.

- **myDeque.addToFront("Jess");**

[Jess, Jim]

Adds "Jess" to the front of the deque.

- **myDeque.addToBack("Jen");**

[Jess, Jim, Jen]

Adds "Jen" to the back of the deque.

- **myDeque.addToBack("Josh");**

[Jess, Jim, Jen, Josh]

Adds "Josh" to the back of the deque.

- **String name = myDeque.removeFront();**

[Jim, Jen, Josh]

Removes “Jess” from the front of the deque.

- **`myDeque.addToBack(name);`**

[Jim, Jen, Josh, Jess]

Adds “Jess” from the stored value to the back of the deque.

- **`myDeque.addToBack(myDeque.getFront());`**

[Jim, Jen, Josh, Jess, Jim]

Gets “Jim” from the front of the deque. Then, add it to the back.

- **`myDeque.addToFront(myDeque.removeBack());`**

[Jim, Jim, Jen, Josh, Jess]

Removes “Jim” from the back of the deque. Then, add it to the front.

- **`myDeque.addToFront(myDeque.getBack());`**

[Jess, Jim, Jim, Jen, Josh, Jess]

Get “Jess” from the front of the deque. Then, add it to the front.

3. - 5 pts - After **each of the following statements executes**, what are the contents of the priority queue? Please explain.

- See assignment04PartC3 in the attached files for the code.
- **`PriorityQueueInterface<String> myPriorityQueue = new  
LinkedPriorityQueue<>();`**

[ ]

Nothing, instantiating queue.

- **myPriorityQueue.add("Jim");**

[Jim]

Adds "Jim" to the queue.

- **myPriorityQueue.add("Josh");**

[Jim, Josh]

Adds "Josh" to the queue. Goes after "Jim" due to alphabetical order.

- **myPriorityQueue.add("Jon");**

[Jim, Jon, Josh]

Adds "Jon" to the queue. Goes after "Jim" and before "Josh" due to alphabetical order.

- **myPriorityQueue.add("Jane");**

[Jane, Jim, Jon, Josh]

Adds "Jane" to the queue. Goes before "Jim" due to alphabetical order.

- **String name = myPriorityQueue.remove();**

[Jim, Jon, Josh]

Removes "Jane" from the queue. Gets removed due to having highest priority in alphabetical order.

- **`myPriorityQueue.add(name);`**

[Jane, Jim, Jon, Josh]

Adds “Jane” from the stored value to the queue. Goes before “Jim” due to alphabetical order.

- **`myPriorityQueue.add(myPriorityQueue.peak());`**

[Jane, Jane, Jim, Jon, Josh]

Gets “Jane” by peeking at the top of the priority queue. Gets returned due to having highest priority in alphabetical order. Then, add it to the queue. Goes next to existing “Jane” due to alphabetical order.

- **`myPriorityQueue.add("Jose");`**

[Jane, Jane, Jim, Jon, Jose, Josh]

Adds “Jose” to the queue. Goes before “Josh” and after “Jim” due to alphabetical order.

- **`myPriorityQueue.remove();`**

[Jane, Jim, Jon, Jose, Josh]

Removes “Jane” from the queue. Gets removed due to having highest priority in alphabetical order.

*It is OK to assume that the alphabetically earliest string has the highest priority.*

## **PART D – Queue and Deque, Circular Doubly Linked Chain, 20 points**

Use a circular doubly linked chain to **implement** the ADT deque.

In a **doubly linked chain**, the first and last nodes each contain one null reference, since the first node has no previous node and the last node has no node after it. In a circular doubly linked chain, the first node references the last node, and the last node references the first. Only one external reference is necessary—a reference to the first node—since we can quickly get to the last node from the first node.

The code for this problem is provided in the **Assignment-04-Code.zip** archive. Our output must be **identical** to the output to the right.

- See assignment04PartD in the attached files.

Note that the order of the output may not match due to a strange race condition between the output streams of standard out and standard error. This behavior may vary between different environments. This output is determined by the provided driver file and cannot be edited as such.

The expected output may be obtained by running the code driver file several times. In the process, you may observe that the output varies between each run despite being a linear program with no randomization element.

*4/24 update: it appears that the issue is reproducible in NetBeans but not IntelliJ.*

## **PART E – Priority Queue, 20 points**

The San Francisco State University's One Stop Student Services Center asks us to recommend solutions for their service lines.

- The starter code for this problem is provided in the Assignment-04-Code.zip archive.
- Our output must be identical to the complete output provided in the ZIP archive:

***PartE-The\_Complete\_Sample\_Run.pdf***

- *The right table is a portion of the output for preview purposes. It is NOT the complete output.*
- *It is a good idea to analyze the complete output thoroughly before programming a solution.*
- See assignment04PartE in the attached files.