Rohan is given n ropes of different lengths, he needs to connect these ropes into one rope. The cost to connect two ropes is equal to the sum of their lengths. He needs to connect the ropes with minimum cost. Help him to connect ropes.

Solution:

```
import java.util.Scanner;

class MinHeap {

    int[] harr;

    int heap_size;

    int capacity;

    public MinHeap(int a[], int size)

    {

            heap_size = size;

            capacity = size;

            harr = a;

            int i = (heap_size - 1) / 2;

            while (i >= 0) {

                    MinHeapify(i);

                    i--;

            }

    }


    void MinHeapify(int i)

    {
```

```
        int l = left(i);
        int r = right(i);
        int smallest = i;
        if (l < heap_size && harr[l] < harr[i])
                smallest = l;
        if (r < heap_size && harr[r] < harr[smallest])
                smallest = r;
        if (smallest != i) {
                swap(i, smallest);
                MinHeapify(smallest);
        }
    }
    int parent(int i) {
 return (i - 1) / 2;
}

    int left(int i) {
return (2 * i + 1);
 }
    int right(int i) {
return (2 * i + 2);
}

    int extractMin()
```

```
{
        if (heap_size <= 0)
                return Integer.MAX_VALUE;
        if (heap_size == 1) {
                heap_size--;
                return harr[0];
        }

        int root = harr[0];
        harr[0] = harr[heap_size - 1];
        heap_size--;
        MinHeapify(0);
        return root;
}

void insertKey(int k)
{
        if (heap_size == capacity) {
                System.out.println("Overflow: Could not insertKey");
                return;
        }

        heap_size++;
        int i = heap_size - 1;
```

```
        harr[i] = k;
        while (i != 0 && harr[parent(i)] > harr[i]) {
                swap(i, parent(i));
                i = parent(i);
        }
}

boolean isSizeOne()
{
        return (heap_size == 1);
}

void swap(int x, int y)
{
        int temp = harr[x];
        harr[x] = harr[y];
        harr[y] = temp;
}

static int minCost(int len[], int n)
{
        int cost = 0;
        MinHeap minHeap = new MinHeap(len, n);
```

```
        while (!minHeap.isSizeOne()) {

                int min = minHeap.extractMin();
                int sec_min = minHeap.extractMin();
                cost += (min + sec_min); // Update total cost

                minHeap.insertKey(min + sec_min);
        }

        return cost;
}

public static void main(String args[])
{   Scanner sc=new Scanner(System.in);
    int a=sc.nextInt();

        int len[] =new int[a];
     for(int i=0;i<len.length;i++)
     {
       len[i]=sc.nextInt();
     }
        int size = len.length;
        System.out.println(minCost(len, size));
}
```

};

Input Format:

First line is a positive integer n, denoting the number of ropes. 1<=n<=20
Second line contains a list of rope lengths.

Output Format:

A positive integer, denoting minimum cost to connect ropes.