I2206 – Data Structures Final Project

Name: Jean-Paul Khawam

File Number: 57300

**Problem 1:** Dynamic Arrays (Vectors)

Sample Input: *No input needed; code automatically runs main().*

*(The code says line by line what it is doing)*

Sample output:

Creating an empty array a...

Creating an array b of size 10...

Creating an array c with size 15, and initializing all values to 7...

Displaying array a: Empty array.

Array a is empty.

Displaying array b: Empty array.

Array b is empty.

Displaying array c: 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7

Array c is not empty.

Assigning the first 2 elements of a to 9...

Displaying array a: 9 9

Assigning the first 4 elements of b to 3...

Displaying array b: 3 3 3 3

Assigning the first 7 elements of c to 0...

Displaying array c: 0 0 0 0 0 0 0 7 7 7 7 7 7 7 7

Current size of array a: 10

Current number of elements in a: 2

Resizing a to 3...

Current size of array a: 3

Displaying array a: 9 9

Current size of array c: 15

Current number of elements in c: 15

Resizing c to 5...

Current size of array c: 5

Displaying array c: 0 0 0 0 0

Displaying array c: 0 0 0 0 0

Pushing back the number 88 to the array

Displaying array c: 0 0 0 0 0 88

Inserting the value 44 at position 0...

Displaying array c: 44 0 0 0 0 0 88

Erasing the element at position 2...

Displaying array c: 44 0 0 0 0 88

The first element of c is 44

The last element of c is 88

Deleting the last element...

Displaying array c: 44 0 0 0 0

**Problem 2:** Queues

Sample Input:

Number of queues: 3

How many rounds would you like to run?

3

Sample Output:

Round 1:

Numbers inputted into each queue: 3 4 10

Size of queues after input:

q0.size(): 3

q1.size(): 4

q2.size(): 10

Amount of data outputted from the system: 7

Size of queues after output:

q0.size(): 0

q1.size(): 0

q2.size(): 10

Round 2:

Numbers inputted into each queue: 7 3 1

Size of queues after input:

q0.size(): 7

q1.size(): 3

q2.size(): 11

Amount of data outputted from the system: 19

Size of queues after output:

q0.size(): 0

q1.size(): 0

q2.size(): 2

Round 3:

Numbers inputted into each queue: 9 1 6

Size of queues after input:

q0.size(): 9

q1.size(): 1

q2.size(): 8

Amount of data outputted from the system: 8

Size of queues after output:

q0.size(): 1

q1.size(): 1

q2.size(): 8

**Problem 3:** Stacks

Sample Input:

{8 6 4 1 1 7 1 4 1 3 7 1 4 4 4}

Sample Output:

Initial vector: 8 6 4 1 1 7 1 4 1 3 7 1 4 4 4

Sorted vector: 8 7 7 6 4 4 4 4 4 3 1 1 1 1 1

Stack 0: 8 1 1

Stack 1: 7 3

Stack 2: 7

Stack 3: 6 4

Stack 4: 4 4 1 1

Stack 5: 4 4 1

Complexity of the function: O(N logN), where N is the number of elements in the vector.

The complexities are from

1. The standard C++ sort function which has O(N logN) complexity
2. A for loop that runs N times, in it a for loop that runs as many times as there exists stacks. In the worst-case scenario (if all elements of the vector were 10), this would amount to a O(N2) complexity. But the actual run-time is way less than that and runs around O(N logN).

The total complexity becomes O(N logN) + O(N logN) = O(N logN)

**Problem 4:** Trees

**Problem 5:** Sorting

1. Sample Input:   
   { 0,1,1,0,0,2,2,2,0,0,1,0,2,1 }

Sample output:

Initial array: 0 1 1 0 0 2 2 2 0 0 1 0 2 1

Sorted Array: 0 0 0 0 0 0 1 1 1 1 2 2 2 2

1. Sample Input:   
   { 1,5,2,5,7,2,4,8,11,22,13,16,2,6,9,5,3,1 }  
     
   Sample Output:   
   1 1 2 2 2 3 4 5 5 5 6 7 8 9 11 13 16 22  
     
   Complexity of the function:  
   The complexity of the function is the complexity of passing once on each element of the input array θ(n) and then, for each occurrence of the value 0<i<27 insert it in the output array. Since we have n elements, this only takes θ(n) as well. The resulting complexity is θ(2n), or O(n).