CS-2009 Design and Analysis of Algorithms

Assignment Two

22L-6552

Section: 4H

22L-6552 4H

Q1) You are given two sorted arrays A[] and B[] of size "N" each. Your task is to design an in-place algorithm using divide and conquer technique to determine the median of both the arrays in O(Log N). Median is the value located at the central position of sorted data.

```
Median(A[], B[], leftA, rightA, leftB, rightB)
{
    if(rightA-leftA+1<=2 AND rightB-leftB+1<=2)
    {
        return ManualMedian(A, B, leftA, rightA, leftB, rightB)
    }
    else
    {
        midA= (leftA+rightA)/2;
        midB = (leftB+rightB)/2
        medianA = A[midA]
        medianB = B[midB]
        if(medianA == medianB)
        {
            return medianA
        }
        elseif(medianA < medianB)
        {
            return Median(A, B, midA, rightA, leftB, midB)
        }
```

```
else
{
    return Median(A, B, leftA, midA, midB, rightB)
}
}
}
```

22L-6552 4H

```
Manual Median (A[], B[], leftA, rightA, leftB, rightB)
{
    if( rightA - leftA + 1 == 1 and rightB - leftB + 1 == 1)
    {
        return (A[leftA] + B[leftB]) / 2
    }
    else if (rightA - leftA + 1 == 1 and rightB - leftB + 1 == 2)
    {
        temp [1 - - - - 3]
        temp[1] = A[leftA]
        temp[2] = B[leftB]
        temp[3] = B[rightB]
        sort(temp, 3)

        return temp[2]
    }
    else if (rightA - leftA + 1 == 2 and rightB - leftB + 1 == 1)
    {
        temp [1 - - - 3]
        temp[1] = A[leftA]
        temp[2] = A[rightA]
        temp[3] = B[leftB]
        sort(temp, 3)

        return temp[2]
    }
```

```
else
{
    temp[1 --- 4]
    temp[1] = A[leftA]
    temp[2] = A[rightA]
    temp[3] = B[leftB]
    temp[4] = B[rightB]
    sort(temp, 4)
    return (temp[2] + temp[3])/2
}
}
```

22L-6552 4H

Q2) Given an array of integers Arr[] of length "N". Your task is to design an algorithm using divide and conquer technique to determine the majority element in the array. A majority element is an element that appears more than half (N/2) times in the array. Your algorithm must return the majority element if it exists in an array, otherwise -1.

```
FindMajorityElement(Arr[], left, right)
{
    if(left == right)
    {
        return Arr[left]
    }

    mid = (left + right)/2
    leftMajority = findMajorityElement(Arr, left, mid)
    rightMajority = findMajorityElement(Arr, mid+1, right)
    if(leftMajority == rightMajority)
    {
        return leftMajority
    }
}
```

```
leftCount = countFrequency(Arr, left, right, leftMajority)
rightCount = countFrequency(Arr, left, right, rightMajority)
if ( leftCount > (right - left + 1) / 2)
{
        return leftMajority
}
else if (rightCount > (right - left + 1) / 2)
{
        return rightMajority
}
else
{
        return -1
}
}
```

22L-6552 4H

```
countFrequency(Arr[], left, right, value)
{
    count = 0
    for(i = left; i <= right; i++)
    {
        if(Arr[i] == value)
        {
            count++
        }
    }
    return count
}
```

Q3) Consider an array of distinct integers Arr[] of size "N" was sorted in ascending order. The array has been rotated clockwise "K" number of times. Given such an array, your task is to find the value of "K". Design and algorithm using divide and conquer technique to get the value of "K" in O(Log N).

```
FindRotations (Arr[], left, right)
{
    if (right < left)
    {
        return 0
    }
    if (right == left)
    {
        return left -1
    }
    mid = (left+right)/2
    if ( mid < right and Arr[mid+1] < Arr[mid]
    {
        return mid+1 -1
    }
}
```

22L-6552 4H

```
if (mid > low and Arr[mid] < Arr[mid-1])
{
    return mid-1
}

if( Arr[right] > Arr[mid])
{
    return findRotations(Arr, left, mid-1)
}
else
{
    return findRotations(Arr, mid+1, right)
}
}
```

Q4) Assume that you are given an unsorted array of integers Arr[] of size "N" and two integers "x" and "y". It is guaranteed that the values of both the integers are from the original array. Your task is to find the distance between "x" and "y". The distance between two elements of the array is the number of elements that lie between them in sorted order. Design an efficient algorithm using the divide and conquer approach to get the distance between two elements of ~~the~~ the array.

```
main()
{
        d = DistanceXY(Arr, left, right, X, Y)

        return d-2  // -2 is to exclude the
                       count of boundaries
}
```

