Assignment 2

Erik Norlin, CID: norliner

September 10, 2023

Problem 4

a)

An algorithm for this problem that runs in $O(n)$ would be an algorithm that touches each element once. Since the rooms are sorted in a descending manner we can utilize this. We start by computing the largest and smallest rooms added together, i.e., $s_i + s_j$ where $i = 0$ and $j = n$. If this is equal to s we are done. if it is too large we pick one room smaller than s_i (s_{i+1}) and keep the smallest room, add them together and check again. Instead if it is too small we pick one room larger than s_j (s_{j-1}) and keep the largest room, add them together and check again. We repeat this until we found $s_i + s_j = s$ or if $i > j$. When $i > j$ we start doing the same comparisons again which is unnecessary. When we have checked $s_i + s_j$ where $i = j$ we have done n operations and thus the algorithm runs in $O(n)$.

Pseudo code for this algorithms

```
r_1 = 0 # index of the first room to return
r_2 = 0 # index of the second room to return
i = 1
j = n
bool = false

while(!bool)
    if s_i + s_j > s
        i += 1
    else if s_i + s_j < s
        j -= 1
    else if s_i + s_j = s
        r_1 = i
        r_2 = j
        bool = true
    if i > j
        bool = true

return r_1, r_2
```

If no room size equal to s are found r_1 and r_2 return as zeros, otherwise they return indices of rooms that added together equal to the size of s.

b)

The nature of this algorithms ensures that no solutions are missed by mistake. this algorithm relies on that the rooms $s_1, ..., s_n$ are ordered in a descending manner by size. If we start with $s_1 + s_n$ and this is larger than s it only make sense to check $s_2 + s_n$. If instead proceeding with checking $s_1 + s_{n-1}$ this would result in an even larger size which we do not want. Checking $s_i + s_j$ for every increment and decrement of i and j respectively ensures no mistake.