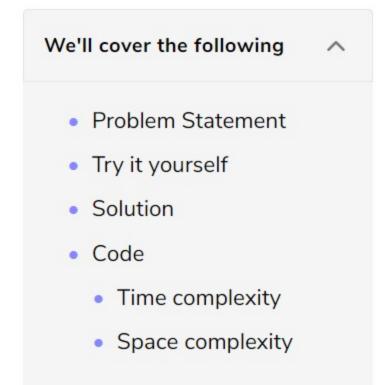


Q Search Course

educative

Connect Ropes (easy)



Problem Statement

Given 'N' ropes with different lengths, we need to connect these ropes into one big rope with minimum cost. The cost of connecting two ropes is equal to the sum of their lengths.

₿

Example 1:

```
Input: [1, 3, 11, 5]
Output: 33
Explanation: First connect 1+3(=4), then 4+5(=9), and then 9+11(=20). So the total cost is 33 (4+9+20)
```

Example 2:

```
Input: [3, 4, 5, 6]
Output: 36
Explanation: First connect 3+4(=7), then 5+6(=11), 7+11(=18). Total cost is 36 (7+11+18)
```

Example 3:

```
Input: [1, 3, 11, 5, 2]
Output: 42
Explanation: First connect 1+2(=3), then 3+3(=6), 6+5(=11), 11+11(=22). Total cost is 42 (3+6+11+22)
```

Try it yourself

Try solving this question here:

```
@ C++
            Python3
                          JS JS
Java
 1 def minimum_cost_to_connect_ropes(ropeLengths):
      result = []
      # TODO: Write your code here
      return result
    def main():
      print("Minimum cost to connect ropes: " +
              str(minimum_cost_to_connect_ropes([1, 3, 11, 5])))
      print("Minimum cost to connect ropes: " +
            str(minimum_cost_to_connect_ropes([3, 4, 5, 6])))
12
      print("Minimum cost to connect ropes: " +
13
            str(minimum_cost_to_connect_ropes([1, 3, 11, 5, 2])))
14
15
17 main()
19
 Run
                                                                                              Save
                                                                                                        Reset
```

Solution

In this problem, following a greedy approach to connect the smallest ropes first will ensure the lowest cost. We can use a **Min Heap** to find the smallest ropes following a similar approach as discussed in **Kth Smallest Number**. Once we connect two ropes, we need to insert the resultant rope back in the **Min Heap** so that we can connect it with the remaining ropes.

Code

Here is what our algorithm will look like:

```
Python3
                          G C++
                                       JS JS
🔮 Java
    from heapq import *
    def minimum_cost_to_connect_ropes(ropeLengths):
      minHeap = []
      # add all ropes to the min heap
      for i in ropeLengths:
        heappush(minHeap, i)
      # go through the values of the heap, in each step take top (lowest) rope lengths from the min heap
      # connect them and push the result back to the min heap.
      # keep doing this until the heap is left with only one rope
      result, temp = 0, 0
      while len(minHeap) > 1:
14
        temp = heappop(minHeap) + heappop(minHeap)
15
        result += temp
        heappush(minHeap, temp)
17
      return result
21
    def main():
22
      print("Minimum cost to connect ropes: " +
24
            str(minimum_cost_to_connect_ropes([1, 3, 11, 5])))
25
      print("Minimum cost to connect ropes: " +
            str(minimum_cost_to_connect_ropes([3, 4, 5, 6])))
27
      print("Minimum cost to connect ropes: " +
            str(minimum_cost_to_connect_ropes([1, 3, 11, 5, 2])))
30
31 main()
                                                                                                                  :3
                                                                                                         Reset
 Run
                                                                                               Save
```

Time complexity

Given 'N' ropes, we need O(N*logN) to insert all the ropes in the heap. In each step, while processing the heap, we take out two elements from the heap and insert one. This means we will have a total of 'N' steps, having a total time complexity of O(N*logN).

Space complexity

The space complexity will be ${\cal O}(N)$ because we need to store all the ropes in the heap.

