VELAMMAL ENGINEERING COLLEGE

DEPARTMENT OF ELECTRONICS AND COMMUNICATION

CODING TEST I

Editorial

1. Search:

C:

```
#include<stdio.h>

// Function to implement search operation
int findElement(int arr[], int n,
                int key)
{
    int i;
    for (i = 0; i < n; i++)
        if (arr[i] == key)
            return i;

    return -1;
}

// Driver Code
int main()
{
    int arr[] = {12, 34, 10, 6, 40};
    int n = sizeof(arr) / sizeof(arr[0]);

    // Using a last element as search element
    int key = 40;
    int position = findElement(arr, n, key);

    if (position == - 1)
        printf("Element not found");
    else
```

```
        printf("Element Found at Position: %d", position + 1 );

    return 0;
}
```

Python:

```
def findElement(arr, n, key):
    for i in range (n):
        if (arr[i] == key):
            return i
    return -1

arr = [12, 34, 10, 6, 40]
key = 40
n = len(arr)

#search operation
index = findElement(arr, n, key)
if index != -1:
    print ("element found at position: " + str(index + 1 ))
else:
    print ("element not found")
```

Insert:

C:

```
int insertSorted(int arr[], int n,
                 int key,
                 int capacity)
{

    // Cannot insert more elements if n is
    // already more than or equal to capcity
    if (n >= capacity)
       return n;

    arr[n] = key;

    return (n + 1);
}
```

```
// Driver Code
int main()
{
    int arr[20] = {12, 16, 20, 40, 50, 70};
    int capacity = sizeof(arr) / sizeof(arr[0]);
    int n = 6;
    int i, key = 26;

    printf("\n Before Insertion: ");
    for (i = 0; i < n; i++)
        printf("%d  ", arr[i]);

    // Inserting key
    n = insertSorted(arr, n, key, capacity);

    printf("\n After Insertion: ");
    for (i = 0; i < n; i++)
        printf("%d  ",arr[i]);

    return 0;
}
```

Python:

```
def insert(arr, element):
    arr.append(element)

# declaring array and key to insert
arr = [12, 16, 20, 40, 50, 70]
key = 26

# array before inserting an element
print ("Before Inserting: ")
print (arr)

# array after Inserting element
insert(arr, key)
print("After Inserting: ")
print (arr)
```

Delete:

```
#include<stdio.h>

// To search a key to be deleted
int findElement(int arr[], int n,
                int key);

// Function to delete an element
int deleteElement(int arr[], int n,
                  int key)
{
    // Find position of element to be deleted
    int pos = findElement(arr, n, key);

    if (pos == - 1)
    {
        printf("Element not found");
        return n;
    }

    // Deleting element
    int i;
    for (i = pos; i < n - 1; i++)
        arr[i] = arr[i + 1];

    return n - 1;
}

// Function to implement search operation
int findElement(int arr[], int n, int key)
{
    int i;
    for (i = 0; i < n; i++)
        if (arr[i] == key)
            return i;

    return - 1;
}

// Driver code
int main()
{
```

```
    int i;
    int arr[] = {10, 50, 30, 40, 20};

    int n = sizeof(arr) / sizeof(arr[0]);
    int key = 30;

    printf("Array before deletion\n");
    for (i = 0; i < n; i++)
      printf("%d  ", arr[i]);

    n = deleteElement(arr, n, key);

    printf("\nArray after deletion\n");
    for (i = 0; i < n; i++)
      printf("%d  ", arr[i]);

    return 0;
}
```

Python:

```
arr = [10, 50, 30, 40, 20]
key = 30

print("Array before deletion:")
print arr

# deletes key if found in the array
# otherwise shows error not in list
arr.remove(key)
print("Array after deletion")
print(arr)
```

