K-th Largest Sum Contiguous Subarray - GeeksforGeeks

Source: https://www.geeksforgeeks.org/k-th-largest-sum-contiguous-subarray/

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 Largest Sum Contiguous Subarray Last Updated : 8 May, 2025 Given an array arr[] of size n , the task is to find the k th largest sum of contiguous subarray within the array of numbers that has both negative and positive numbers. Examples: Input: arr[] = [20, -5, -1], k = 3 Output: 14 Explanation: All sum of contiguous subarrays are (20, 15, 14, -5, -6, -1), so the 3rd largest sum is 14. Input: arr[] = [10, -10, 20, -40], k = 6 Output: -10 Explanation: The 6th largest sum among sum of all contiguous subarrays is -10. Try it on GfG Practice Table of Content [Naive Approach - 1] - Using Sorting - O(n2 * log n) Time and O(n ^ 2) Space [Naive Approach - 2] - Using Prefix Sum and Sorting - O(n2 * log n) Time and O(n ^ 2) Space [Expected Approach] - Using Min Heap - O(n2 * log k) Time and O(k) Space [Naive Approach - 1] Using Sorting - O(n 2 * log n) Time and O(n ^ 2) Space The idea is to calculate the K th largest sum of contiguous subarrays by generating all possible contiguous subarray sums, storing them, and then sorting them in decreasing order to directly access the K th largest. However, this approach can lead to memory issues when the input array size is large, as the number of contiguous subarrays grows quadratically with the size of the input array. Step-by-Step Implementation: Traverse the input array using two nested loops to generate all contiguous subarrays. For each subarray, calculate the sum and store it in a list. After generating all subarray sums, sort the list in descending order. Return the element at the (k - 1) th index from the sorted list as the Kth largest sum. C++ #include <iostream> #include <vector> #include <algorithm> using namespace std ; // Function to calculate Kth largest element // in contiguous subarray sum int kthLargest (vector < int > & arr , int k) { int n = arr . size (); // to store all subarray sums vector < int > sums ; // Generate all subarrays for (int i = 0 ; i < n ; i ++) { int sum = 0 ; for (int j = i ; j < n ; j ++) { sum += arr [j]; sums . push_back (sum); } } // Sort in decreasing order sort (sums . begin (), sums . end (), greater < int > ()); // return the Kth largest sum return sums [k - 1]; } int main () { vector < int > arr = { 20 , -5 , -1 }; \ int k = 3 ; cout << kthLargest (arr , k); return 0 ; } Java import java.util.* ; class GfG { // Function to calculate Kth largest element // in contiguous subarray sum public static int kthLargest (int [] arr , int k) { int n = arr . length ; // to store all subarray sums ArrayList < Integer > sums = new ArrayList <> (); // Generate all subarrays for (int i = 0 ; i < n ; i ++) { int sum = 0 ; for (int j = i ; j < n ; j ++) { sum += arr [j] ; sums . add (sum); } } // Sort in decreasing order Collections . sort (sums , Collections . reverseOrder ()); // return the Kth largest sum return sums . get (k - 1); } public static void main (String [] args) { int [] arr = { 20 , - 5 , - 1 }; int k = 3 ; System . out . println (kthLargest (arr , k)); } } Python # Function to calculate Kth largest element # in contiguous subarray sum def kthLargest (arr , k): n = len (arr) # to store all subarray sums sums = [] # Generate all subarrays for i in range (n): sum = 0 for j in range (i , n): sum += arr [j] sums . append (sum) # Sort in decreasing order sums . sort (reverse = True) # return the Kth largest sum return sums [k - 1] if __name__ == "__main__" : arr = [20 , - 5 , - 1] k = 3 print (kthLargest (arr , k)) C# using System ; using System.Collections.Generic ; class GfG { // Function to calculate Kth largest element // in contiguous subarray sum public static int kthLargest (int [] arr , int k) { int n = arr . Length ; // to store all subarray sums List < int > sums = new List < int > (); // Generate all subarrays for (int i = 0 ; i < n ; i ++) { int sum = 0 ; for (int j = i ; j < n ; j ++) { sum += arr [j]; sums . Add (sum); } } // Sort in decreasing order sums . Sort ((a , b) => b . CompareTo (a)); // return the Kth largest sum return sums [k - 1]; } public static void Main () { int [] arr = { 20 , - 5 , - 1 }; int k = 3 ; Console . WriteLine (kthLargest (arr , k

