

Design and Analysis of Algorithms

Assignment No. 4

Group members:

Harsh Meena -IIT2019005

Asha Jyothi Donga -IIT2019006

Sampada Kathar -IIT2019007

Find the kth largest element in a merged array of 2 sorted arrays.

Problem:

*Design an algorithm to find the location
of kth sorted element in from
two sorted arrays*

Algorithm1

1. Compare the middle elements of arrays `arr1` and `arr2`, let us call these indices `mid1` and `mid2` respectively.
2. Let us assume `arr1[mid1] < k`, then clearly the elements after `mid2` cannot be the required element.
3. Set the last element of `arr2` to be `arr2[mid2]`.
4. In this way, define a new subproblem with half the size of one of the arrays.

Pseudo Code :

```
int kth(int *arr1,int *arr2,int *end1,int
        *end2,int k)

    if arr1==end1
        return arr2[k]
    if arr2==end2
        return arr1[k]

    int mid1=(end1-arr1) / 2
    int mid2=(end2-arr2) / 2

    if mid1+mid2 < k
        if arr1[mid1] > arr2[mid2]
```

```
        return kth(arr1, arr2+mid2+1,
                    end1, end2,
                    k-mid2-1)
    else
        return kth(arr1+mid1+1, arr2,
                    end1, end2,
                    k-mid1-1)

    else
        if arr1[mid1] > arr2[mid2]
            return kth(arr1, arr2, arr1+mid1,
                        end2, k)
        else
            return kth(arr1, arr2, end1,
                        arr2+mid2, k)
```

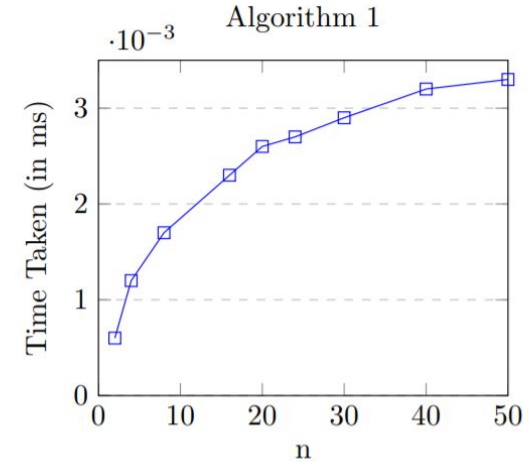
Time Complexity and Space Complexity Analysis of Algorithm 1

Time Complexity

Using this recursive divide and conquer approach, the function is called a total of $\log m + \log n$ times. Thus the worst case complexity of this algorithm will be $O(\log m + \log n)$. The best case occurs when either m or n is equal to 1, i.e., one of the arrays contains only a single element. Thus, the best case time complexity is $\Omega(1)$

Space Complexity

This algorithm has a space complexity of $O(\log k)$



The above graph shows the variance of Time with respect to size of the second array, and follows our theoretically calculated TC of $O(\log(m) + \log(n))$.

Algorithm 2

1. To find the index of the k th greatest element for the given Binary Max heap Array, we will use the same above algorithm and instead of dividing the segment in $m/2$ and $n/2$ and recursing again, we divide them both by $k/2$ and then recurse.
2. define a new subproblem with $k/2$ the size of one of the arrays.

Pseudo Code :

```
int Kth(int arr1[],int arr2[],int m,int n,
        int k,int in1,int in2)

    int current = k/2
    if in1==m
        return arr2[in2+k-1]

    if in2==n
        return arr1[in1+k-1]

    if k==0 or k>((m-in1)+(n-in2))
        return -1

    if k==1
        int x=arr1[in1]<arr2[in2]?
            arr1[in1]:arr2[in2]
        return x

    if current-1 >= m-in1
        if arr1[m-1]<arr2[in2+current-1]
            return arr2[in2+(k-(m-in1)-1)]
        else
            return kth(arr1,arr2,m,n,k-current,
                in1,in2+current);

    if current-1 >= n-in2
        if arr2[n-1]<arr1[in1+current-1]
            return arr1[in1+(k-(n-in2)-1)]
        else
            return kth(arr1,arr2,m,n,k-current,
                in1+current,in2)
    }else{
        if arr1[current+in1-1]<arr2[in2+
            current-1]

            return kth(arr1,arr2,m,n,k-current,
                in1+current,in2)
        else
            return kth(arr1,arr2,m,n,k-current,
                in1,in2+current)
    }
}
```

```
if current-1 >= n-in2
    if arr2[n-1]<arr1[in1+current-1]
        return arr1[in1+(k-(n-in2)-1)]
    else
        return kth(arr1,arr2,m,n,k-current,
            in1+current,in2)
}else{
    if arr1[current+in1-1]<arr2[in2+
        current-1]

        return kth(arr1,arr2,m,n,k-current,
            in1+current,in2)
    else
        return kth(arr1,arr2,m,n,k-current,
            in1,in2+current)
}
}
```

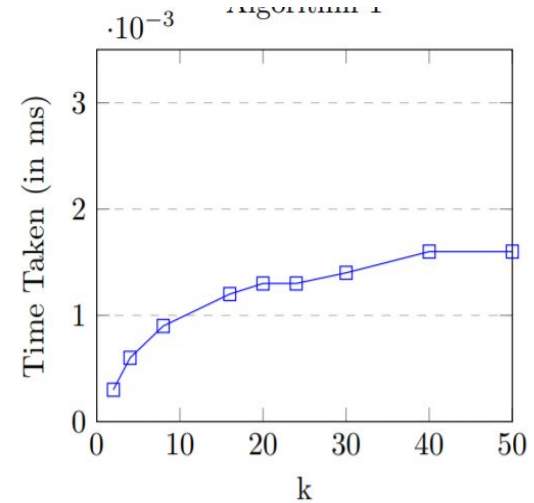
Time Complexity and Space Complexity Analysis of Algorithm 2

Time Complexity

Using this recursive divide and conquer approach, the function is called a total of $\log k$ times. Thus the worst case complexity of this algorithm will be $O(\log k)$. The best case occurs when either m or n is equal to 0, or k is equal to 0. Thus, the best case time complexity is $\Omega(1)$

Space Complexity

This algorithm has a space complexity of $O(\log k)$



The above graph shows the variance of Time with respect to the value of k , and follows our theoretically calculated TC of $O(\log(k))$.

Conclusion

we can conclude that the second algorithm devised by us is the more efficient one for our purposes. It is much more efficient than our first algorithm in terms of both time and memory, with a worst case time complexity of only $O(\log(k))$ and a space complexity of $O(\log(k))$

References:

- [1]vaibhav29498, 'K-th Greatest Element in a two sorted arrays', GeeksforGeeks, 2018. [Online] [Accessed: 1-Feb-2021]
- [2]'Divide and Conquer', tutorialspoint. [Online][Accessed: 1-Feb-2021]
- [3]GeeksforGeeks, 'Divide and Conquer — Set 1 (Introduction)', GeeksforGeeks, 2018. [Online][Accessed: 1-Feb-2021]
- [4]https://lh3.googleusercontent.com/proxy/mKO_ixx5uUiB93Juy0xjQx7sPvRWmYmZB4EwBpoVn97ZTaED_BMriwLtjjmttoL9_ttqoRbfFRlHeTsMUJEHkWYfqEOjqK2iB2PR-HdTiNVFGZoQmX104waXu-HrFEfksEks8OjUjAbrwLy5GADZTU