K’th Smallest Element - GeeksforGeeks

Source: https://www.geeksforgeeks.org/kth-smallest-largest-element-in-unsorted-array/

Courses Tutorials Practice Jobs DSA Tutorial Interview Questions Quizzes Must Do Advanced DSA System Design Aptitude Puzzles Interview Corner DSA Python Technical Scripter 2026 Explore DSA Fundamentals Logic Building Problems Analysis of Algorithms Data Structures Array Data Structure String in Data Structure Hashing in Data Structure Linked List Data Structure Stack Data Structure Queue Data Structure Tree Data Structure Graph Data Structure Trie Data Structure Algorithms Searching Algorithms Sorting Algorithms Introduction to Recursion Greedy Algorithms Tutorial Graph Algorithms Dynamic Programming or DP Bitwise Algorithms Advanced Segment Tree Binary Indexed Tree or Fenwick Tree Square Root (Sqrt) Decomposition Algorithm Binary Lifting Geometry Interview Preparation Interview Corner GfG160 Practice Problem GeeksforGeeks Practice - Leading Online Coding Platform Problem of The Day - Develop the Habit of Coding DSA Course 90% Refund K’th Smallest Element Last Updated : 18 Oct, 2025 Given an integer array arr[] and k. Find the k'th smallest element in the given array. Note: k is always smaller than the size of the array. Examples: Input : arr[] = [10, 5, 4, 3, 48, 6, 2, 33, 53, 10], k = 4 Output : 5 Explanation: 4th smallest element in the given array is 5. Input : arr[] = [7, 10, 4, 3, 20, 15], k = 3 Output : 7 Explanation: 3rd smallest element in the given array is 7. Try it on GfG Practice Table of Content [Naive Approach] Using Sorting - O(n log(n)) Time and O(1) Space [Expected Approach] Using Max-Heap - O(n * log(k)) Time and O(k) Space [Alternative Approach 1] Using QuickSelect [Alternative Approach 2] Using Counting Sort [Naive Approach] Using Sorting - O(n log(n)) Time and O(1) Space The idea is to sort the given array and return the element at the index k - 1. C++ //Driver Code Starts #include <iostream> #include <algorithm> #include <vector> using namespace std ; //Driver Code Ends int kthSmallest (vector < int >& arr , int k) { // Sort the given vector sort (arr . begin (), arr . end ()); // Return k'th element in the sorted vector return arr [k - 1]; } //Driver Code Starts int main () { vector < int > arr = { 10 , 5 , 4 , 3 , 48 , 6 , 2 , 33 , 53 , 10 }; int k = 4 ; cout << kthSmallest (arr , k); return 0 ; } //Driver Code Ends Java //Driver Code Starts import java.util.Arrays ; class GFG { //Driver Code Ends static int kthSmallest (int [] arr , int k) { // Sort the given array Arrays . sort (arr); // Return k'th element in the sorted array return arr [k - 1] ; } //Driver Code Starts public static void main (String [] args) { int [] arr = { 10 , 5 , 4 , 3 , 48 , 6 , 2 , 33 , 53 , 10 }; int k = 4 ; System . out . println (kthSmallest (arr , k)); } } //Driver Code Ends Python def kthSmallest (arr , k): # Sort the given vector arr . sort () # Return k'th element in the sorted vector return arr [k - 1] #Driver Code Starts if __name__ == "__main__" : arr = [10 , 5 , 4 , 3 , 48 , 6 , 2 , 33 , 53 , 10] k = 4 print (kthSmallest (arr , k)) #Driver Code Ends C# //Driver Code Starts using System ; class GFG //Driver Code Ends { static int kthSmallest (int [] arr , int k) { // Sort the given array Array . Sort (arr); // Return k'th element in the sorted array return arr [k - 1]; } //Driver Code Starts static void Main () { int [] arr = { 10 , 5 , 4 , 3 , 48 , 6 , 2 , 33 , 53 , 10 }; int k = 4 ; Console . WriteLine (kthSmallest (arr , k)); } } //Driver Code Ends JavaScript function kthSmallest (arr , k) { // Sort the given vector arr . sort ((a , b) => a - b); // Return k'th element in the sorted vector return arr [k - 1]; } //Driver Code //Driver Code Starts let arr = [10 , 5 , 4 , 3 , 48 , 6 , 2 , 33 , 53 , 10]; let k = 4 ; console . log (kthSmallest (arr , k)); //Driver Code Ends Output 5 [Expected Approach] Using Max-Heap - O(n * log(k)) Time and O(k) Space The idea is to maintain a max heap of size k while iterating through the array. The heap always contains the k smallest elements seen so far. If the heap size exceeds k, remove the largest element. At the end, the heap holds the k smallest elements. C++ //Driver Code Starts #include <iostream> #include <vector> #include <queue> using namespace std ; //Driver Code Ends int kthSmallest (vector < int >& arr , int k) { // Create a max heap priority_queue < int > pq ; // Iterate through the array elements for (int i = 0 ; i < arr . size (); i ++) { // Push the current element onto the max heap pq . push (arr [i]); // If the size of the max heap exceeds k, //remove the largest element if (pq . size () > k) pq . pop (); } return pq . top (); } int main () //Driver Code Starts { vector < int > arr = { 10 , 5 , 4 , 3 , 48 , 6 , 2 , 33 , 53 , 10 }; int k = 4 ; cout << kthSmallest (arr , k); } //Driver Code Ends Java //Driver Code Starts import java.util.PriorityQueue ; import java.util.Collections ; //Driver Code Ends class GFG { static int kthSmallest (int [] arr , int k) { // Create a max heap PriorityQueue < Integer > pq = new PriorityQueue <> (Collections . reverseOrder ()); // Iterate through the array elements for (int val : arr) { // Push the current element onto the max heap pq . add (val); // If the size of the max heap exceeds k, // remove the largest element if (pq . size () > k) pq . poll (); } // Return the kth smallest element (top of the max heap) return pq . peek (); } //Driver Code Starts public static void main (String [] args) { int [] arr = { 10 , 5 , 4 , 3 , 48 , 6 , 2 , 33 , 53 , 10 }; int k = 4 ; System . out . println (kthSmallest (arr , k)); } } //Driver Code Ends Python #Driver Code Starts import heapq #Driver Code Ends def kthSmallest (arr , k): # Create a max heap pq = [] # Iterate through the array elements for i in range (len (arr)): # Push the current element onto the max heap heapq . heappush (pq , - arr [i]) # If the size of the max heap exceeds k, #remove the largest element if len (pq) > k : heapq . heappop (pq) return - pq [0] #Driver Code Starts if __name__ == '__main__' : arr = [10 , 5 , 4 , 3 , 48 , 6 , 2 , 33 , 53 , 10] k = 4 print (kthSmallest (arr , k)) #Driver Code Ends C# //Driver Code Starts using System ; class MaxHeap { private int [] heap ; private int size ; public MaxHeap (int capacity) { heap = new int [capacity]; size = 0 ; } public int Count => size ; public void Push (int val) { heap [size] = val ; int i = size ; size ++ ; while (i > 0) { int parent = (i - 1) / 2 ; if (heap [parent] >= heap [i]) break ; int temp = heap [parent]; heap [parent] = heap [i]; heap [i] = temp ; i = parent ; } } public int Pop () { if (size == 0) { Console . WriteLine ("Heap is empty"); return - 1 ; } int top = heap [0]; heap [0] = heap [size - 1]; size -- ; int i = 0 ; while (true) { int left = 2 * i + 1 ; int right = 2 * i + 2 ; int largest = i ; if (left < size && heap [left] > heap [largest]) largest = left ; if (right < size && heap [right] > heap [largest]) largest = right ; if (largest == i) break ; int temp = heap [i]; heap [i] = heap [largest]; heap [largest] = temp ; i = largest ; } return top ; } public int Top () { if (size == 0) { Console . WriteLine ("Heap is empty"); return - 1 ; } return heap [0]; } } class GFG { //Driver Code Ends static int kthSmallest (int [] arr , int k) { // Create a max heap MaxHeap pq = new MaxHeap (arr . Length); // Iterate through the array elements for (int i = 0 ; i < arr . Length ; i ++) { // Push the current element onto the max heap pq . Push (arr [i]); // If the size of the max heap exceeds k, //remove the largest element if (pq . Count > k) pq . Pop (); } // Return the kth smallest element (top of the max heap) return pq . Top (); } //Driver Code Starts static void Main () { int [] arr = { 10 , 5 , 4 , 3 , 48 , 6 , 2 , 33 , 53 , 10 }; int k = 4 ; Console . WriteLine (kthSmallest (arr , k)); } } //Driver Code Ends JavaScript //Driver Code Starts class MaxHeap { constructor () { this . heap = []; } get count () { return this . heap . length ; } push (val) { this . heap . push (val); let i = this . heap . length - 1 ; while (i > 0) { let parent = Math . floor ((i - 1) / 2); if (this . heap [parent] >= this . heap [i]) break ; // Swap [this . heap [parent], this . heap [i]] = [this . heap [i], this . heap [parent]]; i = parent ; } } pop () { if (this . heap . length === 0) { console . log ("Heap is empty"); return - 1 ; } let top = this . heap [0]; this . heap [0] = this . heap [this . heap . length - 1]; this . heap . pop (); let i = 0 ; while (true) { let left = 2 * i + 1 ; let right = 2 * i + 2 ; let largest = i ; if (left < this . heap . length && this . heap [left] > this . heap [largest]) largest = left ; if (right < this . heap . length && this . heap [right] > this . heap [largest]) largest = right ; if (largest === i) break ; [this . heap [i], this . heap [largest]] = [this . heap [largest], this . heap [i]]; i = largest ; } return top ; } top () { if (this . heap . length === 0) { console . log ("Heap is empty"); return - 1 ; } return this . heap [0]; } } //Driver Code Ends function kthSmallest (arr , k) { // Create a max heap let pq = new MaxHeap (); // Iterate through the array elements for (let i = 0 ; i < arr . length ; i ++) { // Push the current element onto the max heap pq . push (arr [i]); // If the size of the max heap exceeds k, //remove the largest element if (pq . count > k) pq . pop (); } return pq . top (); } //Driver Code Starts // Driver code let arr = [10 , 5 , 4 , 3 , 48 , 6 , 2 , 33 , 53 , 10]; let K = 4 ; console . log (kthSmallest (arr , K)); //Driver Code Ends Output 5 [Alternative Approach 1] Using QuickSelect The main idea is to use QuickSelect to find the k-th largest element by picking a pivot and partitioning the array so that all elements greater than the pivot are on the left and smaller ones on the right. If the pivot ends up at index k–1, it is the k-th largest element. Otherwise, we recursively search only in the left or right part that contains the k-th largest element. C++ //Driver Code Starts #include <iostream> #include <vector> using namespace std ; //Driver Code Ends int partition (vector < int >& arr , int left , int right) { // Choose the last element as pivot int pivot = arr [right]; int i = left ; // Traverse the array and move elements <= pivot to the left for (int j = left ; j < right ; j ++) { if (arr [j] <= pivot) { // Swap current element with element at i swap (arr [i], arr [j]); i ++ ; } } // Place the pivot in its correct position swap (arr [i], arr [right]); return i ; } // QuickSelect function: recursively finds k-th smallest int quickSelect (vector < int >& arr , int left , int right , int k) { if (left <= right) { // Partition around pivot int pivotIndex = partition (arr , left , right); // Found k-th smallest if (pivotIndex == k) return arr [pivotIndex]; else if (pivotIndex > k) return quickSelect (arr , left , pivotIndex - 1 , k); else return quickSelect (arr , pivotIndex + 1 , right , k); } return -1 ; } int kthSmallest (vector < int >& arr , int k) { return quickSelect (arr , 0 , arr . size () -1 , k -1); } //Driver Code Starts int main () { vector < int > arr = { 10 , 5 , 4 , 3 , 48 , 6 , 2 , 33 , 53 , 10 }; int k = 4 ; cout << kthSmallest (arr , k); } //Driver Code Ends Java //Driver Code Starts class GFG { //Driver Code Ends static int partition (int [] arr , int left , int right) { // Choose the last element as pivot int pivot = arr [right] ; int i = left ; // Traverse the array and move elements <= pivot to the left for (int j = left ; j < right ; j ++) { if (arr [j] <= pivot) { // Swap current element with element at i int temp = arr [i] ; arr [i] = arr [j] ; arr [j] = temp ; i ++ ; } } // Place the pivot in its correct position int temp = arr [i] ; arr [i] = arr [right] ; arr [right] = temp ; return i ; } static int quickSelect (int [] arr , int left , int right , int k) { if (left <= right) { // Partition around pivot int pivotIndex = partition (arr , left , right); // Found k-th smallest if (pivotIndex == k) return arr [pivotIndex] ; else if (pivotIndex > k) return quickSelect (arr , left , pivotIndex - 1 , k); else return quickSelect (arr , pivotIndex + 1 , right , k); } return - 1 ; } static int kthSmallest (int [] arr , int k) { return quickSelect (arr , 0 , arr . length - 1 , k - 1); } //Driver Code Starts public static void main (String [] args) { int [] arr = { 10 , 5 , 4 , 3 , 48 , 6 , 2 , 33 , 53 , 10 }; int k = 4 ; System . out . println (kthSmallest (arr , k)); } } //Driver Code Ends Python def partition (arr , left , right): # Choose the last element as pivot pivot = arr [right] i = left # Traverse the array and move elements <= pivot to the left for j in range (left , right): if arr [j] <= pivot : # Swap current element with element at i arr [i], arr [j] = arr [j], arr [i] i += 1 # Place the pivot in its correct position arr [i], arr [right] = arr [right], arr [i] return i # QuickSelect function: recursively finds k-th smallest def quickSelect (arr , left , right , k): if left <= right : # Partition around pivot pivotIndex = partition (arr , left , right) # Found k-th smallest if pivotIndex == k : return arr [pivotIndex] elif pivotIndex > k : return quickSelect (arr , left , pivotIndex - 1 , k) else : return quickSelect (arr , pivotIndex + 1 , right , k) return - 1 def kthSmallest (arr , k): return quickSelect (arr , 0 , len (arr) - 1 , k - 1) if __name__ == "__main__" : #Driver Code Starts arr = [10 , 5 , 4 , 3 , 48 , 6 , 2 , 33 , 53 , 10] k = 4 print (kthSmallest (arr , k)) #Driver Code Ends C# //Driver Code Starts using System ; class GFG { //Driver Code Ends static int partition (int [] arr , int left , int right) { // Choose the last element as pivot int pivot = arr [right]; int i = left ; // Traverse the array and move elements <= pivot to the left for (int j = left ; j < right ; j ++) { if (arr [j] <= pivot) { // Swap current element with element at i int temp = arr [i]; arr [i] = arr [j]; arr [j] = temp ; i ++ ; } } // Place the pivot in its correct position int tempPivot = arr [i]; arr [i] = arr [right]; arr [right] = tempPivot ; return i ; } static int quickSelect (int [] arr , int left , int right , int k) { if (left <= right) { // Partition around pivot int pivotIndex = partition (arr , left , right); // Found k-th smallest if (pivotIndex == k) return arr [pivotIndex]; else if (pivotIndex > k) return quickSelect (arr , left , pivotIndex - 1 , k); else return quickSelect (arr , pivotIndex + 1 , right , k); } return - 1 ; } static int kthSmallest (int [] arr , int k) { return quickSelect (arr , 0 , arr . Length - 1 , k - 1); } //Driver Code Starts static void Main () { int [] arr = { 10 , 5 , 4 , 3 , 48 , 6 , 2 , 33 , 53 , 10 }; int k = 4 ; Console . WriteLine (kthSmallest (arr , k)); } } //Driver Code Ends JavaScript function partition (arr , left , right) { // Choose the last element as pivot let pivot = arr [right]; let i = left ; // Traverse the array and move elements <= pivot to the left for (let j = left ; j < right ; j ++) { if (arr [j] <= pivot) { // Swap current element with element at i [arr [i], arr [j]] = [arr [j], arr [i]]; i ++ ; } } // Place the pivot in its correct position [arr [i], arr [right]] = [arr [right], arr [i]]; return i ; } function quickSelect (arr , left , right , k) { if (left <= right) { // Partition around pivot let pivotIndex = partition (arr , left , right); // Found k-th smallest if (pivotIndex === k) return arr [pivotIndex]; else if (pivotIndex > k) return quickSelect (arr , left , pivotIndex - 1 , k); else return quickSelect (arr , pivotIndex + 1 , right , k); } return - 1 ; } function kthSmallest (arr , k) { return quickSelect (arr , 0 , arr . length - 1 , k - 1); } // Driver code //Driver Code Starts let arr = [10 , 5 , 4 , 3 , 48 , 6 , 2 , 33 , 53 , 10]; let k = 4 ; console . log (kthSmallest (arr , k)); //Driver Code Ends Output 5 Time Complexity : O(n 2) in the worst case, but on average works in O(n log n) time and performs better than priority queue based algorithm. Auxiliary Space : O(n) for recursion call stack in worst case. On average : O(log n) [Alternative Approach 2] Using Counting Sort The main idea is to use counting sort's frequency counts to track how many elements are smaller or equal to each value, and then directly identify the K'th smallest element from these cumulative counts without fully sorting the array. Note: This approach is particularly useful when the range of elements is small, this is because we are declaring a array of size maximum element. If the range of elements is very large, the counting sort approach may not be the most efficient choice. C++ //Driver Code Starts #include <iostream> #include <vector> using namespace std ; //Driver Code Ends int kthSmallest (vector < int >& arr , int k) { // First, find the maximum element in the vector int maxElement = arr [0]; for (int i = 1 ; i < arr . size (); i ++) { if (arr [i] > maxElement) { maxElement = arr [i]; } } // Create