**Array Data Structure**

**1A.** Aim: Write and implement a program to reverse the given array.

**Algorithm:**

Step 1: Initialize variable to store size of the array and input the size of the array.

Step 2: Allocate memory from the heap for the array.

Step 3: Use a loop to input the elements of the array from the user.

Step 4: Allocate memory from the heap for the reversed.

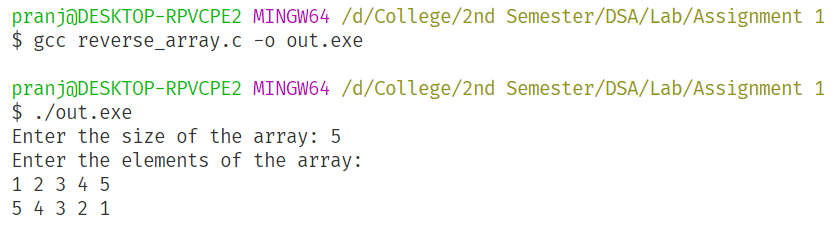
Step 5: Use a for loop with counter i from 0 to size – 1, and put reversed [i] = array[size-i-1].

Step 6: Use a for loop do display the contents of the reversed array.

**Program:** [In next page]



**Output:**



**Results:**

Thus, the program to reverse the given array is implemented with time complexity .

**1B.** Aim: Write a program to rotate the elements of an array by *d* elements.

**Algorithm:**

Step 1: Initialize variable to store size of the array and input the size of the array.

Step 2: Allocate memory from the heap for the array.

Step 3: Use a loop to input the elements of the array from the user

Step 4: Initialize a variable to store the degree of rotation and input the degree from user.

Step 5: Allocate memory from the heap for the rotated array

Step 6: Use a for loop to iterate through the elements of the array and insert the item in the new array’s index (

**Program:** [In next page]



**Output:**



**Results:**

Thus, the program to rotate the given array is implemented with time complexity .

**1C.** Aim: Given two strings S1 and S2, find if S1 is a substring of S2. If yes, return the index of the first occurrence, else return -1.

**Algorithm:**

Step 1: Initialize variable for two char arrays and input the S1 and S2.

Step 2: Traverse through each of the arrays and calculate the length of S1 and S2.

Step 3: Traverse through the main string and find the index where the character is the same as the

first character of the sub string.

Step 4: Check the next n characters of the main string to check if it matches the substring, where

n is the length of the sub string.

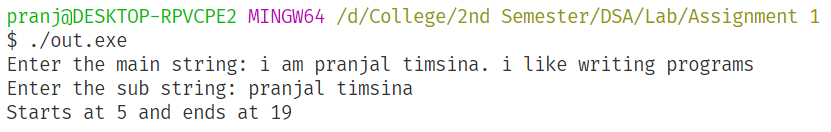
Step 5: If the substring is found, print the index and exit the program.

Step 6: If the loop iterates completely without match, return -1.

**Program:** [In next page]



**Output:**



**Results:**

Thus, the program to find the index of the substring is implemented with time complexity . Where n is the length of the main string and m is the length of the substring.

**1D.** Aim: Write a program to insert an element in an array at a specified location.

**Algorithm:**

Step 1: Input size of array from the user and allocate memory for the array

Step 2: Create another array of length size to store whether a particular index in an array is

occupied or not.

Step 3: Take input from the user for the position and data

Step 4: If the position is within bounds and the index is unoccupied, insert data in the position

Step 5: Try to shift the contents of the array cyclically right starting from the position till a

0 is encountered.

Step 6: If shifting is successful, insert data in the position

Step 7: If shifting is unsuccessful due to array being full, ask the user whether they want to

increase the size of the array.

Step 8: If the user chooses to increase the size of the array then shift data cyclically and insert

data in the position. Then go to step 3.

Step 9: If the user chooses not to increase the size of the array, exit.

**Program:** [In next page]

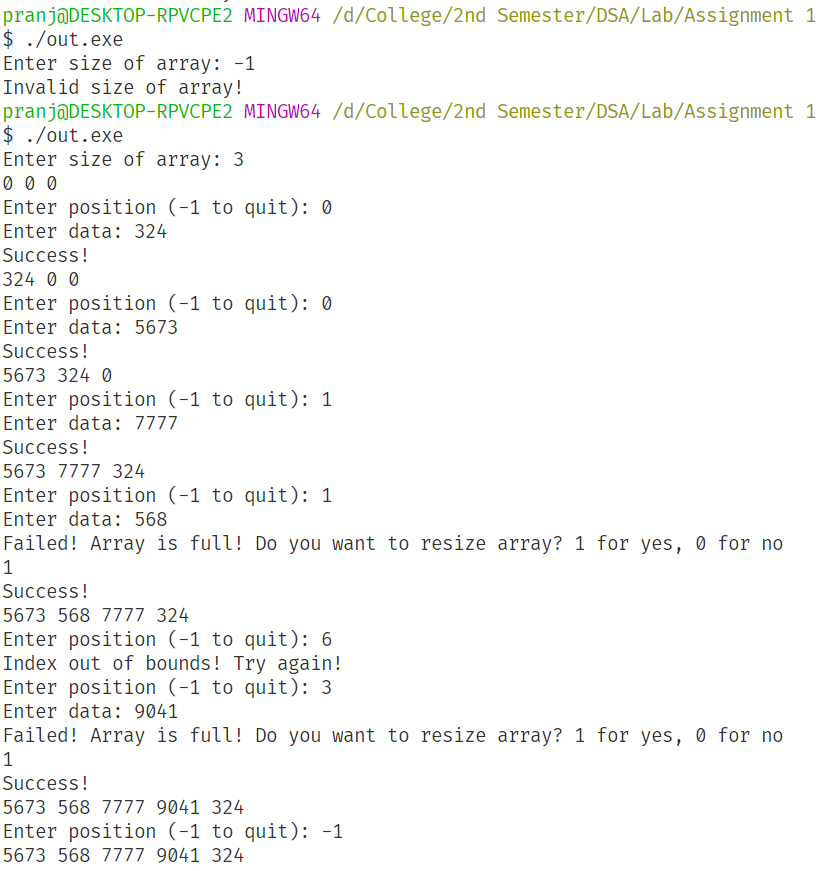








**Output:** [In next page]



**Results:**

Thus, the program to insert and element in an array at a specified location was implemented.

