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 a similar problem to print k largest elements.

Recommended: Please solve it on “*PRACTICE* ” first, before moving on to the solution.

Method 1 (Simple Solution)

A Simple Solution is to sort the given array using a O(nlogn) sorting algorithm like Merge Sort, Heap Sort, etc and return the element at index k-1 in the sorted array. Time Complexity of this solution is O(nLogn).

C++

```
// Simple C++ program to find k'th smallest element
#include<iostream>
#include<algorithm>
using namespace std;

// Function to return k'th smallest element in a given array
int kthSmallest(int arr[], int n, int k)
{
    // Sort the given array
}

// Driver program to test above methods
int main()
{
    int arr[] = {12, 3, 5, 7, 19};
    int n = sizeof(arr)/sizeof(arr[0]), k = 2;
    cout << "K'th smallest element is " <<  kthSmallest(arr, n, k);
    return 0;
}
```

Run on IDE Copy Code

Java

```
// Java code for kth smallest element
// in an array
import java.util.Arrays;
import java.util.Collections;

class GFG
{
    // Function to return k'th smallest
    // element in a given array
    public static int kthSmallest(Integer [] arr,
                                         int k)
    {
        // Sort the given array
        Arrays.sort(arr);

        // Return k'th element in
        // the sorted array
        return arr[k-1];
    }

    // driver program
    public static void main(String[] args)
    {
        Integer arr[] = new Integer[]{12, 3, 5, 7, 19};
        int k = 2;
        System.out.print( "K'th smallest element is "+
                            kthSmallest(arr, k) );
    }
}

// This code is contributed by Chhavi
```

Run on IDE Copy Code

C#

```
// C# code for kth smallest element
// in an array
using System;

class GFG {

    // Function to return k'th smallest
    // element in a given array
    public static int kthSmallest(int []arr,
                                      int k)
    {

        // Sort the given array
        return arr[k-1];
    }

    // driver program
    public static void Main()
    {
        int []arr = new int[]{12, 3, 5,
                             7, 19};
        int k = 2;
        Console.Write( "K'th smallest element"
              + " is "+ kthSmallest(arr, k) );
    }
}

// This code is contributed by nitin mittal.
```

Run on IDE Copy Code

PHP

```
<?php
// Simple PHP program to find
// k'th smallest element

// Function to return k'th smallest
// element in a given array
function kthSmallest($arr, $n, $k)
{

    // Sort the given array
    sort($arr);

    // Return k'th element
    // in the sorted array
    return $arr[$k - 1];
}

    // Driver Code
    $arr = array(12, 3, 5, 7, 19);
    $n =count($arr);
    $k = 2;
    echo "K'th smallest element is "
       , kthSmallest($arr, $n, $k);

// This code is contributed by anuj_67.
?>
```

Run on IDE Copy Code

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

Method 2 (Using Min Heap – HeapSelect)

We can find k'th smallest element in time complexity better than O(nLogn). A simple optomization is to create a Min Heap of the given n elements and call extractMin() k times.

