Members:

Chua, Kyle Matthew Chan (11426012)

Santos, Kyle-AltheaFrancesca Anne Manankil (11428317)

Section: S11

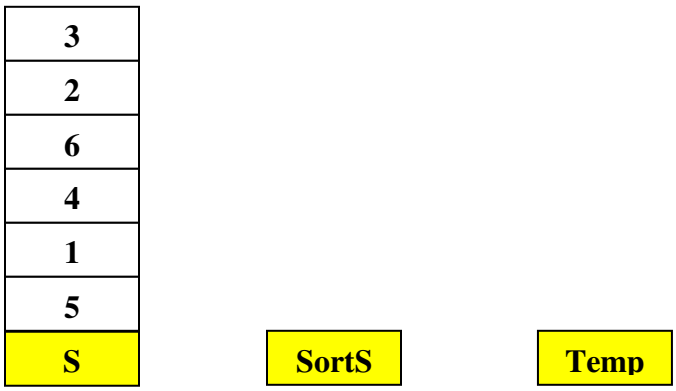
Subject: DASALGO

CASE STUDY #2

a) Discuss how your proposed stack-based and queue-based sorting algorithms work. Run through your algorithm under different cases (i.e, to sort a set of data arranged in ascending order; sort a set of data arranged in descending order; sort a set of data arranged not in any particular order) and provide a step by step trace or illustration on the crucial points of your algorithm as to how sorting process is achieved. It is also important that every time you test &/or trace your algorithm, different data size must be considered (i.e., sort a list having 6 elements; or 10 elements; or 20 elements; or 100 elements; and so on).

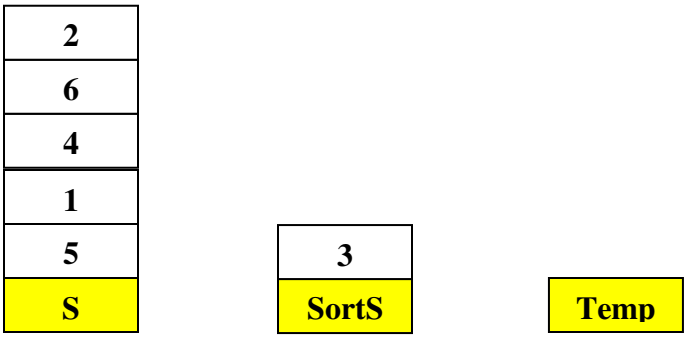
I. Stacks

(Assume sorting stack S whose elements are arranged randomly):



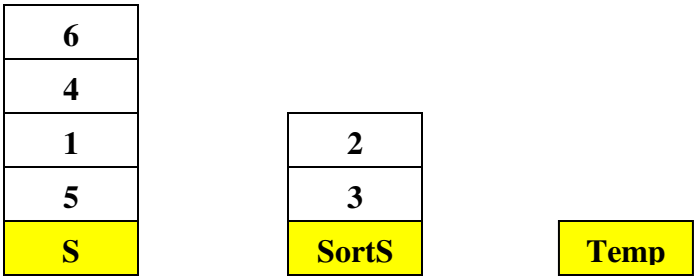
A. Sorting Stack in Ascending Order :

1. If S (original/unsorted stack) is not empty, then we push the top element of S to SortS (sorted stack), we assume that the top element of S to be the maximum of the sorted list SortS.

