Connect n ropes with minimum cost

There are given n ropes of different lengths, we need to connect these ropes into one rope. The cost to connect two ropes is equal to sum of their lengths. We need to connect the ropes with minimum cost.

For example if we are given 4 ropes of lengths 4, 3, 2 and 6. We can connect the ropes in following ways.
1) First connect ropes of lengths 2 and 3. Now we have three ropes of lengths 4, 6 and 5.
2) Now connect ropes of lengths 4 and 5. Now we have two ropes of lengths 6 and 9.
3) Finally connect the two ropes and all ropes have connected.

Total cost for connecting all ropes is 5 + 9 + 15 = 29. This is the optimized cost for connecting ropes. Other ways of connecting ropes would always have same or more cost. For example, if we connect 4 and 6 first (we get three strings of 3, 2 and 10), then connect 10 and 3 (we get two strings of 13 and 2). Finally we connect 13 and 2. Total cost in this way is 10 + 13 + 15 = 38.

We strongly recommend that you click here and practice it, before moving on to the solution.

If we observe the above problem closely, we can notice that the lengths of the ropes which are picked first are included more than once in total cost. Therefore, the idea is to connect smallest two ropes first and recur for remaining ropes. This approach is similar to Huffman Coding. We put smallest ropes down the tree so that they can be repeated multiple times rather than the longer ropes.

Following is complete algorithm for finding the minimum cost for connecting n ropes.
Let there be n ropes of lengths stored in an array len[0..n-1]
1) Create a min heap and insert all lengths into the min heap.
2) Do following while number of elements in min heap is not one.
.....a) Extract the minimum and second minimum from min heap
.....b) Add the above two extracted values and insert the added value to the min-heap.
3) Return the value of only left item in min heap.

Following is C++ implementation of above algorithm.

```
// C++ program for connecting n ropes with minimum cost
#include <iostream>
using namespace std;

// A Min Heap:  Collection of min heap nodes
struct MinHeap
{
    unsigned size;    // Current size of min heap
    unsigned capacity;   // capacity of min heap
    int *harr;  // Attay of minheap nodes
};

// A utility function to create a min heap of given capacity
struct MinHeap* createMinHeap(unsigned capacity)
{
    struct MinHeap* minHeap = new MinHeap;
    minHeap->size = 0;  // current size is 0
    minHeap->capacity = capacity;
    minHeap->harr = new int[capacity];
    return minHeap;
}

// A utility function to swap two min heap nodes
void swapMinHeapNode(int* a, int* b)
{
    int temp = *a;
    *a = *b;
```

```
    *b = temp;
}

// The standard minHeapify function.
void minHeapify(struct MinHeap* minHeap, int idx)
{
    int smallest = idx;
    int left = 2 * idx + 1;
    int right = 2 * idx + 2;

    if (left < minHeap->size &&
        minHeap->harr[left] < minHeap->harr[smallest])
      smallest = left;

    if (right < minHeap->size &&
        minHeap->harr[right] < minHeap->harr[smallest])
      smallest = right;

    if (smallest != idx)
    {
        swapMinHeapNode(&minHeap->harr[smallest], &minHeap->harr[idx]);
        minHeapify(minHeap, smallest);
    }
}

// A utility function to check if size of heap is 1 or not
int isSizeOne(struct MinHeap* minHeap)
{
    return (minHeap->size == 1);
}

// A standard function to extract minimum value node from heap
int extractMin(struct MinHeap* minHeap)
{
    int temp = minHeap->harr[0];
    minHeap->harr[0] = minHeap->harr[minHeap->size - 1];
    --minHeap->size;
    minHeapify(minHeap, 0);
    return temp;
}

// A utility function to insert a new node to Min Heap
void insertMinHeap(struct MinHeap* minHeap, int val)
{
    ++minHeap->size;
    int i = minHeap->size - 1;
    while (i && (val < minHeap->harr[(i - 1)/2]))
    {
        minHeap->harr[i] = minHeap->harr[(i - 1)/2];
        i = (i - 1)/2;
    }
    minHeap->harr[i] = val;
}

// A standard funvtion to build min heap
void buildMinHeap(struct MinHeap* minHeap)
{
    int n = minHeap->size - 1;
    int i;
    for (i = (n - 1) / 2; i >= 0; --i)
        minHeapify(minHeap, i);
}

// Creates a min heap of capacity equal to size and inserts all values
// from len[] in it. Initially size of min heap is equal to capacity
struct MinHeap* createAndBuildMinHeap(int len[], int size)
{
    struct MinHeap* minHeap = createMinHeap(size);
    for (int i = 0; i < size; ++i)
        minHeap->harr[i] = len[i];
    minHeap->size = size;
    buildMinHeap(minHeap);
    return minHeap;
```

```
}

// The main function that returns the minimum cost to connect n ropes of
// lengths stored in len[0..n-1]
int minCost(int len[], int n)
{
    int cost = 0;  // Initialize result

    // Create a min heap of capacity equal to n and put all ropes in it
    struct MinHeap* minHeap = createAndBuildMinHeap(len, n);

    // Iterate while size of heap doesn't become 1
    while (!isSizeOne(minHeap))
    {
        // Extract two minimum length ropes from min heap
        int min     = extractMin(minHeap);
        int sec_min = extractMin(minHeap);

        cost += (min + sec_min);  // Update total cost

        // Insert a new rope in min heap with length equal to sum
        // of two extracted minimum lengths
        insertMinHeap(minHeap, min+sec_min);
    }

    // Finally return total minimum cost for connecting all ropes
    return cost;
}

// Driver program to test above functions
int main()
{
    int len[] = {4, 3, 2, 6};
    int size = sizeof(len)/sizeof(len[0]);
    cout << "Total cost for connecting ropes is " << minCost(len, size);
    return 0;
}
```

Output:

```
Total cost for connecting ropes is 29
```

Time Complexity: Time complexity of the algorithm is O(nLogn) assuming that we use a O(nLogn) sorting algorithm. Note that heap operations like insert and extract take O(Logn) time.

Algorithmic Paradigm: Greedy Algorithm

A simple implementation with STL in C++

Following is a simple implementation that uses priority_queue available in STL. Thanks to Pango89 for providing below code.

```
#include<iostream>
#include<queue>
using namespace std;

int minCost(int arr[], int n)
{
    // Create a priority queue ( http://www.cplusplus.com/reference/queue/priority_queue/ )
 // By default 'less' is used which is for decreasing order
 // and 'greater' is used for increasing order
 priority_queue< int, vector<int>, greater<int> > pq(arr, arr+n);

    // 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 program to test above function
int main()
{
    int len[] = {4, 3, 2, 6};
    int size = sizeof(len)/sizeof(len[0]);
    cout << "Total cost for connecting ropes is " << minCost(len, size);
    return 0;
}
```

Output:

```
Total cost for connecting ropes is 29
```