Connect Ropes (easy)

We'll cover the following

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 3
3 (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+1
1+22)
```

Try it yourself

Try solving this question here:

Java | Python3 | JS | C++

```java
import java.util.*;

class ConnectRopes {

  public static int minimumCostToConnectRopes(int[] ropeLengths) {
    // TODO: Write your code here
    return -1;
  }

  public static void main(String[] args) {
    int result = ConnectRopes.minimumCostToConnectRopes(new int[] { 1, 3, 11, 5 });
    System.out.println("Minimum cost to connect ropes: " + result);
    result = ConnectRopes.minimumCostToConnectRopes(new int[] { 3, 4, 5, 6 });
    System.out.println("Minimum cost to connect ropes: " + result);
    result = ConnectRopes.minimumCostToConnectRopes(new int[] { 1, 3, 11, 5, 2 });
    System.out.println("Minimum cost to connect ropes: " + result);
  }
}

```

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:

Java | Python3 | C++ | JS

```java
import java.util.*;

class ConnectRopes {

  public static int minimumCostToConnectRopes(int[] ropeLengths) {
    PriorityQueue<Integer> minHeap = new PriorityQueue<Integer>((n1, n2) -> n1 - n2);
    // add all ropes to the min heap
    for (int i = 0; i < ropeLengths.length; i++)
      minHeap.add(ropeLengths[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
    int result = 0, temp = 0;
    while (minHeap.size() > 1) {
      temp = minHeap.poll() + minHeap.poll();
      result += temp;
      minHeap.add(temp);
    }

    return result;
  }

  public static void main(String[] args) {
    int result = ConnectRopes.minimumCostToConnectRopes(new int[] { 1, 3, 11, 5 });
    System.out.println("Minimum cost to connect ropes: " + result);
    result = ConnectRopes.minimumCostToConnectRopes(new int[] { 3, 4, 5, 6 });
    System.out.println("Minimum cost to connect ropes: " + result);
```

Run | Save | Reset

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 $O(N)$ because we need to store all the ropes in the heap.

Back

'K' Closest Points to the Origin (easy)

Next

Top 'K' Frequent Numbers (medium)

Mark as Completed

Report an Issue | Ask a Question