CMPS 102 — Fall 2018 – Homework 2

Alyssa Melton

I have read and agree to the collaboration policy.
Collaborators: none

Solution to Problem 4

Key fact:
The cost of joining two ropes, equals the sum of the lengths of the two ropes. $C = l_i + l_j$.

It is useful to think of the most efficient way to elliminate a rope while incurring the least cost. By adding the smallest ropes, we add the smallest cost that we possibly can, while decreasing the total number of ropes. Consider the following algorithm:

```
sort all ropes by length from smallest to greatest.
initialize the cost to be zero. C = 0.

tieropes(List, Cost)
    if length of List = 1, return Cost.
    else tie together the smallest two ropes, forming a single rope.
    $Cost = Cost + (lengthrope_1 + lengthrope_2)$
    move the new rope of length $(lengthrope_1 + lengthrope_2)$ into it's appropriate spot in the sorted list.
    tieropes(List, Cost)
```

Claim 1. *The above algorithm is optimal.*

Proof. If the above algorithm, call it, *Greedy*, is not optimal, then there must be some other algorithm, call it *O*, that is optimal. Say that *O* is the same as *Greedy* up until tie_{k+1}, and tie_k was the last tie that was the same as the *Greedy*. Then, this tie must have tied two ropes that were not both the smallest ropes in the list, otherwise it would be the same as *greedy*. Call the two smallest ropes r_1 and r_2. Any other ropes, r_i or r_j where $i, j \in \{1, 2, ...n\}$, is longer than r_1 and r_2. Then it must be that $r_i + r_j > r_1 + r_2$. Because the cost of adding r_i and r_j is greater than or equal to $r_1 + r_2$, *Greedy* takes no harm in adding r_1 and r_2 instead. Thus, the *Cost* of *Greedy* must be at most the *Cost* of *O*, thus *Greedy* must be, if not also, optimal. □

Claim 2. *The above algorithm is $O(nlog\, n)$.*

Proof. The algorithm takes $O(nlog\, n)$ to sort the list (or build the min-heap). It then extracts the top two smallest ropes in $O(log\, n)$ time, and does this n times, so we get $2 * n * logn$. It then inserts the new rope back in in O(logn) time and does this n times. We ultimately get $nlog\, n + 2nlog\, n + nlog\, n$ which evaluates to $O(nlog\, n)$ □