2. Reverse:

C:

```
// Iterative C program to reverse an array
#include<stdio.h>

/* Function to reverse arr[] from start to end*/
void rvereseArray(int arr[], int start, int end)
{
    int temp;
    while (start < end)
    {
        temp = arr[start];
        arr[start] = arr[end];
        arr[end] = temp;
        start++;
        end--;
    }
}

/* Utility that prints out an array on a line */
void printArray(int arr[], int size)
{
  int i;
  for (i=0; i < size; i++)
    printf("%d ", arr[i]);

  printf("\n");
}

/* Driver function to test above functions */
int main()
{
    int arr[] = {1, 2, 3, 4, 5, 6};
    int n = sizeof(arr) / sizeof(arr[0]);
    printArray(arr, n);
    rvereseArray(arr, 0, n-1);
    printf("Reversed array is \n");
    printArray(arr, n);
    return 0;
}
```

Python:

```
# Iterative python program to reverse an array

# Function to reverse A[] from start to end
def reverseList(A, start, end):
    while start < end:
        A[start], A[end] = A[end], A[start]
        start += 1
        end -= 1

# Driver function to test above function
A = [1, 2, 3, 4, 5, 6]
print(A)
reverseList(A, 0, 5)
print("Reversed list is")
print(A)
```

3. Head of array:

Python:

```
# Python Function to print leaders in array

def printLeaders(arr,size):

    for i in range(0, size):
        for j in range(i+1, size):
            if arr[i]<=arr[j]:
                break
        if j == size-1: # If loop didn't break
            print arr[i],

# Driver function
arr=[16, 17, 4, 3, 5, 2]
printLeaders(arr, len(arr))
```

C:

```
#include<stdio.h>


/*C Function to print leaders in an array */
void printLeaders(int arr[], int size)
{
    for (int i = 0; i < size; i++)
    {
        int j;
        for (j = i+1; j < size; j++)
        {
            if (arr[i] <= arr[j])
                break;
        }
        if (j == size) // the loop didn't break
            printf("%d",arr[i]);
}
}

/* Driver program to test above function */
int main()
{
    int arr[] = {16, 17, 4, 3, 5, 2};
    int n = sizeof(arr)/sizeof(arr[0]);
    printLeaders(arr, n);
    return 0;
}
```

4.
Method 1:

C:

```
#include <stdio.h>
#define bool int

void quickSort(int*, int, int);

bool hasArrayTwoCandidates(int A[], int arr_size, int sum)
{
    int l, r;

    /* Sort the elements */
    quickSort(A, 0, arr_size - 1);

    /* Now look for the two candidates in the sorted
       array*/
    l = 0;
    r = arr_size - 1;
    while (l < r) {
        if (A[l] + A[r] == sum)
            return 1;
        else if (A[l] + A[r] < sum)
            l++;
        else // A[i] + A[j] > sum
            r--;
    }
    return 0;
}

/* FOLLOWING FUNCTIONS ARE ONLY FOR SORTING
    PURPOSE */
void exchange(int* a, int* b)
{
    int temp;
    temp = *a;
    *a = *b;
    *b = temp;
```

```
}

int partition(int A[], int si, int ei)
{
    int x = A[ei];
    int i = (si - 1);
    int j;

    for (j = si; j <= ei - 1; j++) {
        if (A[j] <= x) {
            i++;
            exchange(&A[i], &A[j]);
        }
    }
    exchange(&A[i + 1], &A[ei]);
    return (i + 1);
}

/* Implementation of Quick Sort
A[] --> Array to be sorted
si  --> Starting index
ei  --> Ending index
*/
void quickSort(int A[], int si, int ei)
{
    int pi; /* Partitioning index */
    if (si < ei) {
        pi = partition(A, si, ei);
        quickSort(A, si, pi - 1);
        quickSort(A, pi + 1, ei);
    }
}

/* Driver program to test above function */
int main()
{
    int A[] = { 1, 4, 45, 6, 10, -8 };
    int n = 16;
    int arr_size = 6;

    if (hasArrayTwoCandidates(A, arr_size, n))
```

```
        printf("Array has two elements with given sum");
    else
        printf("Array doesn't have two elements with given sum");

    getchar();
    return 0;
}
```