)); } } JavaScript // Function to calculate Kth largest element // in contiguous subarray sum function kthLargest (arr , k) { let n = arr . length ; // to store all subarray sums let sums = []; // Generate all subarrays for (let i = 0 ; i < n ; i ++) { let sum = 0 ; for (let j = i ; j < n ; j ++) { sum += arr [j]; sums . push (sum); } } // Sort in decreasing order sums . sort ((a , b) => b - a); // return the Kth largest sum return sums [k - 1]; } let arr = [20 , - 5 , - 1]; let k = 3 ; console . log (kthLargest (arr , k)); Output 14 [Naive Approach - 2] Using Prefix Sum and Sorting - O(n 2 * log n) Time and O(n ^ 2) Space The idea is to leverage a prefix-sum array to compute every contiguous subarray sum in constant time per pair of endpoints, collect all those sums into a single list, sort that list in descending order, and then directly pick out the Kth largest value. Step-by-Step Implementation: Build a prefix-sum array prefixSum of length n+1 where prefixSum [i] equals the total of the first i elements. Initialize an empty list to hold all subarray sums. For each start index i from 1 to n , and for each end index j from i to n , compute the subarray sum as prefixSum [j] – prefixSum[i–1] and append it to the list. Sort the list of subarray sums in decreasing order. Return the element at index k–1 in the sorted list as the K th largest sum. C++ #include <iostream> #include <vector> #include <algorithm> using namespace std ; // The main function to find the K-th largest sum of // contiguous subarray using Prefix Sum and Sorting // approach. int kthLargest (vector < int >& arr , int k) { int n = arr . size (); // Create a prefix sum array. vector < int > prefixSum (n + 1); prefixSum [0] = 0 ; for (int i = 1 ; i <= n ; i ++) { prefixSum [i] = prefixSum [i - 1] + arr [i - 1]; } // Create a vector to store all possible subarray sums. vector < int > subarraySums ; for (int i = 0 ; i <= n ; i ++) { for (int j = i + 1 ; j <= n ; j ++) { subarraySums . push_back (prefixSum [j] - prefixSum [i]); } } // Sort the subarray sums in decreasing order. sort (subarraySums . begin (), subarraySums . end (), greater < int > ()); // Return the K-th largest sum of contiguous subarray. return subarraySums [k - 1]; } int main () { vector < int > arr = { 20 , -5 , -1 }; int k = 3 ; cout << kthLargest (arr , k); return 0 ; } Java import java.util.ArrayList ; import java.util.Collections ; class GfG { // The main function to find the K-th largest sum of // contiguous subarray using Prefix Sum and Sorting approach. static int kthLargest (int [] arr , int k) { int n = arr . length ; // Create a prefix sum array. int [] prefixSum = new int [n + 1] ; prefixSum [0] = 0 ; for (int i = 1 ; i <= n ; i ++) { prefixSum [i] = prefixSum [i - 1] + arr [i - 1] ; } // Create a list to store all possible subarray sums. ArrayList < Integer > subarraySums = new ArrayList <> (); for (int i = 0 ; i <= n ; i ++) { for (int j = i + 1 ; j <= n ; j ++) { subarraySums . add (prefixSum [j] - prefixSum [i]); } } // Sort the subarray sums in decreasing order. Collections . sort (subarraySums , Collections . reverseOrder ()); // Return the K-th largest sum of contiguous subarray. return subarraySums . get (k - 1); } // Driver Code public static void main (String [] args) { int [] arr = { 20 , - 5 , - 1 }; int k = 3 ; System . out . println (kthLargest (arr , k)); } } Python # Python code to implement Prefix sum approach import heapq # The main function to find the K-th largest sum of # contiguous subarray using Prefix Sum and Sorting # approach. def kthLargest (arr , k): n = len (arr) # Create a prefix sum array. prefixSum = [0] * (n + 1) for i in range (1 , n + 1): prefixSum [i] = prefixSum [i - 1] + arr [i - 1] # Create a heap to store K largest subarray sums. subarraySums = [] heapq . heapify (subarraySums) for i in range (n + 