**1E.** Aim: Write a program to find the median of two arrays of the same size. (Use bubble sort)

**Algorithm:**

Step 1: Initialize variable for the size of the arrays.

Step 2: Allocate memory for the two arrays from the heap. Allocate twice the size for the first

array to easily merge the two arrays later.

Step 3: Append the elements of the second array to the end of the first array.

Step 4: Bubble sort the merged array.

Step 5: Calculate the median which will be the average of merged\_list[size] and

merged\_list[size-1] as the number of elements in the merged\_list is always even.

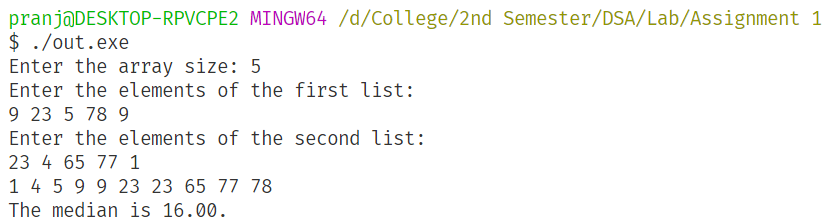
Step 6: Display the median of the merged array.

**Program:** [In next page]



****

**Output:**



**Results:**

Thus, the program to find the median of two lists was implemented.

**1F.** Aim: Write a program to print all the repeated numbers in an array with frequency.

**Algorithm:**

Step 1: Initialize the variable for the size of array and get user input

Step 2: Allocate memory from the heap for the array

Step 3: Sort the array using insertion sort, or a more efficient sorting algorithm so that the

repeated numbers will be adjacent to each other in the array.

Step 4: Initialize a counter variable to 1

Step 5: Traverse the array and compare with next.

Step 6: If the next item is different and check if counter > 1 and then print the

Frequency, then reset counter to 1.

Step 7: If the next item is the same increment the counter variable.

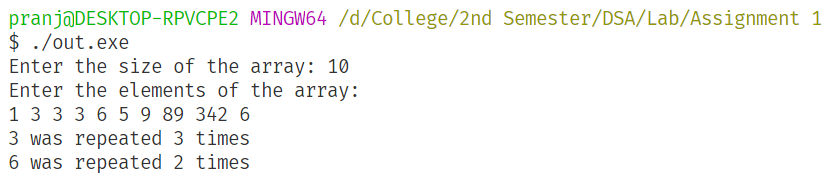
Step 8: After the array is fully traversed exit the program

**Program:** [In next page]





**Output:**



**Results:**

Thus, the program to print all the repeated numbers in an array with frequency was implemented.

**1G.** Aim: Write a program to multiply two matrices.

**Algorithm:**

Step 1: Input the dimensions for the two matrices

Step 2: Check if the dimensions are valid for multiplication

Step 3: If the dimensions are invalid, exit with error message

Step 4: Input the elements of the two matrices

Step 5: Initialize a 2D product array with rows R1 and columns C2.

Step 6: Iterate through the indices of the product array.

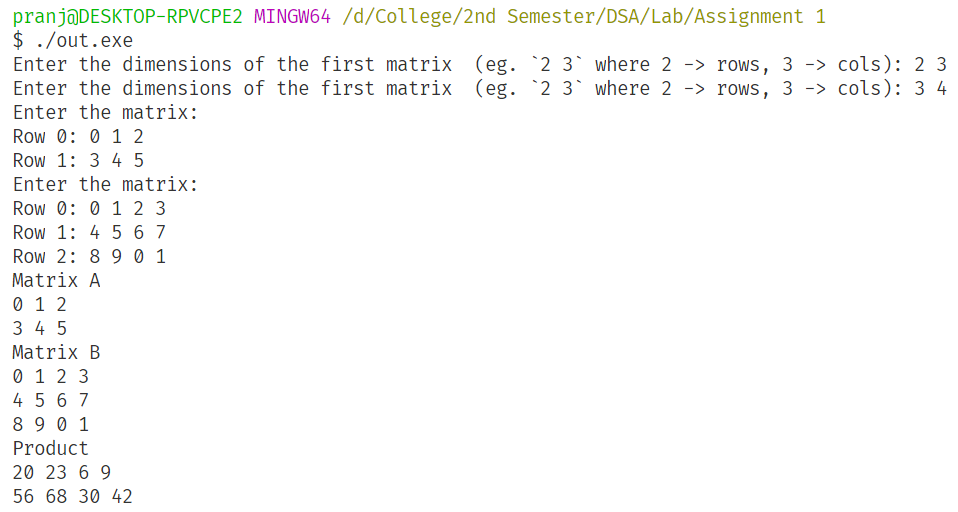
Step 7: The value of the element[i][j] of the product array is given by:

element[i][j] = a[i][h]\*b[h][j], where h = 0 to C1.

Step 8: Display the product.

**Program:** [In next page]

**Output:**



**Results:**

Thus, the program multiply two matrices was implemented.