Python:

```
def hasArrayTwoCandidates(A, arr_size, sum):

    # sort the array
    quickSort(A, 0, arr_size-1)
    l = 0
    r = arr_size-1

    # traverse the array for the two elements
    while l<r:
        if (A[l] + A[r] == sum):
            return 1
        elif (A[l] + A[r] < sum):
            l += 1
        else:
            r -= 1
    return 0

# Implementation of Quick Sort
# A[] --> Array to be sorted
# si  --> Starting index
# ei  --> Ending index
def quickSort(A, si, ei):
    if si < ei:
        pi = partition(A, si, ei)
        quickSort(A, si, pi-1)
        quickSort(A, pi + 1, ei)

# Utility function for partitioning the array(used in quick sort)
def partition(A, si, ei):
    x = A[ei]
    i = (si-1)
```

```
    for j in range(si, ei):
        if A[j] <= x:
            i += 1

            # This operation is used to swap two variables is python
            A[i], A[j] = A[j], A[i]

        A[i + 1], A[ei] = A[ei], A[i + 1]

    return i + 1


# Driver program to test the functions
A = [1, 4, 45, 6, 10, -8]
n = 16
if (hasArrayTwoCandidates(A, len(A), n)):
    print("Array has two elements with the given sum")
else:
    print("Array doesn't have two elements with the given sum")
```

Method 2:

C:

```
// Works only if range elements is limited
#include <stdio.h>
#define MAX 100000

void printPairs(int arr[], int arr_size, int sum)
{
    int i, temp;
    bool s[MAX] = { 0 }; /*initialize hash set as 0*/

    for (i = 0; i < arr_size; i++) {
        temp = sum - arr[i];
        if (s[temp] == 1)
            printf("Pair with given sum %d is (%d, %d) n",
                   sum, arr[i], temp);
        s[arr[i]] = 1;
    }
}
```

```
/* Driver program to test above function */
int main()
{
    int A[] = { 1, 4, 45, 6, 10, 8 };
    int n = 16;
    int arr_size = sizeof(A) / sizeof(A[0]);

    printPairs(A, arr_size, n);

    getchar();
    return 0;
}
```

Python:

```
# Python program to find if there are
# two elements wtih given sum

# function to check for the given sum
# in the array
def printPairs(arr, arr_size, sum):

    # Create an empty hash set
    s = set()

    for i in range(0, arr_size):
        temp = sum-arr[i]
        if (temp in s):
            print "Pair with given sum "+ str(sum) + " is (" + str(arr[i]) + ", " +
str(temp) + ")"
        s.add(arr[i])

# driver program to check the above function
A = [1, 4, 45, 6, 10, 8]
n = 16
printPairs(A, len(A), n)
```

5.

Method 1:
C:

```c
#include <stdio.h>
void findMajority(int arr[], int n)
{
    int maxCount = 0;
    int index = -1; // sentinels
    for(int i = 0; i < n; i++)
    {
        int count = 0;
        for(int j = 0; j < n; j++)
        {
            if(arr[i] == arr[j])
            count++;
        }

        // update maxCount if count of
        // current element is greater
        if(count > maxCount)
        {
            maxCount = count;
            index = i;
        }
    }

    // if maxCount is greater than n/2
    // return the corresponding element
    if (maxCount > n/2)
    printf("%d \n", arr[index] );

    else
        printf( "No Majority Element" );
}
int main()
{
    int arr[] = {1, 1, 2, 1, 3, 5, 1};
    int n = sizeof(arr) / sizeof(arr[0]);

    findMajority(arr, n);

    return 0;
}
```

Python:

```
def findMajority(arr, n):
    maxCount = 0;
    index = -1 # sentinels
    for i in range(n):

        count = 0
        for j in range(n):

            if(arr[i] == arr[j]):
                count += 1

        # update maxCount if count of
        # current element is greater
        if(count > maxCount):

            maxCount = count
            index = i

    # if maxCount is greater than n/2
    # return the corresponding element
    if (maxCount > n//2):
        print(arr[index])

    else:
        print("No Majority Element")

if __name__ == "__main__":
    arr = [1, 1, 2, 1, 3, 5, 1]
    n = len(arr)
    findMajority(arr, n)
```