The following is C++ implementation of above method.

```
using namespace std;

// Prototype of a utility function to swap two integers
void swap(int *x, int *y);

// A class for Min Heap
class MinHeap
{
    int *harr; // pointer to array of elements in heap
    int capacity; // maximum possible size of min heap
    int heap_size; // Current number of elements in min heap
public:
    MinHeap(int a[], int size); // Constructor
    void MinHeapify(int i);  //To minheapify subtree rooted with index i
    int parent(int i) { return (i-1)/2; }
    int left(int i) { return (2*i + 1); }
    int right(int i) { return (2*i + 2); }

    int extractMin();  // extracts root (minimum) element
    int getMin() { return harr[0]; } // Returns minimum
};

MinHeap::MinHeap(int a[], int size)
{
    heap_size = size;
    harr = a;  // store address of array
    int i = (heap_size - 1)/2;
    while (i >= 0)
    {
        MinHeapify(i);
        i--;
    }
}

// Method to remove minimum element (or root) from min heap
int MinHeap::extractMin()
{
    if (heap_size == 0)
        return INT_MAX;

    // Store the minimum vakue.
    int root = harr[0];

    // If there are more than 1 items, move the last item to root
    // and call heapify.
    if (heap_size > 1)
    {
        harr[0] = harr[heap_size-1];
        MinHeapify(0);
    }
    heap_size--;

    return root;
}

// A recursive method to heapify a subtree with root at given index
// This method assumes that the subtrees are already heapified
void MinHeap::MinHeapify(int i)
{
    int l = left(i);
    int r = right(i);
    int smallest = i;
    if (l < heap_size && harr[l] < harr[i])
        smallest = l;
    if (r < heap_size && harr[r] < harr[smallest])
        smallest = r;
    if (smallest != i)
    {
        swap(&harr[i], &harr[smallest]);
        MinHeapify(smallest);
```

```
void swap(int *x, int *y)
{
    int temp = *x;
    *x = *y;
    *y = temp;
}

// Function to return k'th smallest element in a given array
int kthSmallest(int arr[], int n, int k)
{
    // Build a heap of n elements: O(n) time
    MinHeap mh(arr, n);

    // Do extract min (k-1) times
    for (int i=0; i<k-1; i++)
        mh.extractMin();

    // Return root
    return mh.getMin();
}

// Driver program to test above methods
int main()
{
    int arr[] = {12, 3, 5, 7, 19};
    int n = sizeof(arr)/sizeof(arr[0]), k = 2;
    cout << "K'th smallest element is " << kthSmallest(arr, n, k);
    return 0;
}
```

Run on IDE Copy Code

Output:

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

Time complexity of this solution is O(n + kLogn).

Method 3 (Using Max-Heap)

We can also use Max Heap for finding the k'th smallest element. Following is algorithm.

1) Build a Max-Heap MH of the first k elements (arr[0] to arr[k-1]) of the given array. O(k)

2) For each element, after the k'th element (arr[k] to arr[n-1]), compare it with root of MH.

……a) If the element is less than the root then make it root and call heapify for MH

……b) Else ignore it.

// The step 2 is O((n-k)*logk)

3) Finally, root of the MH is the kth smallest element.

Time complexity of this solution is O(k + (n-k)*Logk)

The following is C++ implementation of above algorithm

```
// A C++ program to find k'th smallest element using max heap
#include<iostream>
#include<climits>
using namespace std;
```

```
// A class for Max Heap
class MaxHeap
{
    int *harr; // pointer to array of elements in heap
    int capacity; // maximum possible size of max heap
    int heap_size; // Current number of elements in max heap
public:
    MaxHeap(int a[], int size); // Constructor
    void maxHeapify(int i);  //To maxHeapify subtree rooted with index i
    int parent(int i) { return (i-1)/2; }
    int left(int i) { return (2*i + 1); }
    int right(int i) { return (2*i + 2); }

    int extractMax();  // extracts root (maximum) element
    int getMax() { return harr[0]; } // Returns maximum

    // to replace root with new node x and heapify() new root
    void replaceMax(int x) { harr[0] = x;  maxHeapify(0); }
};

MaxHeap::MaxHeap(int a[], int size)
{
    heap_size = size;
    harr = a;  // store address of array
    int i = (heap_size - 1)/2;
    while (i >= 0)
    {
        maxHeapify(i);
        i--;
    }
}

// Method to remove maximum element (or root) from max heap
int MaxHeap::extractMax()
{
    if (heap_size == 0)
        return INT_MAX;

    // Store the maximum vakue.
    int root = harr[0];

    // If there are more than 1 items, move the last item to root
    // and call heapify.
    if (heap_size > 1)
    {
        harr[0] = harr[heap_size-1];
        maxHeapify(0);
    }
    heap_size--;

    return root;
}

// A recursive method to heapify a subtree with root at given index
// This method assumes that the subtrees are already heapified
void MaxHeap::maxHeapify(int i)
{
    int l = left(i);
    int r = right(i);
    int largest = i;
    if (l < heap_size && harr[l] > harr[i])
        largest = l;
    if (r < heap_size && harr[r] > harr[largest])
        largest = r;
    if (largest != i)
    {
        swap(&harr[i], &harr[largest]);
        maxHeapify(largest);
    }
```

```
{
    int temp = *x;
    *x = *y;
    *y = temp;
}

// Function to return k'th largest element in a given array
int kthSmallest(int arr[], int n, int k)
{
    // Build a heap of first k elements: O(k) time
    MaxHeap mh(arr, k);

    // Process remaining n-k elements.  If current element is
    // smaller than root, replace root with current element
    for (int i=k; i<n; i++)
        if (arr[i] < mh.getMax())
           mh.replaceMax(arr[i]);

    // Return root
    return mh.getMax();
}

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

Run on IDE Copy Code

Output:

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

Method 4 (QuickSelect)

This is an optimization over method 1 if QuickSort is used as a sorting algorithm in first step. In QuickSort, we pick a pivot element, then move the pivot element to its correct position and partition the array around it. The idea is, not to do complete quicksort, but stop at the point where pivot itself is k’th smallest element. Also, not to recur for both left and right sides of pivot, but recur for one of them according to the position of pivot. The worst case time complexity of this method is O(n^2), but it works in O(n) on average.

C++