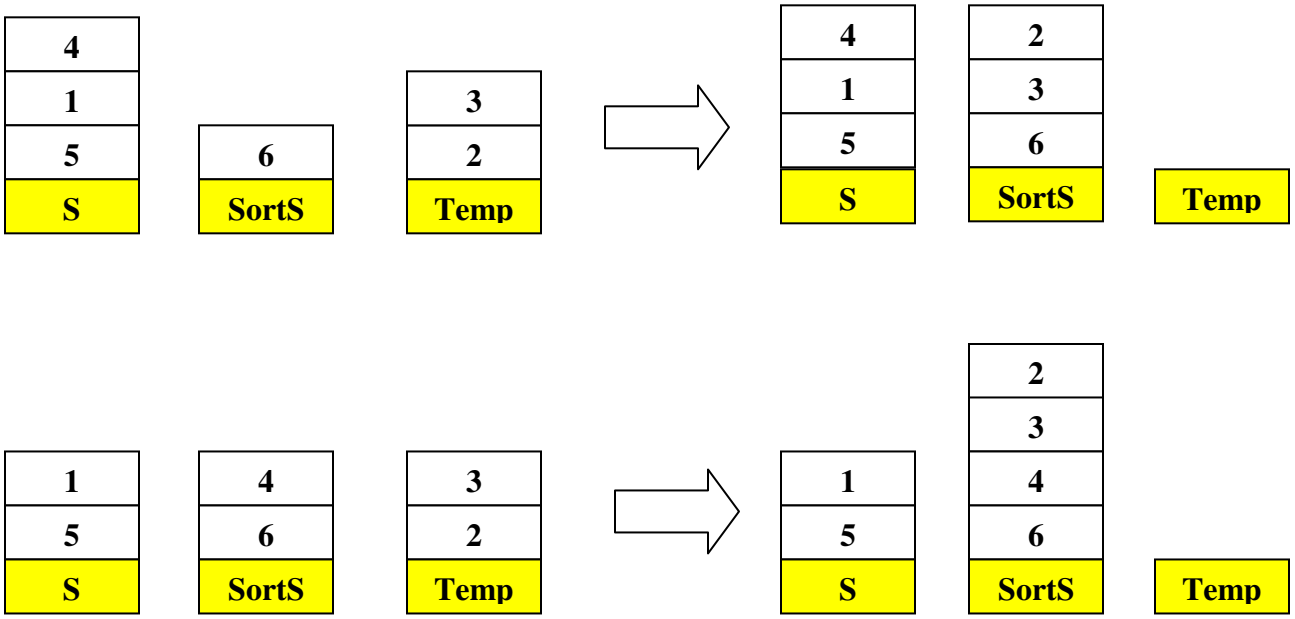


2. if Stack S is not empty, go to the next step, else go to step 5.

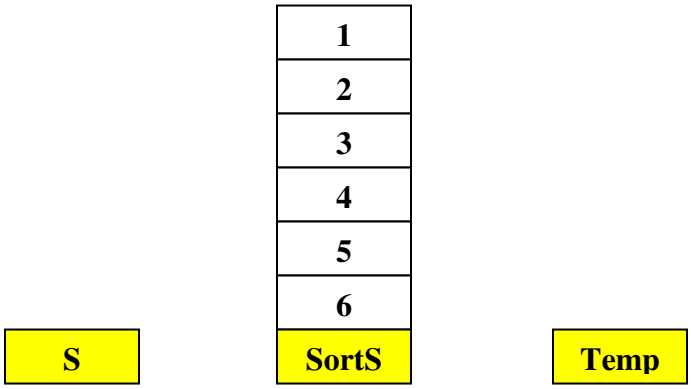
3. Compare the top element of S to the top element of SortS. If the top of S is less than the top of SortS, we just push the top of S to SortS. repeat step 2.



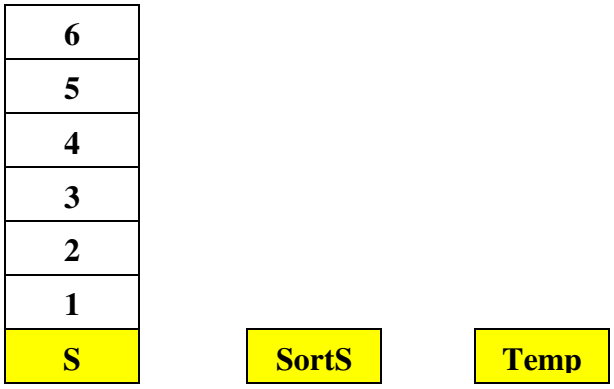
4. Else if the top of S is greater than the top of SortS, we push the top of SortS to the temporary stack temp until the top of S is less than the top of SortS and SortS is not empty, then after pushing the element of SortS to temp, we push the top of S to SortS and return all the elements of SortS from the temporary stack. Repeat step 2.



5. Now SortS contains all the elements of S in descending order, now we return all the elements to the original stack S in reverse order by pushing all elements of SortS to S.