22L-6552 4H

```
Distance XY ( Arr[], left, right, X, Y)
{
    if( left < right)
    {
        mid = (left + right)/2
        distance = 0
        distance += Distance XY (Arr, left, mid, X, Y)
        distance += Distance XY (Arr, mid+1, right, X, Y)
        return distance

    }
    else
    {
        if (X < Y)
        {
            lower Bound = X
            upper Bound = Y
        }
        else
        {
            lower Bound = Y
            upper Bound = X
        }
```

~~Arr[left] >= lowerBound and Arr[left] <=~~

~~Arr[left] >= lowerBound and Arr[left]~~

```
if (Arr[left] >= lowerBound and Arr[left] <= upperBound)
{
    return 1
}
else
{
    return 0
}
}
}
```

22L-6552 4H

Q5) Consider an unsorted array of integers Arr[] of size "N" and an integer "Y". Design an algorithm using divide and conquer approach to solve this problem in O(NLog(N))!

```
findPair(Arr[], left, right, y)
{
    flag = false
    MergeSort(Arr, left, right)
    for (i = left; i<=right and flag==false; i++)
    {
        searchKey = Arr[i] + y ; // if b-a = y
                                    then b = a+y
        flag = BinarySearch(Arr, left+1, right, searchKey)
        // binary search function returns the index of the
        value if the value is found otherwise 0.
    }
    return flag
}
```

```
Binary Search (Arr[], left, right, searchKey)
{
    while (left <= right)
    {
        mid = (left + right) / 2
        if (Arr[mid] == searchKey)
        {
            return mid
        }
        else if (Arr[mid] < searchKey)
        {
            left = mid + 1
        }
        else
        {
            right = mid - 1
        }
    }
    return 0
}
```

22L-6552 4H

```
MergeSort(Arr[], left, right)
{
    if (left < right)
    {
        M = (left + right)/2

        MergeSort(Arr, left, M)
        MergeSort(Arr, M+1, right)
        Merge(Arr, left, M, right)
    }
}
```

```
Merge(Arr[], left, M, right)
{
    n1 = M - left + 1
    n2 = right - M

    A[1 --- n1]  // temporary arrays
    B[1 --- n2]

    Copy(Arr, A, left, M)  // copy data into
    Copy(Arr, B, M+1, right)  temporary arrays
```

```
i = 1
j = 1
K = left

while(i <= n1 and j <= n2)
{
    if ( A[i] <= B[j])
    {
        Arr[K++] = A[i++]
    }
    else
    {
        Arr[K++] = B[j++]
    }
}

while(i <= n1)
{
    Arr[K++] = A[i++]
}

while(j <= n2)
{
    Arr[K++] = B[j++]
}
```

22L-6552 4H

Delete A and B

}

Mid 1 Q 1) You are given a sorted array A of n integers, a search key V, and a positive integer $K < n$. Your goal is to find the K nearest values to the search key present in the array. Note that the search key may or may not be present in the array. Devise an efficient algorithm that returns the subarray (starting and ending index) containing the K nearest values.

```
findSubArray (Arr[], left, right, V, K)
{
    nearestCount = 0
    closestIndex = BinarySearch (Arr, left, right, V)
    // binary search function returns closest index
       or the exact index
    nearestCount ++
    i = closestIndex - 1
    j = closestIndex + 1
```

22L-6552 4H

```
while(nearestCount<K and i>= left and j<= right)
{
    diff1= AbsDiff (V, Arr[i])
    diff2= AbsDiff (V, Arr[j])
    if (diff1 <= diff2)
    {
        i--
    }
    else
    {
        j++
    }
    nearestCount++
}
```

```
while (nearestCount < K and i >= left)
{
    i--
    nearestCount++
}

while (nearestCount < K and j <= right)
{
    j++
    nearestCount++
}

start = i++
end = j--
return (start, end)
}
```

22L-6552 4H

```
BinarySearch(Arr[], left, right, searchKey)
{
    while(left <= right)
    {
        mid = (left + right)/2
        if(Arr[mid] == searchKey)
        {
            return mid
        }
        else if(Arr[mid] < searchKey)
        {
            left = mid + 1
        }
        else
        {
            right = mid - 1
        }
    }
    return mid
}
```