K'th Smallest/Largest Element in Unsorted Array | Set 2 (Expected Linear Time)

We recommend to read following post as a prerequisite of this post.

K'th Smallest/Largest Element in Unsorted Array | Set 1

Given an array and a number k where k is smaller than size of array, we need to find the k'th smallest element in the given array. It is given that ll array elements are distinct.

Examples:

```
Input: arr[] = {7, 10, 4, 3, 20, 15}
       k = 3
Output: 7

Input: arr[] = {7, 10, 4, 3, 20, 15}
       k = 4
Output: 10
```

We have discussed three different solutions here.

In this post method 4 is discussed which is mainly an extension of method 3 (QuickSelect) discussed in the previous post. The idea is to randomly pick a pivot element. To implement randomized partition, we use a random function, rand() to generate index between l and r, swap the element at randomly generated index with the last element, and finally call the standard partition process which uses last element as pivot.

Following is implementation of above Randomized QuickSelect.

C/C++

```
// C++ implementation of randomized quickSelect
#include<iostream>
#include<climits>
#include<cstdlib>
using namespace std;

int randomPartition(int arr[], int l, int r);

// This function returns k'th smallest element in arr[l..r] using
// QuickSort based method. ASSUMPTION: ELEMENTS IN ARR[] ARE DISTINCT
int kthSmallest(int arr[], int l, int r, int k)
{
```

```
    // If k is smaller than number of elements in array
    if (k > 0 && k <= r - l + 1)
    {
        // Partition the array around a random element and
        // get position of pivot element in sorted array
        int pos = randomPartition(arr, l, r);

        // If position is same as k
        if (pos-l == k-1)
            return arr[pos];
        if (pos-l > k-1)  // If position is more, recur for left subarray
            return kthSmallest(arr, l, pos-1, k);

        // Else recur for right subarray
        return kthSmallest(arr, pos+1, r, k-pos+l-1);
    }

    // If k is more than number of elements in array
    return INT_MAX;
}

void swap(int *a, int *b)
{
    int temp = *a;
    *a = *b;
    *b = temp;
}

// Standard partition process of QuickSort().  It considers the last
// element as pivot and moves all smaller element to left of it and
// greater elements to right. This function is used by randomPartition()
int partition(int arr[], int l, int r)
{
    int x = arr[r], i = l;
    for (int j = l; j <= r - 1; j++)
    {
        if (arr[j] <= x)
        {
            swap(&arr[i], &arr[j]);
            i++;
        }
    }
    swap(&arr[i], &arr[r]);
    return i;
}

// Picks a random pivot element between l and r and partitions
// arr[l..r] arount the randomly picked element using partition()
int randomPartition(int arr[], int l, int r)
{
    int n = r-l+1;
    int pivot = rand() % n;
    swap(&arr[l + pivot], &arr[r]);
    return partition(arr, l, r);
}

// Driver program to test above methods
int main()
{
    int arr[] = {12, 3, 5, 7, 4, 19, 26};
    int n = sizeof(arr)/sizeof(arr[0]), k = 3;
    cout << "K'th smallest element is " << kthSmallest(arr, 0, n-1, k);
    return 0;
}
```

Run on IDE

Java

```
// Java program to find k'th smallest element in expected
// linear time
```

```
class KthSmallst
{
    // This function returns k'th smallest element in arr[l..r]
    // using QuickSort based method.  ASSUMPTION: ALL ELEMENTS
    // IN ARR[] ARE DISTINCT
    int kthSmallest(int arr[], int l, int r, int k)
    {
        // If k is smaller than number of elements in array
        if (k > 0 && k <= r - l + 1)
        {
            // Partition the array around a random element and
            // get position of pivot element in sorted array
            int pos = randomPartition(arr, l, r);

            // If position is same as k
            if (pos-l == k-1)
                return arr[pos];

            // If position is more, recur for left subarray
            if (pos-l > k-1)
                return kthSmallest(arr, l, pos-1, k);

            // Else recur for right subarray
            return kthSmallest(arr, pos+1, r, k-pos+l-1);
        }

        // If k is more than number of elements in array
        return Integer.MAX_VALUE;
    }

    // Utility method to swap arr[i] and arr[j]
    void swap(int arr[], int i, int j)
    {
        int temp = arr[i];
        arr[i] = arr[j];
        arr[j] = temp;
    }

    // Standard partition process of QuickSort().  It considers
    // the last element as pivot and moves all smaller element
    // to left of it and greater elements to right. This function
    // is used by randomPartition()
    int partition(int arr[], int l, int r)
    {
        int x = arr[r], i = l;
        for (int j = l; j <= r - 1; j++)
        {
            if (arr[j] <= x)
            {
                swap(arr, i, j);
                i++;
            }
        }
        swap(arr, i, r);
        return i;
    }

    // Picks a random pivot element between l and r and
    // partitions arr[l..r] arount the randomly picked
    // element using partition()
    int randomPartition(int arr[], int l, int r)
    {
        int n = r-l+1;
        int pivot = (int)(Math.random()) % n;
        swap(arr, l + pivot, r);
        return partition(arr, l, r);
    }

    // Driver method to test above
    public static void main(String args[])
    {
        KthSmallst ob = new KthSmallst();
        int arr[] = {12, 3, 5, 7, 4, 19, 26};
        int n = arr.length,k = 3;
```

```
        System.out.println("K'th smallest element is "+
                           ob.kthSmallest(arr, 0, n-1, k));
    }
}
/*This code is contributed by Rajat Mishra*/
```

Run on IDE

Output:

```
K'th smallest element is 5
```

Time Complexity:

The worst case time complexity of the above solution is still $O(n^2)$. In worst case, the randomized function may always pick a corner element. The expected time complexity of above randomized QuickSelect is $\Theta(n)$, see CLRS book or MIT video lecture for proof. The assumption in the analysis is, random number generator is equally likely to generate any number in the input range.

Sources:

MIT Video Lecture on Order Statistics, Median

Introduction to Algorithms by Clifford Stein, Thomas H. Cormen, Charles E. Leiserson, Ronald L.

This article is contributed by **Shivam**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

GATE CS Corner Company Wise Coding Practice

Randomized Searching Order-Statistics Quick Sort

Recommended Posts:

K'th Smallest/Largest Element in Unsorted Array | Set 1
K'th Smallest/Largest Element in Unsorted Array | Set 3 (Worst Case Linear Time)
Find the closest pair from two sorted arrays
k largest(or smallest) elements in an array | added Min Heap method
K'th Smallest/Largest Element in Unsorted Array | Set 2 (Expected Linear Time)

(Login to Rate and Mark)

3.5 Average Difficulty : **3.5/5.0**
Based on **42** vote(s)

Add to TODO List

Mark as DONE

Writing code in comment? Please use ide.geeksforgeeks.org, generate link and share the link here.

Load Comments

Share this post!