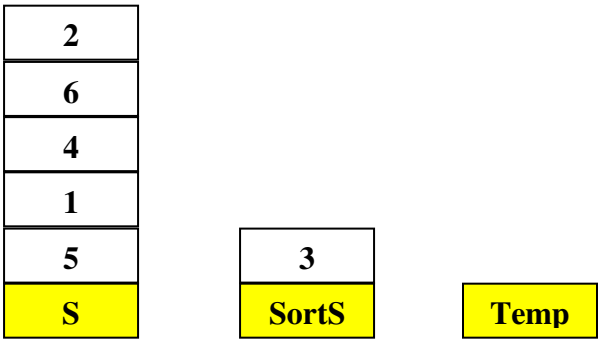


6. The result of this algorithm is that the elements of stack S arranged in ascending order.



B. Sorting Stack in Descending Order :

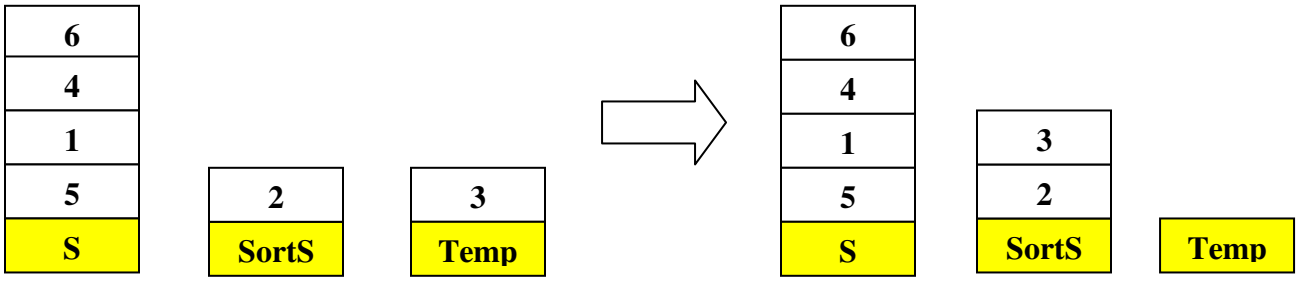
1. If S (original/unsorted stack) is not empty, then we push the top element of S to SortS (sorted stack), we assume that the top element of S to be the minimum of the sorted list SortS.

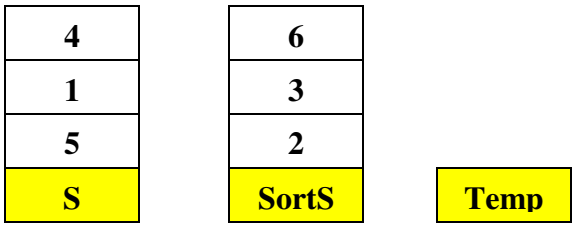


2. if Stack S is not empty, go to the next step, else go to step 5.

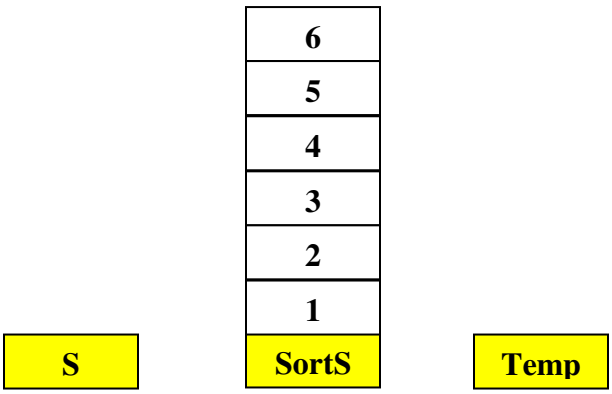
3. Compare the top element of S to the top element of SortS. If the top of S is greater than the top of SortS, we just push the top of S to SortS. repeat step 2.

4. Else if the top of S is less than the top of SortS, we push the top of SortS to the temporary stack temp until the top of S is greater than the top of SortS and SortS is not empty, then after pushing the element of SortS to temp, we push the top of S to SortS and return all the elements of SortS from the temporary stack. Repeat step 2.

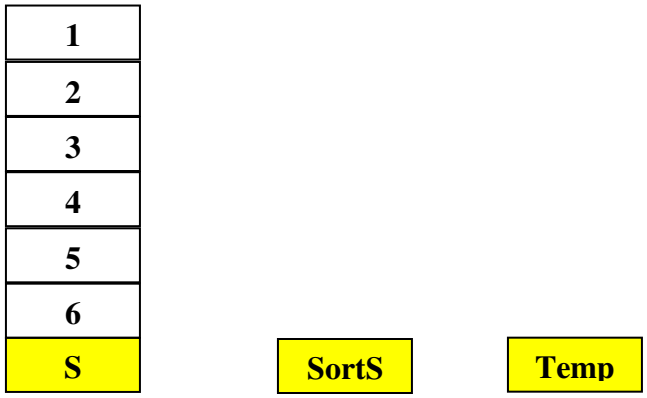




5. Now SortS contains all the elements of S in ascending order, now we return all the elements to the original stack S in reverse order by pushing all elements of SortS to S.



