Connect n ropes with minimum cost - GeeksforGeeks

Source: https://www.geeksforgeeks.org/connect-n-ropes-minimum-cost/

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 Connect n ropes with minimum cost Last Updated : 18 Oct, 2025 Given an array arr[] of rope lengths, connect all ropes into a single rope with the minimum total cost. The cost to connect two ropes is the sum of their lengths. Examples: Input: arr[] = [4, 3, 2, 6] Output: 29 Explanation: Minimum cost to connect all ropes into a single rope is Connect ropes 2 and 3 - [4, 5, 6], cost = 5 Connect ropes 4 and 5 - [9, 6], cost = 9 Connect ropes 9 and 6 - [15], cost = 15 Total cost = 5 + 9 + 15 = 29 Input: arr[] = [10] Output: 0 Try it on GfG Practice Table of Content [Naive Approach] Greedy with Repeated Sorting - O(n^2*log(n)) Time and O(n) Space [Expected Approach] Greedy with Heap - O(n*log(n)) Time and O(n) Space [Naive Approach] Greedy with Repeated Sorting - O(n 2 *log(n)) Time and O(n) Space The idea is to always connect the two shortest ropes first, minimizing the cost at each step using a greedy approach. We repeatedly sort the array, pick the two smallest ropes, connect them, and add their combined length back into the array while accumulating the cost. This process continues until only one rope remains. C++ #include <iostream> #include <vector> #include <algorithm> using namespace std ; int minCost (vector < int >& arr) { int totalCost = 0 ; // Continue until only one rope remains while (arr . size () > 1) { // Sort ropes to get the two smallest lengths sort (arr . begin (), arr . end ()); // Pick the two smallest ropes int first = arr [0]; int second = arr [1]; // Remove the two ropes from the array arr . erase (arr . begin ()); arr . erase (arr . begin ()); // Cost of connecting these two ropes int cost = first + second ; totalCost += cost ; // Push the new rope back into the array arr . push_back (cost); } return totalCost ; } int main () { vector < int > ropes = { 4 , 3 , 2 , 6 }; cout << minCost (ropes) << endl ; return 0 ; } Java import java.util.Arrays ; class GFG { static int minCost (int [] arr) { int totalCost = 0 ; int n = arr . length ; // Continue until only one rope remains while (n > 1) { // Sort ropes to get the two smallest lengths Arrays . sort (arr , 0 , n); // Pick the two smallest ropes int first = arr [0] ; int second = arr [1] ; // Cost of connecting these two ropes int cost = first + second ; totalCost += cost ; // Shift the array to remove the two smallest elements for (int i = 2 ; i < n ; i ++) { arr [i - 2] = arr [i] ; } // Place the new rope at the end arr [n - 2] = cost ; // Reduce array size by 1 (two removed, one added) n -- ; } return totalCost ; } public static void main (String [] args) { int [] ropes = { 4 , 3 , 2 , 6 }; System . out . println (minCost (ropes)); } } Python def minCost (arr): totalCost = 0 # Continue until only one rope remains while len (arr) > 1 : # Sort ropes to get the two smallest lengths arr . sort () # Pick the two smallest ropes first = arr [0] second = arr [1] # Remove the two ropes from the array arr . pop (0) arr . pop (0) # Cost of connecting these two ropes cost = first + second totalCost += cost # Push the new rope back into the array arr . append (cost) return totalCost if __name__ == "__main__" : ropes = [4 , 3 , 2 , 6] print (minCost (ropes)) C# using System ; class GFG { static int minCost (int [] arr) { int totalCost = 0 ; int n = arr . Length ; // Continue until only one rope remains while (n > 1) { // Sort ropes to get the two smallest lengths Array . Sort (arr , 0 , n); // Pick the two smallest ropes int first = arr [0]; int second = arr [1]; // Cost of connecting these two ropes int cost = first + second ; totalCost += cost ; // Shift the array to remove the two smallest elements for (int i = 2 ; i < n ; i ++) { arr [i - 2] = arr [i]; } // Place the new rope at the end arr [n - 2] = cost ; // Reduce array size by 1 (two removed, one added) n -- ; } return totalCost ; } static void Main () { int [] ropes = { 4 , 3 , 2 , 6 }; Console . WriteLine (minCost (ropes)); } } JavaScript function minCost (arr) { let totalCost = 0 ; // Continue until only one rope remains while (arr . length > 1) { // Sort ropes to get the two smallest lengths arr . sort ((a , b) => a - b