Note: for other methods refer:https://www.geeksforgeeks.org/majority-element/

6.
C:

```
#include <stdio.h>

int _mergeSort(int arr[], int temp[], int left, int right);
int merge(int arr[], int temp[], int left, int mid, int right);

/* This function sorts the input array and returns the
   number of inversions in the array */
int mergeSort(int arr[], int array_size)
{
    int* temp = (int*)malloc(sizeof(int) * array_size);
    return _mergeSort(arr, temp, 0, array_size - 1);
}

/* An auxiliary recursive function that sorts the input array and
  returns the number of inversions in the array. */
int _mergeSort(int arr[], int temp[], int left, int right)
{
    int mid, inv_count = 0;
    if (right > left) {
        /* Divide the array into two parts and call _mergeSortAndCountInv()
       for each of the parts */
        mid = (right + left) / 2;

        /* Inversion count will be the sum of inversions in left-part, right-
part
      and number of inversions in merging */
        inv_count += _mergeSort(arr, temp, left, mid);
        inv_count += _mergeSort(arr, temp, mid + 1, right);

        /*Merge the two parts*/
        inv_count += merge(arr, temp, left, mid + 1, right);
    }
    return inv_count;
}

/* This funt merges two sorted arrays and returns inversion count in
   the arrays.*/
int merge(int arr[], int temp[], int left, int mid, int right)
```

```
{
    int i, j, k;
    int inv_count = 0;

    i = left; /* i is index for left subarray*/
    j = mid; /* j is index for right subarray*/
    k = left; /* k is index for resultant merged subarray*/
    while ((i <= mid - 1) && (j <= right)) {
        if (arr[i] <= arr[j]) {
            temp[k++] = arr[i++];
        }
        else {
            temp[k++] = arr[j++];

            /*this is tricky -- see above explanation/diagram for merge()*/
            inv_count = inv_count + (mid - i);
        }
    }

    /* Copy the remaining elements of left subarray
   (if there are any) to temp*/
    while (i <= mid - 1)
        temp[k++] = arr[i++];

    /* Copy the remaining elements of right subarray
   (if there are any) to temp*/
    while (j <= right)
        temp[k++] = arr[j++];

    /*Copy back the merged elements to original array*/
    for (i = left; i <= right; i++)
        arr[i] = temp[i];

    return inv_count;
}

/* Driver program to test above functions */
int main(int argv, char** args)
{
    int arr[] = { 1, 20, 6, 4, 5 };
    printf(" Number of inversions are %d \n", mergeSort(arr, 5));
    getchar();
```

```
    return 0;
}
```

Python:

```
# Function to Use Inversion Count
def mergeSort(arr, n):
    # A temp_arr is created to store
    # sorted array in merge function
    temp_arr = [0]*n
    return _mergeSort(arr, temp_arr, 0, n-1)

# This Function will use MergeSort to count inversions

def _mergeSort(arr, temp_arr, left, right):

    # A variable inv_count is used to store
    # inversion counts in each recursive call

    inv_count = 0

    # We will make a recursive call if and only if
    # we have more than one elements

    if left < right:

        # mid is calculated to divide the array into two subarrays
        # Floor division is must in case of python

        mid = (left + right)//2

        # It will calculate inversion counts in the left subarray

        inv_count += _mergeSort(arr, temp_arr, left, mid)

        # It will calculate inversion counts in right subarray

        inv_count += _mergeSort(arr, temp_arr, mid + 1, right)

        # It will merge two subarrays in a sorted subarray

        inv_count += merge(arr, temp_arr, left, mid, right)
```

```
    return inv_count

# This function will merge two subarrays in a single sorted subarray
def merge(arr, temp_arr, left, mid, right):
    i = left     # Starting index of left subarray
    j = mid + 1 # Starting index of right subarray
    k = left     # Starting index of to be sorted subarray
    inv_count = 0

    # Conditions are checked to make sure that i and j don't exceed their
    # subarray limits.

    while i <= mid and j <= right:

        # There will be no inversion if arr[i] <= arr[j]

        if arr[i] <= arr[j]:
            temp_arr[k] = arr[i]
            k += 1
            i += 1
        else:
            # Inversion will occur.
            temp_arr[k] = arr[j]
            inv_count += (mid-i + 1)
            k += 1
            j += 1

    # Copy the remaining elements of left subarray into temporary array
    while i <= mid:
        temp_arr[k] = arr[i]
        k += 1
        i += 1

    # Copy the remaining elements of right subarray into temporary array
    while j <= right:
        temp_arr[k] = arr[j]
        k += 1
        j += 1

    # Copy the sorted subarray into Original array
    for loop_var in range(left, right + 1):
```

```
        arr[loop_var] = temp_arr[loop_var]

    return inv_count

# Driver Code
# Given array is
arr = [1, 20, 6, 4, 5]
n = len(arr)
result = mergeSort(arr, n)
print("Number of inversions are", result)
```

For more info: https://www.geeksforgeeks.org/counting-inversions/

7.

