Data Structures Assignment 1

Name: Bushra Naeemi **ID: 29064**

Q1. Draw an array of 10 indices with 10 values in it, now target any value for searching and write down the low, high, and mid values at each iteration until the target value is found. [1pts]

Answer:

5	20	10	50	25	40	30	35	45	15
0	1	2	3	4	5	6	7	8	9

1. In order to use binary search algorithm, the first condition is to sort the array as ascending or descending order.
   Updated array: A[]

5	10	15	20	25	30	35	40	45	50
0	1	2	3	4	5	6	7	8	9

2. Now we can find the low, high, and mid. Moreover, my target is to find 35 in the array.
   A[]
   Search = **20**
   Low = 0
   High = size-1
   = 10 – 1
   = 9
   Mid = (Low + High) / 2

Low	High	Mid	Comparisons	Algorithm

				while(low <= high){
				If (search ==a[mid])
				Print Searched value found

0	9	(0+9)/2=4	20 < 25	else if (search>a[mid])
0	3	(0+3)/2=1	20 > 10	low = mid+1
2	3	(2+3)/2=2	20 > 15	else
3	3	(3+3)/2=3	20==20	high=mid-1

				}
				If(low>high)
				Print searched not found

 # It will print "Searched value found" in this step.

Q2. Use the array in Q1, set the target value for searching that is not available in the array, and show that how the low index becomes greater than the high index. [0.5pts]

5	10	15	20	25	30	35	40	45	50
0	1	2	3	4	5	6	7	8	9

Search = 29

Low = 0

High = size – 1

Mid = (Low + High)/2

Low	High	Mid	Comparisons	Algorithm

0
5
5
5
	9
	9
	6
	4
		(0+9)/2=4
		(5+9)/2=7
		(5+6)/2=5
		-
			29 > 25
			29 < 40
			29 < 30
			-
				while(low <= high){
				If (search ==a[mid])
				Print Searched value found
				else if (search>a[mid])
				low = mid+1
				else
				high=mid-1
				}
				If(low>high)
				Print searched not found

 # 29 is not in the array, that why the low becomes greater than high index 5>4. And it will print "Searched not found".

Q3. Use the array in Q1, sort it out using bubble sort algorithms. Show the series of all the steps. [0.5pts]

5	20	10	50	25	40	30	35	45	15
0	1	2	3	4	5	6	7	8	9

Pass 0									
5	20	10	50	25	40	30	35	45	15
5	10	20	50	25	40	30	35	45	15
5	10	20	50	25	40	30	35	45	15

5	10	20	25	50	40	30	35	45	15
5	10	20	25	40	50	30	35	45	15
5	10	20	25	40	30	50	35	45	15
5	10	20	25	40	30	35	50	45	15
5	10	20	25	40	30	35	45	50	15
5	10	20	25	40	30	35	45	15	50
Pass 1									
5	10	20	25	40	30	35	45	15	50
5	10	20	25	40	30	35	45	15	50
5	10	20	25	40	30	35	45	15	50
5	10	20	25	40	30	35	45	15	50
5	10	20	25	30	40	35	45	15	50
5	10	20	25	30	35	40	45	15	50
5	10	20	25	30	35	40	45	15	50
5	10	20	25	30	35	40	15	45	50
Pass 2									
5	10	20	25	30	35	40	15	45	50
5	10	20	25	30	35	40	15	45	50
5	10	20	25	30	35	40	15	45	50
5	10	20	25	30	35	40	15	45	50
5	10	20	25	30	35	40	15	45	50
5	10	20	25	30	35	40	15	45	50
5	10	20	25	30	35	15	40	45	50
Pass 3									
5	10	20	25	30	35	15	40	45	50
5	10	20	25	30	35	15	40	45	50
5	10	20	25	30	35	15	40	45	50
5	10	20	25	30	35	15	40	45	50
5	10	20	25	30	35	15	40	45	50
5	10	20	25	30	15	35	40	45	50
Pass 4									
5	10	20	25	30	15	35	40	45	50
5	10	20	25	30	15	35	40	45	50
5	10	20	25	30	15	35	40	45	50
5	10	20	25	30	15	35	40	45	50
5	10	20	25	15	30	35	40	45	50
Pass 5									
5	10	20	25	15	30	35	40	45	50
5	10	20	25	15	30	35	40	45	50
5	10	20	25	15	30	35	40	45	50
5	10	20	15	25	30	35	40	45	50
Pass 6									

5	10	20	15	25	30	35	40	45	50
5	10	20	15	25	30	35	40	45	50
5	10	15	20	25	30	35	40	45	50
Pass 7									
5	10	15	20	25	30	35	40	45	50
5	10	15	20	25	30	35	40	45	50
Pass 8									
5	10	15	20	25	30	35	40	45	50

 # It gets sorted in Pass 6.

Q4. Select an algorithm from the following list of algorithms. Explain, how does it work? Give an example and go through all the steps with a visual representation. [2pts]

1. **Insertion Algorithm:**

   Insertion algorithm is based on in-place comparison sorting. These algorithms are not suitable for large data sets as their worst case complexity are of O(n^2). In this algorithm the array divides into two parts where one part will always remain sorted and the other part will be unsorted. By using this algorithm, the array will get searched and unsorted values move and inserted in the sorted part. That's how the array will get sorted when there won't be any element left on the unsorted side, and all of them get inserted on the sorted part.

   For instance, the low part of the list is kept sorted, then the elements from unsorted part has to be inserted in an appropriate place in sorted part until all the elements find their place and gets sorted. That's why we call it insertion algorithm.

   Step by step sorting an array by using insertion algorithm:

   Algorithm:

   Step 1: It is already sorted if it is the first element, return 1;
   Step 2: Then pick next element
   Step 3: compare this element will all the elements in the sorted part
   Step 4: all the elements in the sorted part should be shifted if they are greater than the new value.
   Step 5: Repeat until all of the elements get sorted

Unsorted array A[]:

15	35	23	11	40

We keep the lower part sorted and the higher part unsorted.
The First element is already sorted
We pick the next element and compare it with sorted element.

15	35	23	11	40

As we see 15 and 35 is already sorted so we won't change their position and 15 is in sorted sub-list for now.

15	35	23	11	40

Then we pick the next element and compare 35 & 23.

15	35	23	11	40

By comparing we find out that 35 in not in the right place, it is greater than 23 and should be shifted.

15	35	23	11	40

35 swaps with 23 and gets in ascending order and 23 placed in sorted sub-list after checking all the elements in the sorted sub-list.

15	23	35	11	40

Now it compares the next elements, 35 with 11.

15	23	35	11	40

35 is greater than 11 and they are not in the ascending order, so 35 should be shifted.

15	23	35	11	40

11 checks with all the elements and inserted in the right place in sorted sub-list.

15	23	11	35	40

However 23 and 11 are not sorted now.

15	23	11	35	40

We swap the 23 and 11 to sort them in ascending order.

15	11	23	35	40

Now the 15 and 11 are unsorted so we swap them too.

11	15	23	35	40

Now that 11, 15, and 23 are all in sorting sub-list, we compare the next element.

15	23	11	35	40

35 is less than 40 and they are in ascending order and they won't swap. Now we don't have any unsorted value remaining and all of our values has been inserted in the sorted part.

15	23	11	35	40

 # That's how we use insertion sorting algorithm to sort arrays.

```
Insertion(A, n)
{
 for i = 1 to n -1
 {
  value = A[i]
   place = i
   while(place > 0 && A[place – 1] > value)
   {
    A[place] = A[place-1]
    place = place - 1
   }
A[place]=value
}
}
```

-Thank You-