1): for j in range (i + 1 , n + 1): subarraySum = prefixSum [j] - prefixSum [i] if len (subarraySums) < k : heapq . heappush (subarraySums , subarraySum) else : if subarraySum > subarraySums [0]: heapq . heapreplace (subarraySums , subarraySum) # Return the K-th largest sum of contiguous subarray. return subarraySums [0] # Driver Code if __name__ == '__main__' : arr = [20 , - 5 , - 1] k = 3 print (kthLargest (arr , k)) C# using System ; using System.Collections.Generic ; using System.Linq ; class GfG { // The main function to find the K-th largest sum of // contiguous subarray using Prefix Sum and Sorting // approach. static int kthLargest (List < int > arr , int k){ int n = arr . Count ; // Create a prefix sum array. List < int > prefixSum = new List < int > (n + 1); prefixSum . Add (0); for (int i = 1 ; i <= n ; i ++){ prefixSum . Add (prefixSum [i - 1] + arr [i - 1]); } // Create a list to store all possible subarray sums. List < int > subarraySums = new List < int > (); for (int i = 0 ; i <= n ; i ++){ for (int j = i + 1 ; j <= n ; j ++){ subarraySums . Add (prefixSum [j] - prefixSum [i]); } } // Sort the subarray sums in decreasing order. subarraySums . Sort (); subarraySums . Reverse (); // Return the K-th largest sum of contiguous subarray. return subarraySums [k - 1]; } // Driver Code to test above function public static void Main (string [] args){ List < int > arr = new List < int > { 20 , - 5 , - 1 }; int k = 3 ; Console . WriteLine (kthLargest (arr , k)); // expected output is -10 } } JavaScript // JavaScript code to implement Prefix sum approach // The main function to find the K-th largest sum of // contiguous subarray using Prefix Sum and Sorting // approach. function kthLargest (arr , k) { let n = arr . length ; // Create a prefix sum array. let prefixSum = new Array (n + 1). fill (0); prefixSum [0] = 0 ; for (let i = 1 ; i <= n ; i ++) { prefixSum [i] = prefixSum [i - 1] + arr [i - 1]; } // Create an array to store all possible subarray sums. let subarraySums = []; for (let i = 0 ; i <= n ; i ++) { for (let j = i + 1 ; j <= n ; j ++) { subarraySums . push (

prefixSum [j] - prefixSum [i]); } } // Sort the subarray sums in decreasing order. subarraySums . sort ((a , b) => b - a); // Return the K-th largest sum of contiguous subarray. return subarraySums [k - 1]; } // Driver Code let arr = [20 , - 5 , - 1]; let k = 3 ; console . log (kthLargest (arr , k)); Output 14 [Expected Approach] - Using Min Heap - O(n 2 * log k) Time and O(k) Space The key idea is to store the pre-sum of the array in a sum[] array. One can find the sum of contiguous subarray from index i to j as sum[j] - sum[i-1]. After this step, this problem becomes same as k-th smallest element in an array . So we generate all possible contiguous subarray sums and push them into the Min-Heap only if the size of Min-Heap is less than K or the current sum is greater than the root of the Min-Heap. In the end, the root of the Min-Heap is the required answer. Follow the given steps to solve the problem using the above approach: Create a prefix sum array of the input array Create a Min-Heap that stores the subarray sum Iterate over the given array using the variable i such that 1 <= i <= N, here i denotes the starting point of the subarray Create a nested loop inside this loop using a variable j such that i <= j <= N, here j denotes the ending point of the subarray Calculate the sum of the current subarray represented by i and j, using the prefix sum array If the size of the Min-Heap is less than K, then push this sum into the heap Otherwise, if the current sum is greater than the root of the Min-Heap, then pop out the root and push the current sum into the Min-Heap Now the root of the Min-Heap denotes the Kth largest sum, Return it C++ #include <iostream> #include <vector> #include <queue> using namespace std ; // Function to calculate Kth largest element // in contiguous subarray sum int kthLargest (vector < int > & arr , int k) { int n = arr . size (); // array to store prefix sums vector < int > sum (n + 1); sum [0] = 0 ; sum [1] = arr [0]; for (int i = 2 ; i <= n ; i ++) sum [i] = sum [i - 1] + arr [i - 1]; // min heap priority_queue < int , vector < int > , greater < int >> pq ; // loop to calculate the contiguous subarray // sums position-wise for (int i = 1 ; i <= n ; i ++) { // loop to traverse all positions that // form contiguous subarray for (int j = i ; j <= n ; j ++) { // calculates the contiguous subarray // sums from j to i index int x = sum [j] - sum [i - 1]; // if queue has less than k elements, // then simply push it if (pq . size () < k) pq . push (x); else { // it the min heap has equal to // k elements then just check // if the largest kth element is // smaller than x then insert // else its of no use if (pq . top () < x) { pq . pop (); pq . push (x); } } } } // the top element will be then kth // largest element return pq . top (); } int main () { vector < int > arr = { 20 , -5 , -1 }; int k = 3 ; cout << kthLargest (arr , k); return 0 ; } Java import java.util.* ; class GfG { // Function to calculate Kth largest element // in contiguous subarray sum static int kthLargest (int [] arr , int k) { int n = arr . length ; // array to store prefix sums int [] sum = new int [n + 1] ; sum [0] = 0 ; sum [1] = arr [0] ; for (int i = 2 ; i <= n ; i ++) sum [i] = sum [i - 1] + arr [i - 1] ; // min heap PriorityQueue < Integer > pq = new PriorityQueue <> (); // loop to calculate the contiguous subarray // sums position-wise for (int i = 1 ; i <= n ; i ++) { // loop to traverse all positions that // form contiguous subarray for (int j = i ; j <= n ; j ++) { // calculates the contiguous subarray // sums from j to i index int x = sum [j] - sum [i - 1] ; // if queue has less than k elements, // then simply push it if (pq . size () < k) pq . add (x); else { // it the min heap has equal to // k elements then just check // if the largest kth element is // smaller than x then insert // else its of no use if (pq . peek () < x) { pq . poll (); pq . add (x); } } } } // the top element will be then kth // largest element return pq . peek (); } public static void main (String [] args) { int [] arr = { 20 , - 5 , - 1 }; int k = 3 ; System . out . println (kthLargest (arr , k)); } } Python import heapq # Function to calculate Kth largest element # in contiguous subarray sum def kthLargest (arr , k): n = len (arr) # array to store prefix sums sum = [0] * (n + 1) sum [0] = 0 sum [1] = arr [0] for i in range (2 , n + 1): sum [i] = sum [i - 1] + arr [i - 1] # min heap pq = [] # loop to calculate the contiguous subarray # sums position-wise for i in range (1 , n + 1): # loop to traverse all positions that # form contiguous subarray for j in range (i , n + 1): # calculates the contiguous subarray # sums from j to i index x = sum [j] - sum [i - 1] # if queue has less than k elements, # then simply push it if len (pq) < k : heapq . heappush (pq , x) else : # it the min heap has equal to # k elements then just check # if the largest kth element is # smaller than x then insert # else its of no use if pq [0] < x : heapq . heapreplace (pq , x) # the top element will be then kth # largest element return pq [0] # Driver's code if __name__ == "__main__" : arr = [20 , - 5 , - 1] k = 3 # Function call print (kthLargest (arr , k)) C# // C# program to find the K-th // largest sum of subarray using System ; using System.Collections.Generic ; class GfG { // function to calculate Kth largest // element in contiguous subarray sum static int kthLargest (int [] arr , int N , int K){ // array to store prefix sums int [] sum = new int [N + 1]; sum [0] = 0 ; sum [1] = arr [0]; for (int i = 2 ; i <= N ; i ++) sum [i] = sum [i - 1] + arr [i - 1]; // priority_queue of min heap List < int > Q = new List < int > (); // loop to calculate the contiguous subarray // sum position-wise for (int i = 1 ; i <= N ; i ++) { // loop to traverse all positions that // form contiguous subarray for (int j = i ; j <= N ; j ++) { // calculates the contiguous subarray // sum from j to i index int x = sum [j] - sum [i - 1]; // if queue has less than k elements, // then simply push it if (Q .