); // Pick the two smallest ropes let first = arr [0]; let second = arr [1]; // Remove the two ropes from the array arr . splice (0 , 2); // Cost of connecting these two ropes let cost = first + second ; totalCost += cost ; // Push the new rope back into the array arr . push (cost); } return totalCost ; } function main () { let ropes = [4 , 3 , 2 , 6]; console . log (minCost (ropes)); } main (); Output 29 [Expected Approach] Greedy with Heap - O(n*log(n)) Time and O(n) Space Always connect the two smallest ropes first to minimize total cost, similar to Huffman coding.The idea is to use a min-heap (priority queue) and push all elements into it. While the heap contains more than one element, repeatedly extract the two smallest values, add them, and insert the sum back into the heap. Continue until one rope remains, then return the total cost. C++ //Driver Code Starts #include <iostream> #include <vector> #include <queue> using namespace std ; //Driver Code Ends int minCost (vector < int > & arr) { // Create a min priority queue priority_queue < int , vector < int > , greater < int >> pq (arr . begin (), arr . end ()); // Initialize result int res = 0 ; // While size of priority queue is more than 1 while (pq . size () > 1) { // Extract shortest two ropes from pq int first = pq . top (); pq . pop (); int second = pq . top (); pq . pop (); // Connect the ropes: update result and // insert the new rope to pq res += first + second ; pq . push (first + second); } return res ; } //Driver Code Starts int main () { vector < int > arr = { 4 , 3 , 2 , 6 }; cout << minCost (arr); return 0 ; } //Driver Code Ends Java //Driver Code Starts import java.util.PriorityQueue ; class GFG { //Driver Code Ends static int minCost (int [] arr) { // Create a min priority queue PriorityQueue < Integer > pq = new PriorityQueue <> (); for (int num : arr) { pq . add (num); } // Initialize result int res = 0 ; // While size of priority queue is more than 1 while (pq . size () > 1) { // Extract shortest two ropes from pq int first = pq . poll (); int second = pq . poll (); // Connect the ropes: update result and // insert the new rope to pq res += first + second ; pq . add (first + second); } return res ; } //Driver Code Starts public static void main (String [] args) { int [] arr = { 4 , 3 , 2 , 6 }; System . out . println (minCost (arr)); } } //Driver Code Ends Python #Driver Code Starts import heapq #Driver Code Ends def minCost (arr): # Create a priority queue # By default, heapq provides a min-heap heapq . heapify (arr) # Initialize result res = 0 # While size of priority queue is more than 1 while len (arr) > 1 : # Extract shortest two ropes from pq first = heapq . heappop (arr) second = heapq . heappop (arr) # Connect the ropes: update result and # insert the new rope to pq res += first + second heapq . heappush (arr , first + second) return res #Driver Code Starts if __name__ == "__main__" : arr = [4 , 3 , 2 , 6] print (minCost (arr)) #Driver Code Ends C# //Driver Code Starts using System ; using System.Collections.Generic ; // Custom MinHeap class class MinHeap { private List < int > heap = new List < int > (); private void Swap (int i , int j) { int temp = heap [i]; heap [i] = heap [j]; heap [j] = temp ; } public void Add (int val) { heap . Add (val); int i = heap . Count - 1 ; while (i > 0) { int parent = (i - 1) / 2 ; if (heap [i] >= heap [parent]) break ; Swap (i , parent); i = parent ; } } public int Pop () { if (heap . Count == 0) throw new InvalidOperationException ("Heap is empty"); int root = heap [0]; heap [0] = heap [heap . Count - 1]; heap . RemoveAt (heap . Count - 1); int i = 0 ; while (true) { int left = 2 * i + 1 ; int right = 2 * i + 2 ; int smallest = i ; if (left < heap . Count && heap [left] < heap [smallest]) smallest = left ; if (right < heap . Count && heap [right] < heap [smallest]) smallest = right ; if (smallest == i) break ; Swap (i , smallest); i = smallest ; } return root ; } public int Count () { return heap . Count ; } } class GFG { //Driver Code Ends static int minCost (int [] arr) { // Create a min priority queue MinHeap pq = new MinHeap (); foreach (int val in arr) pq . Add (val); // Initialize result int res = 0 ; // While size of priority queue is more than 1 while (pq . Count () > 1) { // Extract shortest two ropes from pq int first = pq . Pop (); int second = pq . Pop (); // Connect the ropes: update result //and insert the new rope to pq res += first + second ; pq . Add (first + second); } return res ; } //Driver Code Starts static void Main () { int [] arr = { 4 , 3 , 2 , 6 }; Console . WriteLine (minCost (arr)); } } //Driver Code Ends JavaScript //Driver Code Starts class MinHeap { constructor () { this . heap = []; } // Insert a value into the heap push (val) { this . heap . push (val); this . heapifyUp (); } // Remove and return the smallest element pop () { if (this . heap . length === 1) { return this . heap . pop (); } const root = this . heap [0]; this . heap [0] = this . heap . pop (); this . heapifyDown (); return root ; } // Return the smallest element without removing it top () { return this . heap [0]; } // Return the size of the heap size () { return this . heap . length ; } // Restore the heap property upwards heapifyUp () { let index = this . heap . length - 1 ; while (index > 0) { const parentIndex = Math . floor ((index - 1) / 2); if (this . heap [index] >= this . heap [parentIndex]) { break ; } [this . heap [index], this . heap [parentIndex]] = [this . heap [parentIndex], this . heap [index]]; index = parentIndex ; } } // Restore the heap property downwards heapifyDown () { let index = 0 ; const size = this . heap . length ; while (true) { const leftChild = 2 * index + 1 ; const rightChild = 2 * index + 2 ; let smallest = index ; if (leftChild < size && this . heap [leftChild] < this . heap [smallest]) { smallest = leftChild ; } if (rightChild < size && this . heap [rightChild] < this . heap [smallest]) { smallest = rightChild ; } if (smallest === index) {

break ; } [this . heap [index], this . heap [smallest]] = [this . heap [smallest], this . heap [index]]; index = smallest ; } } } //Driver Code Ends function minCost (arr) { // Create a min-heap const pq = new MinHeap (); arr . forEach (num => pq . push (num)); // Initialize result let res = 0 ; // While size of priority queue is more than 1 while (pq . size () > 1) { // Extract shortest two ropes from pq const first = pq . pop (); const second = pq . pop (); // Connect the ropes: update result and // insert the new rope to pq res += first + second ; pq . push (first + second); } return res ; } //Driver Code Starts // Driver Code const arr = [4 , 3 , 2 , 6]; console . log (minCost (arr)); //Driver Code Ends Output 29 Comment Article Tags: Article Tags: Queue Greedy Heap DSA Amazon Goldman Sachs OYO + 3 More