```
#include<iostream>
#include<climits>
using namespace std;

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

// 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)
{
        // position of pivot element in sorted array
        int pos = partition(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
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;
}

// 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 Copy Code

Java

```
// Java code for kth smallest element in an array
import java.util.Arrays;
import java.util.Collections;

class GFG
{
    // 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
    public static int partition(Integer [] arr, int l,
                                              int r)
```

```
            if (arr[j] <= x)
            {
                //Swapping arr[i] and arr[j]
                int temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;

                i++;
            }
        }

        //Swapping arr[i] and arr[r]
        int temp = arr[i];
        arr[i] = arr[r];
        arr[r] = temp;

        return i;
    }

    // This function returns k'th smallest element
    // in arr[l..r] using QuickSort based method.
    // ASSUMPTION: ALL ELEMENTS IN ARR[] ARE DISTINCT
    public static int kthSmallest(Integer[] 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 last
            // element and get position of pivot
            // element in sorted array
            int pos = partition(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;
    }

    // Driver program to test above methods
    public static void main(String[] args)
    {
        Integer arr[] = new Integer[]{12, 3, 5, 7, 4, 19, 26};
        int k = 3;
        System.out.print( "K'th smallest element is " +
                     kthSmallest(arr, 0, arr.length - 1, k) );
    }
}

// This code is contributed by Chhavi
```

Run on IDE Copy Code

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

There are two more solutions which are better than above discussed ones: One solution is to do randomized version of quickSelect() and other solution is worst case linear time algorithm (see the following posts).

K'th Smallest/Largest Element in Unsorted Array | Set 2 (Expected Linear Time)
K'th Smallest/Largest Element in Unsorted Array | Set 3 (Worst Case Linear Time)

References:
http://www.ics.uci.edu/~eppstein/161/960125.html
http://www.cs.rit.edu/~ib/Classes/CS515_Spring12-13/Slides/022-SelectMasterThm.pdf

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Recommended Posts:

- K'th Smallest/Largest Element using STL
- Trapping Rain Water
- Find the largest three elements in an array
- K'th largest element in a stream
- K'th Smallest/Largest Element in Unsorted Array | Set 3 (Worst Case Linear Time)
- K'th Smallest/Largest Element in Unsorted Array | Set 2 (Expected Linear Time)
- K'th Smallest/Largest Element in Unsorted Array | Set 2 (Expected Linear Time)
- Find the closest pair from two sorted arrays
- Binary Heap
- Program to find largest element in an array
- Kth smallest element in a row-wise and column-wise sorted 2D array | Set 1
- HeapSort
- Sort a nearly sorted (or K sorted) array
- Time Complexity of building a heap
- k largest(or smallest) elements in an array | added Min Heap method

Improved By : nitin mittal, vt_m

Article Tags : Arrays Heap Searching ABCO Cisco Microsoft Order-Statistics Rockstand SAP Labs VMWare

It's teamwork, but simpler, more pleasant and more productive.

GeeksforGeeks
A computer science portal for geeks
TECHNICAL
SCRIPTER
2 OCT to 2 DEC
A technical content writing event by GeeksforGeeks

Geeks Classes
Classroom program on Algorithms in Noida
Mentored by Mr Sandeep Jain
Batch starts from 27 Oct, 2018

1

3.1

☐ To-do ☐ Done

Based on **183** vote(s)

Feedback Add Notes Improve Article

Please write to us at contribute@geeksforgeeks.org to report any issue with the above content.

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

Load Comments Share this post!