Jordan Allard
CSCI 264-01
Homework 1

HW1 Problem 5 Writeup

a) Verbal Description

Given a necklace of containing beads of different weights, this algorithm determines if it is possible to divide the necklace into 5 segments of equal weight. The idea behind this algorithm is to increment two indices through the bead array and calculate the weight of the segment between them, keeping track of the start and end points of the segments that add up to the correct target weight. The target weight is found by calculating the total weight of all the beads, then dividing by 5. Afterwards, the algorithm checks to see if any of the valid segments create a closed circle. If so, a solution has been found.

b) Pseudocode

```
Let N be the array of length n containing the weight of each bead
Let totalWeight = 0
For every bead in N:
      totalWeight += N[bead]
Let targetWeight = totalWeight / 5

Let S be an array of length n with all items equal to -1
Let segmentStart, segmentEnd = 1
Let segmentWeight = N[1]
While segmentStart <= n:
      If segmentWeight < targetWeight:
            segmentEnd += 1
            segmentWeight += N[(segmentEnd % n) + 1]
      If segmentWeight > targetWeight:
            If segmentStart == segmentEnd:
                  Print NO and return
            Else:
                  segmentWeight -= N[segmentStart]
                  segmentStart += 1
      If segmentWeight == targetWeight:
            S[segmentStart] = segmentEnd
            segmentEnd += 1
            segmentWeight += N[(segmentEnd % n) + 1]

For every element e in S:
      Let index = 0
      Let nextSegment = e
      While index < 4 and S[nextSegment] != -1
            nextSegment = S[(nextSegment + 1) % n]
            index += 1
      If index == 4 and nextSegment == (e-1) % n
            Print YES and return
Print NO
```

c) Proof of Correctness

By incrementing a start and end index around the circle indepedent of each other, the algorithm is guaranteed to find every segment of numbers that adds up to the target length. The incrementing start index guarantees that every possible segment is checked, and since the numbers are concrete, each segment only needs to be checked one time. Incrementing the end index based on comparisons to the target length allows the algorithm to determine definitively whether the segment can add up to the target sum. This is because incrementing the end value without

incrementing the start will always add more to the sum, so incrementing until the item that the sum goes from less than the target to more than the target lets the algorithm know if the segment is the right length or not. Decrementing the end index is never necessary because if it were possible for the target sum could be found by decrementing the end index, the test on the previous segment would have detected it. Since the algorithm has a list of every possible segment that can add up to the required amount, if it cannot form a closed circle from 5 of those segments, it can guarantee that the division is impossible.

d) Running Time Estimate

O(n)

e) Running Time Estimate Reasoning

There are four operations in this algorithm that have a non-constant running complexity: the initial for loop on N, the for loop that populates the array S with -1, the while loop, and the for loop on S.

Since all operations inside the initial for loop have constant running time, the complexity of that loop is O(n). The same is true for the for loop that populates S.

The while loop will run until every element in the array passes through segmentStart. With every iteration of the while loop, it is guaranteed that either segmentStart or segmentEnd will be incremented. Since the loop breaks if segmentStart = segmentEnd with a weight greater than the target weight (which is less than the total weight of all beads summed together), it follows that segmentEnd cannot increment past segmentStart (it can't "outlap" segmentStart), so while segmentStart iterates n times, segmentEnd can iterate at most 2n times. Thus, the while loop has a worst-case time complexity of 3n.

The final for loop contains a while loop that will run at most 5 times for each element in S, which would be at worst case 5 + 5 + 5 + ... + 5 n times, or 5n.

So the final running time of the algorithm is n + n + 3n + 5n = 10n, which is equivalent to O(n).