an array to store the frequency of each element vector < int > freq (maxElement + 1 , 0); for (int i = 0 ; i < arr . size (); i ++) { freq [arr [i]] ++ ; } // Keep track of the cumulative frequency of elements int count = 0 ; for (int i = 0 ; i <= maxElement ; i ++) { if (freq [i] != 0) { count += freq [i]; if (count >= k) { // If we have seen k or more elements, // return the current element return i ; } } } return -1 ; } //Driver Code Starts int main () { vector < int > arr = { 10 , 5 , 4 , 3 , 48 , 6 , 2 , 33 , 53 , 10 }; int k = 4 ; cout << kthSmallest (arr , k); return 0 ; } //Driver Code Ends Java //Driver Code Starts class GFG { //Driver Code Ends static int kthSmallest (int [] arr , int k) { // First, find the maximum element in the array int maxElement = arr [0] ; for (int i = 1 ; i < arr . length ; i ++) { if (arr [i] > maxElement) {

maxElement = arr [i] ; } } // Create an array to store the frequency of each element int [] freq = new int [maxElement + 1] ; for (int i = 0 ; i < arr . length ; i ++) { freq [arr [i]]++ ; } // Keep track of the cumulative frequency of elements int count = 0 ; for (int i = 0 ; i <= maxElement ; i ++) { if (freq [i] != 0) { count += freq [i] ; if (count >= k) { // If we have seen k or more elements, // return the current element return i ; } } } return - 1 ; } //Driver Code Starts public static void main (String [] args) { int [] arr = { 10 , 5 , 4 , 3 , 48 , 6 , 2 , 33 , 53 , 10 }; int k = 4 ; System . out . println (kthSmallest (arr , k)); } } //Driver Code Ends Python def kthSmallest (arr , k): # First, find the maximum element in the list maxElement = arr [0] for i in range (1 , len (arr)): if arr [i] > maxElement : maxElement = arr [i] # Create a frequency array for each element freq = [0] * (maxElement + 1) for i in range (len (arr)): freq [arr [i]] += 1 # Keep track of cumulative frequency to find k-th smallest count = 0 for i in range (maxElement + 1): if freq [i] != 0 : count += freq [i] if count >= k : # If we have seen k or more elements, # return the current element return i return - 1 if __name__ == "__main__" : #Driver Code Starts arr = [10 , 5 , 4 , 3 , 48 , 6 , 2 , 33 , 53 , 10] k = 4 print (kthSmallest (arr , k)) #Driver Code Ends C# //Driver Code Starts using System ; class GFG { //Driver Code Ends static int kthSmallest (int [] arr , int k) { // First, find the maximum element in the array int maxElement = arr [0]; for (int i = 1 ; i < arr . Length ; i ++) { if (arr [i] > maxElement) maxElement = arr [i]; } // Create a frequency array for each element int [] freq = new int [maxElement + 1]; for (int i = 0 ; i < arr . Length ; i ++) freq [arr [i]] ++ ; // Keep track of cumulative frequency to find k-th smallest int count = 0 ; for (int i = 0 ; i <= maxElement ; i ++) { if (freq [i] != 0) { count += freq [i]; if (count >= k) { // If we have seen k or more elements, // return the current element return i ; } } } return - 1 ; } //Driver Code Starts static void Main () { int [] arr = { 10 , 5 , 4 , 3 , 48 , 6 , 2 , 33 , 53 , 10 }; int k = 4 ; Console . WriteLine (kthSmallest (arr , k)); } } //Driver Code Ends JavaScript function kthSmallest (arr , k) { // First, find the maximum element in the array let maxElement = arr [0]; for (let i = 1 ; i < arr . length ; i ++) { if (arr [i] > maxElement) maxElement = arr [i]; } // Create a frequency array let freq = new Array (maxElement + 1). fill (0); for (let i = 0 ; i < arr . length ; i ++) { freq [arr [i]] ++ ; } // Track cumulative frequency to find k-th smallest let count = 0 ; for (let i = 0 ; i <= maxElement ; i ++) { if (freq [i] !== 0) { count += freq [i]; if (count >= k) { // If we have seen k or more elements, // return the current element return i ; } } } return - 1 ; } // Driver Code //Driver Code Starts let arr = [10 , 5 , 4 , 3 , 48 , 6 , 2 , 33 , 53 , 10]; let k = 4 ; console . log (kthSmallest (arr , k)); //Driver Code Ends Output 5 Time Complexity: O(n + maxElement), where maxElement is the maximum element of the array. Auxiliary Space: O(maxElement) Related Articles: Print k largest elements of an array Comment Article Tags: Article Tags: Searching Heap DSA Arrays Order-Statistics Microsoft Amazon Cisco VMWare Snapdeal Accolite SAP Labs Rockstand ABCO + 10 More