C:

```
#include<stdio.h>
int FindMaxSum(int arr[], int n)
{
  int incl = arr[0];
  int excl = 0;
  int excl_new;
  int i;

  for (i = 1; i < n; i++)
  {
     /* current max excluding i */
     excl_new = (incl > excl)? incl: excl;

     /* current max including i */
     incl = excl + arr[i];
     excl = excl_new;
  }

   /* return max of incl and excl */
   return ((incl > excl)? incl : excl);
}

/* Driver program to test above function */
int main()
{
```

```
  int arr[] = {5, 5, 10, 100, 10, 5};
  int n = sizeof(arr) / sizeof(arr[0]);
  printf("%d n", FindMaxSum(arr, n));
  return 0;
}
```

Python:

```
def find_max_sum(arr):
    incl = 0
    excl = 0

    for i in arr:

        # Current max excluding i (No ternary in
        # Python)
        new_excl = excl if excl>incl else incl

        # Current max including i
        incl = excl + i
        excl = new_excl

    # return max of incl and excl
    return (excl if excl>incl else incl)

# Driver program to test above function
arr = [5, 5, 10, 100, 10, 5]
print find_max_sum(arr)
```

8.

C:

```
#include <stdio.h>

/* Function to left Rotate arr[] of size n by 1*/
void leftRotatebyOne(int arr[], int n);

/*Function to left rotate arr[] of size n by d*/
void leftRotate(int arr[], int d, int n)
{
    int i;
    for (i = 0; i < d; i++)
        leftRotatebyOne(arr, n);
}

void leftRotatebyOne(int arr[], int n)
{
    int temp = arr[0], i;
    for (i = 0; i < n - 1; i++)
        arr[i] = arr[i + 1];
    arr[i] = temp;
}

/* utility function to print an array */
void printArray(int arr[], int n)
{
    int i;
    for (i = 0; i < n; i++)
        printf("%d ", arr[i]);
}

/* Driver program to test above functions */
int main()
{
    int arr[] = { 1, 2, 3, 4, 5, 6, 7 };
    leftRotate(arr, 2, 7);
    printArray(arr, 7);
    return 0;
}
```

Python:

```
def leftRotate(arr, d, n):
    for i in range(d):
        leftRotatebyOne(arr, n)

# Function to left Rotate arr[] of size n by 1*/
def leftRotatebyOne(arr, n):
    temp = arr[0]
    for i in range(n-1):
        arr[i] = arr[i + 1]
    arr[n-1] = temp


# utility function to print an array */
def printArray(arr, size):
    for i in range(size):
        print ("% d"% arr[i], end =" ")


# Driver program to test above functions */
arr = [1, 2, 3, 4, 5, 6, 7]
leftRotate(arr, 2, 7)
printArray(arr, 7)
```

