National University of Computer and Emerging Sciences

Laboratory Manual

for

Programming Fundamentals

Course Instructor	Ms. Anoosha
Lab Instructor(s)	Mr. Adeel Mr. Junaid Hussain
Section	BS SE B
Semester	Fall 2022

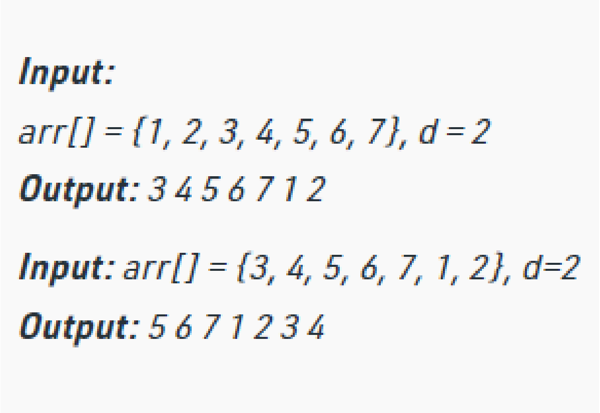
Department of Computer Science
FAST-NU, Lahore, Pakistan

Instructions:

1. Declare **const int Size** to declare 1D array
2. Write a menu -driven program, and use do while loop. The program should ask the user which function he/she wants to test, it should also ask if user wants to run it again or return to main menu, print the appropriate menu (**Zero Marks without it**).

Question No 1. Write a C++ function to find and print all unique elements of a given 1 D array of integers.

Question No 2. Function to cyclically rotate an array by a given factor d



Input:

arr[] = {1, 2, 3, 4, 5, 6, 7}, d = 2

Output: *3 4 5 6 7 1 2*

Input: *arr[] = {3, 4, 5, 6, 7, 1, 2}, d=2*

Output: *5 6 7 1 2 3 4*

Question No 3:

Write c++ program function ReverseString() which takes string as user input and finds the reverse of that string word by word. Note: **Implement through 1D character arrays.**

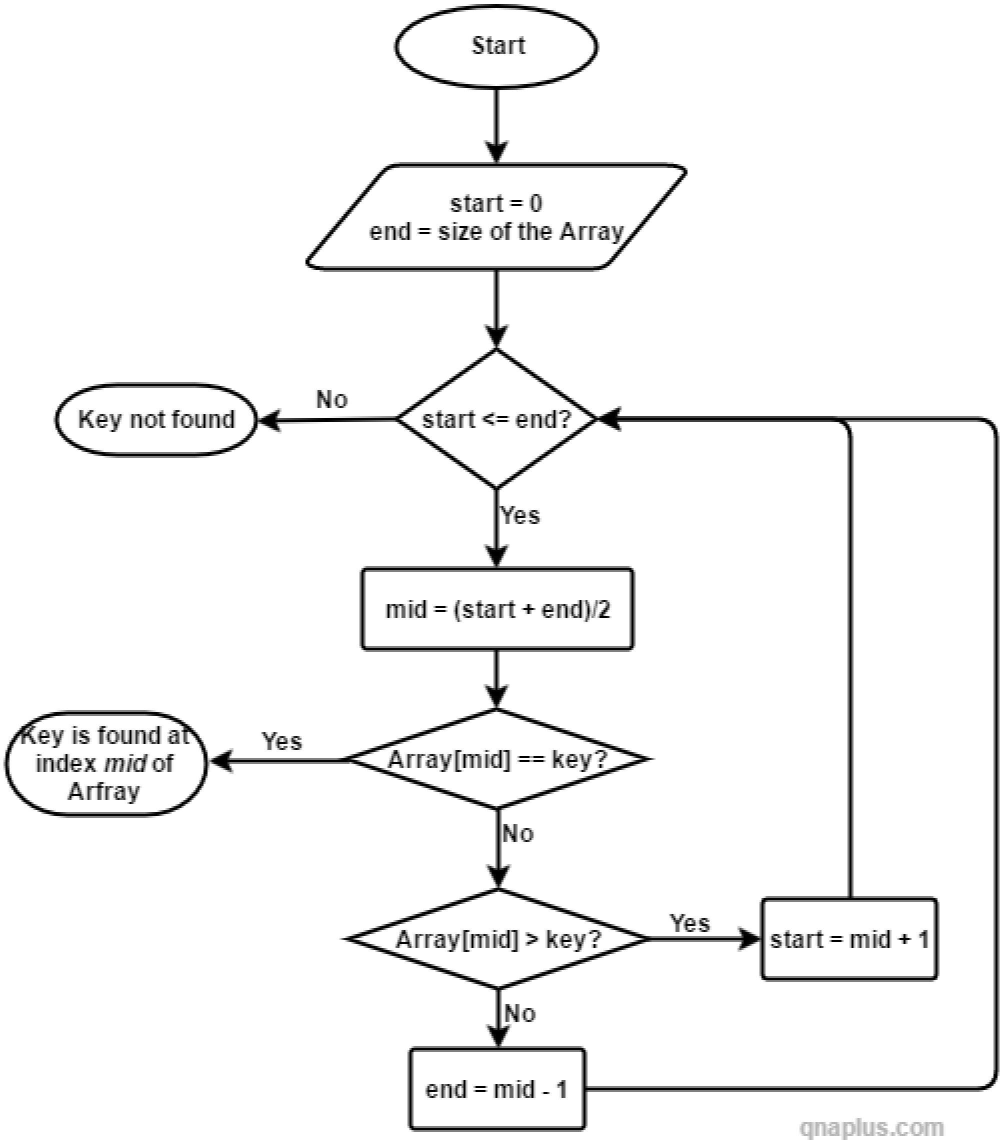
OUTPUT:

```
Enter the string: Fast University
tsaF ytisrevinU
```