Count < K) Q . Add (x); else { // it the min heap has equal to // k elements then just check // if the largest kth element is // smaller than x then insert // else its of no use Q . Sort (); if (Q [0] < x) { Q . RemoveAt (0); Q . Add (x); } } Q . Sort (); } } // the top element will be then Kth // largest element return Q [0]; } // Driver's Code public static void Main (String [] args) { int [] a = new int [] { 20 , - 5 , - 1 }; int N = a . Length ; int K = 3 ; // Function call Console . WriteLine (kthLargest (a , N , K)); } } // This code contributed by Rajput-Ji JavaScript // Custom MinHeap class to act as a priority queue class MinHeap { constructor () { this . heap = []; } // Return the top (minimum) element peek () { return this . heap [0]; } // Insert a new value into the heap insert (val) { this . heap . push (val); this . _heapifyUp (this . heap . length - 1); } // Remove and return the top (minimum) element extractMin () { if (this . heap . length === 0) return null ; if (this . heap . length === 1) return this . heap . pop (); const min = this . heap [0]; this . heap [0] = this . heap . pop (); this . _heapifyDown (0); return min ; } size () { return this . heap . length ; } // Helper to maintain heap property after insertion _heapifyUp (index) { while (index > 0) { const parent = Math . floor ((index - 1) / 2); if (this . heap [parent] > this . heap [index]) { [this . heap [parent], this . heap [index]] = [this . heap [index], this . heap [parent]]; index = parent ; } else { break ; } } } // Helper to maintain heap property after removal _heapifyDown (index) { const n = this . heap . length ; while (true) { let smallest = index ; const left = 2 * index + 1 ; const right = 2 * index + 2 ; if (left < n && this . heap [left] < this . heap [smallest]) { smallest = left ; } if (right < n && this . heap [right] < this . heap [smallest]) { smallest = right ; } if (smallest !== index) { [this . heap [index], this . heap [smallest]] = [this . heap [smallest], this . heap [index]]; index = smallest ; } else { break ; } } } } // Function to find the Kth largest subarray sum function kthLargest (arr , k) { const n = arr . length ; // Create prefix sum array const prefix = new Array (n + 1). fill (0); for (let i = 1 ; i <= n ; i ++) { prefix [i] = prefix [i - 1] + arr [i - 1]; } // Use MinHeap to keep track of top K largest sums const pq = new MinHeap (); for (let i = 1 ; i <= n ; i ++) { for (let j = i ; j <= n ; j ++) { const sum = prefix [j] - prefix [i - 1]; if (pq . size () < k) { pq . insert (sum); } else if (pq . peek () < sum) { pq . extractMin (); pq . insert (sum); } } } return pq . peek (); } // Driver Code let arr = [20 , - 5 , - 1]; let k = 3 ; console . log (kthLargest (arr , k)); // Output: 14 Output 14 Comment Article Tags: Article Tags: Misc Heap DSA Arrays Order-Statistics subarray subarray-sum + 3 More