For more info: https://www.geeksforgeeks.org/array-rotation/

9.
C:

```
#include <stdio.h>

int findPivot(int[], int, int);
int binarySearch(int[], int, int, int);

/* Searches an element key in a pivoted sorted array arrp[]
   of size n */
int pivotedBinarySearch(int arr[], int n, int key)
{
   int pivot = findPivot(arr, 0, n-1);

   // If we didn't find a pivot, then array is not rotated at all
   if (pivot == -1)
      return binarySearch(arr, 0, n-1, key);

   // If we found a pivot, then first compare with pivot and then
   // search in two subarrays around pivot
   if (arr[pivot] == key)
      return pivot;
   if (arr[0] <= key)
      return binarySearch(arr, 0, pivot-1, key);
   return binarySearch(arr, pivot+1, n-1, key);
}

/* Function to get pivot. For array 3, 4, 5, 6, 1, 2 it returns
   3 (index of 6) */
int findPivot(int arr[], int low, int high)
{
   // base cases
   if (high < low)  return -1;
   if (high == low) return low;

   int mid = (low + high)/2;   /*low + (high - low)/2;*/
   if (mid < high && arr[mid] > arr[mid + 1])
      return mid;
   if (mid > low && arr[mid] < arr[mid - 1])
      return (mid-1);
   if (arr[low] >= arr[mid])
      return findPivot(arr, low, mid-1);
```

```
   return findPivot(arr, mid + 1, high);
}

/* Standard Binary Search function*/
int binarySearch(int arr[], int low, int high, int key)
{
   if (high < low)
      return -1;
   int mid = (low + high)/2;  /*low + (high - low)/2;*/
   if (key == arr[mid])
      return mid;
   if (key > arr[mid])
      return binarySearch(arr, (mid + 1), high, key);
   return binarySearch(arr, low, (mid -1), key);
}

/* Driver program to check above functions */
int main()
{
   // Let us search 3 in below array
   int arr1[] = {5, 6, 7, 8, 9, 10, 1, 2, 3};
   int n = sizeof(arr1)/sizeof(arr1[0]);
   int key = 3;
   printf("Index of the element is : %d",
        pivotedBinarySearch(arr1, n, key));
   return 0; }
```

Python:

```
def pivotedBinarySearch(arr, n, key):

    pivot = findPivot(arr, 0, n-1);

    # If we didn't find a pivot,
    # then array is not rotated at all
    if pivot == -1:
        return binarySearch(arr, 0, n-1, key);

    # If we found a pivot, then first
    # compare with pivot and then
```

```
    # search in two subarrays around pivot
    if arr[pivot] == key:
        return pivot
    if arr[0] <= key:
        return binarySearch(arr, 0, pivot-1, key);
    return binarySearch(arr, pivot+1, n-1, key);


# Function to get pivot. For array
# 3, 4, 5, 6, 1, 2 it returns 3
# (index of 6)
def findPivot(arr, low, high):

    # base cases
    if high < low:
        return -1
    if high == low:
        return low

    #low + (high - low)/2;
    mid = int((low + high)/2)

    if mid < high and arr[mid] > arr[mid + 1]:
        return mid
    if mid > low and arr[mid] < arr[mid - 1]:
        return (mid-1)
    if arr[low] >= arr[mid]:
        return findPivot(arr, low, mid-1)
    return findPivot(arr, mid + 1, high)

# Standard Binary Search function*/
def binarySearch(arr, low, high, key):

    if high < low:
        return -1

    #low + (high - low)/2;
    mid = int((low + high)/2)

    if key == arr[mid]:
        return mid
```

```
    if key > arr[mid]:
        return binarySearch(arr, (mid + 1), high,
                                    key);
    return binarySearch(arr, low, (mid -1), key);

arr1 = [5, 6, 7, 8, 9, 10, 1, 2, 3]
n = len(arr1)
key = 3
print("Index of the element is : ",
      pivotedBinarySearch(arr1, n, key))
```

For more info: https://www.geeksforgeeks.org/search-an-element-in-a-sorted-and-pivoted-array/

10.
C:

```
#include <stdio.h>

/* getMissingNo takes array and size of array as arguments*/
int getMissingNo(int a[], int n)
{
    int i, total;
    total = (n + 1) * (n + 2) / 2;
    for (i = 0; i < n; i++)
        total -= a[i];
    return total;
}

/*program to test above function */
int main()
{
    int a[] = { 1, 2, 4, 5, 6 };
    int miss = getMissingNo(a, 5);
    printf("%d", miss);
    getchar();
}
```

Python:

```
def getMissingNo(A):
    n = len(A)
    total = (n + 1)*(n + 2)/2
    sum_of_A = sum(A)
    return total - sum_of_A

A = [1, 2, 4, 5, 6]
miss = getMissingNo(A)
print(miss)
```

For more info: https://www.geeksforgeeks.org/find-the-missing-number/