CS5800: Algorithms Spring 2018
Assignment 2.2

Saptaparna Das

January 29, 2018

2.(a) The question says, we have k security counters and each can accomodate n passengers. These passengers stand acording to priority. Basically, there are k sets of sorted arrays, where each array contains n elements. Now to sort all passengers according to priority, we can merge first two arrays, then the result with third array, as given in question. So, for merging two sorted arrays of length n1, n2, we need O(n1+n2) time. This is done in below steps:
1. Create a new array of size n1+n2;
2. Traverse both arrays simultaneously. Copy smaller of the two current elements in new array. Advance the pointer of that array.
3. If any of the array is still not empty, copy the elements directly in new array.

For 3rd array containing n3 elements, the time required to merge it with result of ist 2 arrays will be O(n1+n2+n3) and so on..
So, total time required to merge k arrays containing n elements would be.. $2n + 3n + 4n + ...kn = O(k^2n)$.

2.(b) Pseudo code:

```
// Usual merge function of merge sort
// which takes two zero indexed sorted arrays
// and merge them
merge(a1[0..n-1],a2[0..n-1]){
        n1=a1.length
        n2=a2.length
        arr[n1+n2]
        i=0,j=0,k=0
        while(i<n1 and j<n2){
                if(a1[i]<=a2[j]) arr[k++]=a1[i++]
                else arr[k++]=a2[j++]
        }
        while(i<n1){
                arr[k++]=a1[i++]
        }
        while(j<n2){
                arr[k++]=a2[j++]
        }
        return arr
}
// Divide list of arrays into two as opposed to
// dividing one array in two parts, like usual merge sort
mergeSort(a[0...k][0..n]){
        if(a.length==1) return a[0]
        if(a.length==2) merge(a[0],a[1])
        mid=a.length/2

        leftHalf[] = mergeSort(a1[0..mid-1])
        rightHalf[] = mergeSort(a2[mid..k])

        result[] = merge(leftHalf,rightHalf)
        return result

}
```

Here, for more than 2 arrays we are recursively partitioning original input into half size, so that it ends up with 1 or 2 arrays. If it is left with 2 arrays, it calls usual merge function to merge them. If it is left with one array that means it is already sorted. It returns the same. So, total number of partitions is $\log k$. The merge function processes total of nk elements. So, the time complexity is $O(nk \log k)$.