6. The result of this algorithm is that the elements of stack S arranged in descending order.



II. Queues

A. ASCENDING

1. The elements are queued into queue Q.
2. The elements in Q are transferred to a new queue, qTemp1, while counting the number of elements stored in Q.
3. Repeat step 3 until Q is empty.
4. A for loop is made to go through and compare every element in order to get the desired output.
5. int min gets the value of qTemp1's head;since the program assumes that it is the current smallest value in the queue.
6. The first while loop nested in the for loop will compare min to every element in the queue.
7. If statement qTemp1 is not empty, it will compare the current value of min to qTemp's current head. If the head of qTemp1 is less than min, min will take the value of qTemp1's head while min's old value is queued into Q; else, qTemp1's head will queued into Q. If qTemp1 is not empty, min will be queued into Q. A nested if statement is needed to make sure that qTemp1 is not empty, since it may be the last element in the queue and may not be greater than min. Therefore leading to the else statement that may cause an underflow error since there is nothing left to dequeue.
8. Step 8 is repeated until qTemp1 is not empty.
9. Finally, min is queued into queue qFinal wherein it will end up sorted in increasing order
10. Repeat from step 5 as long as i is greater than or equal to count or the number of elements in the original queue. This means that every element in qTemp1 has been evaluated and been sorted in order.
11. The final while loop will transfer qFinalinto Q in order.
12. Q will be displayed.

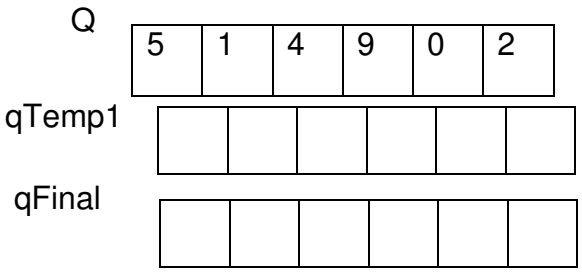
B. DESCENDING

1. The elements arequeued into queue Q.
2. The elements in Q are transferred to a new queue, qTemp1, while counting the number of elements stored in Q.

3. Repeat step 3 until Q is empty.
4. A for loop is made to go through and compare every element in order to get the desired output.
5. intmax gets the value of qTemp1's head; since the program assumes that it is the current largest value in the queue.
6. The first while loop nested in the for loop will compare max to every element in the queue.
7. If statement qTemp1 is not empty, it will compare the current value of max to qTemp's current head. If the head of qTemp1 is greater than max, max will take the value of qTemp1's head while max's old value is queued into Q; else, qTemp1's head will queued into Q. If qTemp1 is not empty, max will be queued into Q. A nested if statement is needed to make sure that qTemp1 is not empty, since it may be the last element in the queue and may not be less than min. Therefore leading to the else statement that may cause an underflow error since there is nothing left to dequeue.
8. Step 8 is repeated until qTemp1 is not empty.
9. Finally, min is queued into queue qFinal wherein it will end up sorted in decreasing order
10. Repeat from step 5 as long as i is greater than or equal to count or the number of elements in the original queue. This means that every element in qTemp1 has been evaluated and been sorted in order.
11. The final while loop will transfer qFinal into Q in order.
12. Q will be displayed.

QUEUE SORTING IN ASCENDING ORDER ILLUSTRATION:

1. Q contains all the elements to be sorted



2. The elements in Q is transferred to qTemp1 and count also incrementing for every element transferred.

   count = 6;

   i = 0;

Q						
qTemp1	5	1	4	9	0	2
qFinal						

3. min = 5;

Q						
qTemp1		1	4	9	0	2
qFinal						

4. min = 1;

Q	5					
qTemp1			4	9	0	2
qFinal						

5. min = 1;

Q	5	4				
qTemp1				9	0	2
qFinal						

6. min =1;

Q	5	4	9			
qTemp1					0	2
qFinal						

7. min =0;

Q	5	4	9	1		
qTemp1						2
qFinal						

8. min =0;

Q	5	4	9	1	2	

qTemp1						
qFinal						

9. min =0;

Q	5	4	9	1	2	
qTemp1						
qFinal	0					

10. min =0;i = 1;

Q						
qTemp1	5	4	9	1	2	
qFinal	0					

11. min = 5;

Q						
qTemp1		4	9	1	2	
qFinal	0					

12. min = 4;

Q	5					
qTemp1			9	1	2	
qFinal	0					

13. min = 4;

