Design and Algorithm (Assignment # 2)

Zaeem Yousaf (19L-1196 4A)

Submitted To Maryam Bashir

Contents

1 Description

- Roll NO: 19L-1196
- Section: 4A
- Department: BSCS
- Subject: Design and Algorithm

2 Q:1

2.1 Summary O(n $\log_2$ M)

1. mergeSort(Array,size) O(n*$\log_2$ n)

2. BinarySearch(N,N[i]+y) each element O(n*$\log_2$ n)

combined O(n $\log_2$ n)

```
void find_difference(N,y){
  for(int i=0; i< n && i+y <= N[n]; i++){
    //O(n)
    if(binary_search(N,0,n,i+y) != -1) cout << i << "& " << y << endl; //O(log_2 n)
    }
   }
}

  merge(array,left,middle,right){
    //O(N)
  }


  merge_sort(array,left,right){
    if(left < right){
      int middle = (left+right)/2;
      merge_sort(array,left,middle);
      merge_sort(array,middle+1,right);
      merge(array,left,middle,right);
    }
  }


  int binary_search(int array[], int left, int right, int x)
  // O(log_2 n)
  {
    if (right >= left) {
      int mid = left + (right - left) / 2;
      if (array[mid] == x)
return mid;
      if (array[mid] > x)
return binarySearch(array, left, mid - 1, x);
      return binarySearch(array, mid + 1, right, x);
    }
    return -1;
  }
```

3 Q: 2

3.1 Summary T(n) = N $\log_2$ M

1. Search each element of 'N' in 'M' in O($\log_2$ M)

2. Transfer elements of 'N' into 'C' first if it is found the smallest by binary seach

3. if $\text{location}_{\text{in M}}$ returns > 0, some element/s of 'M' will be copied till the 'location'

4. T(n) = N*$\log_2$ M

5. N elements will be Binary searched in M Each time the size of M will decrease to search But in worst case the size of M remains constant the T(n) will be N * $\log_2$ M O(N*$\log_2$ M)

```
void merge_two_array(M,N,C){
  int C[M.size+N.size];
  int c_index=0;
  int m_index=0;
  for(int i=0; i < N.size; i++){
    int loc = location_in_M(M,m_index,M.size-1,i); // O(lg n) binary search
    if(loc == 0) C[c_index++] = i;
    else{
      // only M.size times
      for(int m_i=m_index; m_i < loc; m_i++) C[c_index++] = M[m_i]
    }
  }
}

  //-------------------------------------
    location_in_M(M,a,b,n){
      // find the location of 'n' in array 'M'
      // perforem search between 'a=0' and 'b=size-1' indices
      if(a == b && n > M[a] ) return a+1;
      else if(a == b && n < M[a]) return a;
      else if(a < b){
int mid = (a+b)/2;
if(M[mid] == n) return mid;
else if(n > M[mid]){
```

```
  location_in_M(M,mid+1,b,n);
}
else{
  location_in_M(M,a,mid-1,n);
}
      }
    }
```

4 Q: 3

4.1 Summary O(lg n)

1. Target is to find the index of greatest 'element' from array
2. if middle is greatest from surrounding and three elements are not in order then middle is greatest
3. if middle, middle's left and middle's right are in ascending order then greates is on right side
4. else greatest element is on left side

- $T(n) = T(n/2) + 1$
- $T(n) = 1 + 1 + 1 + \ldots\ 1 = k*1$
- $n = 2^k$
- $k = \log_2 n$
- $T(n) = \log_2 n$
- $O(\lg n)$

```
findRotation(array,a,b){
  if(a <= b){
    if(a == b) return 0; // base_case
    else if(b-a == 2){
      if(array[a] > array[b]) return a; // base case_2
      else{return b;}
    }
    else if(b-a > 2){
      int middle = (b-a)/2;
```

```
      if(array[middle] > array[middle-1] && array[middle-1] > array[middle+1]){
      return middle;
      }
      else if(array[a] > array[b] && array[a] > array[middle]){
findRotation(array,a,middle-1);
      }
      else{
return (middle + findRotation(array,middle,b) ; // base case_3
      }
    }
  }
  else{
    return 0;
  }
}
```

5 Q: 4

5.1 Summary O(lg n) Almost same as Q # 3

1. Target is to find the greatest 'element' from array
2. if middle, middle's left and middle's right are in ascending ord descending order then greates is on right/left side

   - T(n) = T(n/2) + 1
   - T(n) = 1 + 1 + 1 + . . . 1 = k*1
   - n = 2^k
   - k = $\log_2$ n
   - T(n) = $\log_2$ n
   - O(lg n)

```
maximum(array,a,b){
  if(a <= b){
    if(a == b) return array[a]; // base_case
    else if(b-a == 2){
      if(array[a] > array[b]) return array[a]; // base case_2
      else{return array[b];}
```

```
    }
    else if(b-a > 2){
      int middle = (b-a)/2;
      if(array[middle-1] > array[middle] && array[middle] > array[middle+1]){
maximum(array,middle,b);
      }
      else if(array[middle-1] < array[middle] && array[middle] < array[middle+1]){
maximum(array,0,middle-1);
      }
      else{
return middle; // base case_3
      }
    }
  }
  else{
    return 0;
  }
}
```