Question No 4:

Write c++ program function which returns the index for a given key, implement only using binary search in the given flow chart.

Binary search algorithm: find *key* in a sorted *Array*



qnaplus.com

Question No 5:

Write c++ program to implement bubble sort as per the given algorithm below:

Input: arr[] = {5, 1, 4, 2, 8}

First Pass:

Bubble sort starts with the very first two elements, comparing them to check which one is greater.

- (**5 1** 4 2 8) –> (**1 5** 4 2 8), Here, algorithm compares the first two elements, and swaps since 5 > 1.
- (1 **5 4** 2 8) –> (1 **4 5** 2 8), Swap since 5 > 4
- (1 4 **5 2** 8) –> (1 4 **2 5** 8), Swap since 5 > 2

- (1 4 2 **5 8**) –> (1 4 2 **5 8**), Now, since these elements are already in order (8 > 5), algorithm does not swap them.

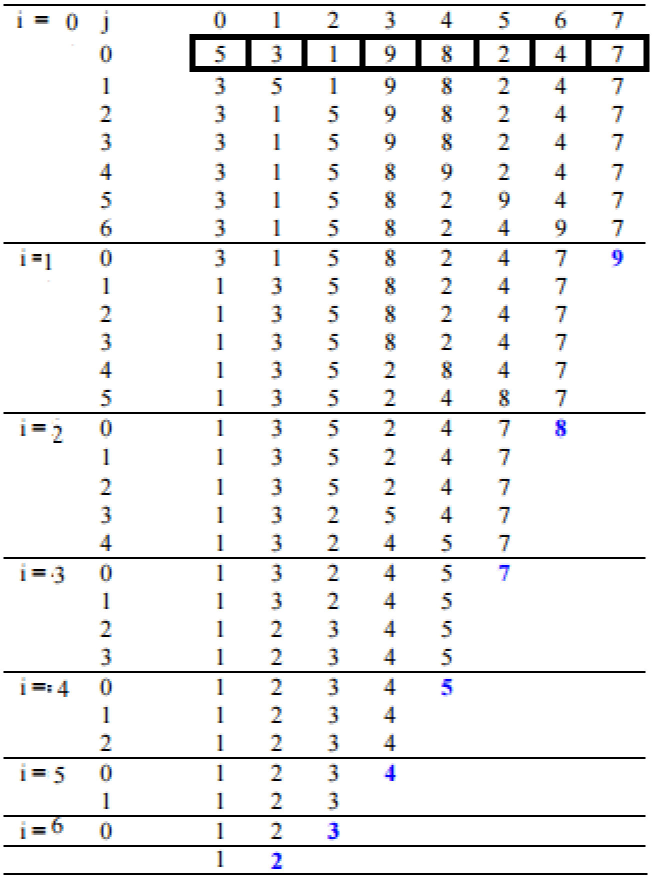
Second Pass:

Now, during the second iteration it should look like this:

- (1 **4** 2 5 8) –> (1 **4** 2 5 8)
- (1 **4 2** 5 8) –> (1 **2 4** 5 8), Swap since 4 > 2
- (1 2 **4 5** 8) –> (1 2 **4 5** 8)
- (1 2 4 **5 8**) –> (1 2 4 **5 8**)

Third Pass:

- Now, the array is already sorted, but our algorithm does not know if it is completed.
- The algorithm needs one **whole** pass without **any** swap to know it is sorted.
  - (1 **2** 4 5 8) –> (**1 2** 4 5 8)
  - (1 **2 4** 5 8) –> (1 **2 4** 5 8)
  - (1 2 **4 5** 8) –> (1 2 **4 5** 8)
  - (1 2 4 **5 8**) –> (1 2 4 **5 8**)



i = 0	j	0	1	2	3	4	5	6	7
	0	5	3	1	9	8	2	4	7
	1	3	5	1	9	8	2	4	7
	2	3	1	5	9	8	2	4	7
	3	3	1	5	9	8	2	4	7
	4	3	1	5	8	9	2	4	7
	5	3	1	5	8	2	9	4	7
	6	3	1	5	8	2	4	9	7
i =1	0	3	1	5	8	2	4	7	**9**
	1	1	3	5	8	2	4	7	
	2	1	3	5	8	2	4	7	
	3	1	3	5	8	2	4	7	
	4	1	3	5	2	8	4	7	
	5	1	3	5	2	4	8	7	
i = 2	0	1	3	5	2	4	7	**8**	
	1	1	3	5	2	4	7		
	2	1	3	5	2	4	7		
	3	1	3	2	5	4	7		
	4	1	3	2	4	5	7		
i = 3	0	1	3	2	4	5	**7**		
	1	1	3	2	4	5			
	2	1	2	3	4	5			
	3	1	2	3	4	5			
i = 4	0	1	2	3	4	**5**			
	1	1	2	3	4				
	2	1	2	3	4				
i = 5	0	1	2	3	**4**				
	1	1	2	3					
i = 6	0	1	2	**3**					
		1	**2**						

Question No. 6: Implement selection sort Algorithm

The **selection sort algorithm** sorts an array by repeatedly finding the minimum element (considering ascending order) from the unsorted part and putting it at the beginning. The algorithm maintains two subarrays in a given array.

- The subarray which already sorted.
- The remaining subarray was unsorted.

In every iteration of the selection sort, the minimum element (considering ascending order) from the unsorted subarray is picked and moved to the sorted subarray.

How does selection sort work?

Let's consider the following array as an example:
arr[] = {64, 25, 12, 22, 11}

First pass:

For the first position in the sorted array, the whole array is traversed from index 0 to 4 sequentially. The first position where **64** is stored presently, after traversing whole array it is clear that **11** is the lowest value.

64 25 12 22 11

Thus, replace 64 with 11. After one iteration **11**, which happens to be the least value in the array, tends to appear in the first position of the sorted list.

11 25 12 22 64

Second Pass:

For the second position, where 25 is present, again traverse the rest of the array in a sequential manner.

11 **25** 12 22 64

After traversing, we found that **12** is the second lowest value in the array and it should appear at the second place in the array, thus swap these values.

11 **12** 25 22 64

Third Pass:

Now, for third place, where **25** is present again traverse the rest of the array and find the third least value present in the array.

11 12 **25** 22 64

While traversing, **22** came out to be the third least value and it should appear at the third place in the array, thus swap **22** with element present at third position.

11 12 **22** 25 64

Fourth pass:
Similarly, for fourth position traverse the rest of the array and find the fourth least element in the array
As **25** is the 4th lowest value hence, it will place at the fourth position.

11 12 22 **25** 64

Fifth Pass:
At last the largest value present in the array automatically get placed at the last position in the array
The resulting array is the sorted array.

11 12 22 **25** 64

-----------GOOD LUCK-----------