Q	5	9				
qTemp1				1	2	
qFinal	0					

14. min = 1;

Q	5	9	4			
qTemp1					2	
qFinal	0					

15. min = 1;

Q	5	9	4	2		

qTemp1

qFinal	0					

16. min = 1;

Q	5	9	4	2		

qTemp1						
qFinal	0	1				

17. min = 1; i = 2;

Q						

qTemp1	5	9	4	2		

qFinal	0	1				

18. min = 5;

Q						

qTemp1		9	4	2		
qFinal	0	1				

19. min = 5;

Q	9					
			4	2		
qTemp1	0	1				
qFinal						

20. min = 4;

Q	9	5				
qTemp1				2		
qFinal	0	1				

21. min = 2;

Q	9	5	4			
qTemp1						
qFinal	0	1				

22. min = 2;

Q	9	5	4			
qTemp1						
qFinal	0	1	2			

23. min = 2; i = 3;

Q						
qTemp1	9	5	4			
qFinal	0	1	2			

24. min = 9;

Q						
qTemp1		5	4			
qFinal	0	1	2			

25. min = 5;

Q	9					
qTemp1			4			

qFinal

0	1	2			

26. min = 4;

Q

9	5				

qTemp1

qFinal

0	1	2			

27. min = 4;

Q

9	5				

qTemp1

qFinal

0	1	2	4		

28. min = 4; i = 4;

Q

qTemp1

9	5				

qFinal

0	1	2	4		

29. min = 9;

Q	9					
qTemp1		5				
qFinal	0	1	2	4		

30. min = 5;

Q	9					
qTemp1						
qFinal	0	1	2	4		

31. min = 5;

Q	9					
qTemp1						
qFinal	0	1	2	4	5	

32. min = 9; i = 5;

Q						

qTemp1

qFinal

0	1	2	4	5	

33. min = 9;

Q

qTemp1

qFinal

0	1	2	4	5	9

34. max = 0;

Q

0	1	2	4	5	9

qTemp1

qFinal

QUEUE SORTING IN DESCENDING ORDER ILLUSTRATION:

1. Q contains all the elements to be sorted

Q

5	1	4	9	0	2

qTemp1

qFinal

2. The elements in Q is transferred to qTemp1 and count also incrementing for every element transferred.

   count = 6;

   i = 0;

Q						
qTemp1	5	1	4	9	0	2
qFinal						

3. max = 5;

Q						
qTemp1		1	4	9	0	2
qFinal						

4. max = 5;

Q	1					
qTemp1			4	9	0	2
qFinal						

5. max = 5;

Q	1	4				
qTemp1				9	0	2
qFinal						

6. max = 9;

Q	1	4	5			
qTemp1					0	2
qFinal						

7. max = 9;

Q	1	4	5	0		
qTemp1						2
qFinal						

8. max = 9;

Q	1	4	5	0	2	
qTemp1						

qFinal						

9. max = 9;

Q	1	4	5	0	2	
qTemp1						
qFinal	9					

10. max = 9; i = 1;

Q						
qTemp1	1	4	5	0	2	
qFinal	9					

11. max = 1;

Q						
qTemp1		4	5	0	2	
qFinal	9					

12. max = 4;

Q	1					
qTemp1			5	0	2	
qFinal	9					

13. max = 5;

Q	1	4				
qTemp1				0	2	
qFinal	9					

14. max = 5;

Q	1	4	0			
qTemp1					2	
qFinal	9					

15. max = 5;

Q	1	4	0	2		
qTemp1						
qFinal	9					

16. max = 5;

Q	1	4	0	2		
qTemp1						
qFinal	9	5				

17. max = 5;
    i = 2;

Q						
qTemp1	1	4	0	2		
qFinal	9	5				

18. max = 1;

Q						
qTemp1		4	0	2		
qFinal	9	5				

19. max = 4;

Q	1					
qTemp1			0	2		
qFinal	9	5				

20. max = 4;

Q	1	0				
qTemp1				2		
qFinal	9	5				

21. max = 4;

Q	1	0	2			
qTemp1						
qFinal	9	5				

22. max = 4;

Q	1	0	2			
qTemp1						
qFinal	9	5	4			

23. max = 4;
    i = 3;

Q						
qTemp1	1	0	2			
qFinal	9	5	4			

24. max = 1;

Q						
qTemp1		0	2			
qFinal	9	5	4			

25. max = 1;

Q	0					
qTemp1			2			
qFinal	9	5	4			

26. max = 2;

Q	0	1				
qTemp1						
qFinal	9	5	4			

27. max = 2;

Q	0	1				
qTemp1						
qFinal	9	5	4	2		

28. max = 2;
    i = 4;

Q						
qTemp1	0	1				
qFinal	9	5	4	2		

29. max = 0;

Q						
qTemp1		1				

qFinal

9	5	4	2		

30. max = 1;

Q

0					

qTemp1

qFinal

9	5	4	2		

31. max = 1;
    i = 5;

Q

qTemp1

0					

qFinal

9	5	4	2	1	

32. max = 0;

Q

qTemp1

qFinal

9	5	4	2	1	

33. max = 0;

Q

qTemp1

qFinal

9	5	4	2	1	0

34. max = 0;

Q

9	5	4	2	1	0

qTemp1

qFinal

b) Given the different experiments you will do in (a), determine the efficiency of your sorting algorithms. Explain &/or elaborate your analysis.

Stacks:

The stack-sorting algorithm's efficiency has a best case of O(n) when the stack is already sorted in ascending order and we want it to be sorted in ascending order or the stack is already arranged in descending when we want it to be sorted in descending. It is because when the stack is already sorted in ascending order, we will only go through all the elements in the source stack that would require n number of time of pop operation, however, there is no pop operation involved in the destination stack for it doesn't require to move some elements to the temporary stack, therefore satisfying the O(n) as the best case of the algorithm. The worst case of the algorithm is $O(n^2)$ when the stack is arranged in descending order but we want it sorted in ascending order and vice-versa. As stated a while there will be n number of times of pop operations to go through all the elements of the source stack, but now we need to insert the bottom element of the source stack to the bottom of the destination stack which would require n-1 pop operations in the destination stack to move all its elements to temp that would make the insertion possible. n(n-1) and therefore $O(n^2)$ is the worst case of the algorithm.

Queues:

Since the algorithm for sorting queues in descending and ascending order, follows the selection sort algorithm. Its worst case scenario is n^2 since the codes contain two nested loops. The outer loop makes sure that all the elements in the queue are checked while the inner loop compares every unarranged element with one another. This algorithm will be able to sort both small and large amount of numbers but takes time and space. Running at n^2, selection sort is slow compared to merge sort with a big Oh of nlogn.

c) Perform comparison and contrast type of analysis between your proposed stack-based and queue-based algorithms and classic sorting algorithms discussed in class (i.e., selection sort, bubble sort, among others).

Stacks:

The stack-sorting algorithm is very similar to the Insertion sort algorithm wherein an element is placed at the right place by swapping, but in the stack-based algorithm instead of swapping the elements, we remove some elements in the destination stack and placed these elements in a temporary stack until the top is the right position of the current element, insert the element and return all elements back to the destination stack.

Queues:

The algorithm used for sorting queues in descending and ascending order is based on the selection sort. Selection has a best case and worst case of O(n^2). Even if the elements in the queue is already sorted in ascending order, the program will still check if the queue is sorted or not. The bubble sort, the simplest sort, runs with the best case of O(n^2) while its best case is O(n). Merge sort, a sorting algorithm following the divide and conquer strategy, has a big Oh of O(nlogn). We can say that the selection sort is fairly slow compared to the other algorithms since it has to check all of the elements and compare it with one another one by one. It is a slow algorithm but probably the easiest one to implement with the use of queues.

d) If you are to reflect clearly, what advantages and disadvantages can be seen using stack-based sorting algorithm or queue-based sorting algorithm?

One of the advantages of using stack and queue-based sorting algorithm is that there is/are fixed points on where to start and end sorting. These points are namely the head and tail (queues) and top (stacks). These make it easier to understand and code. On the otherhand, since there is/are fixed points in accessing certain elements of the stack/queue, there is no free access of the other elements except the accessible element/s. Because of this, in-place sorting algorithm is impossible to achieve; therefore inefficient use of memory space are required to sort.

Contributions:

Kyle Chua

- Stack Algorithm
- Visualization of Stack Algorithm
- Step-by-step demonstration on stack sorting
- Analysis on efficiency of stacks algorithm
- Comparison and contrast between stack-based algorithm from classic sorting algorithm
- Advantages and disadvantages of stacks and queues

Kyle Santos

- Queues program
- Visualization of Queues Algorithm
- Step-by-Step demonstration
- Analysis on efficiency of queues algorithm
- Comparison and contrast between queue-based algorithm from classic sorting algorithm
- Advantages and disadvantages of stacks and queues