COS 350 HW 1

Taidgh Robinson

January 2018

2.

It is easiest to think of this problem recursively. We will need to define a function that takes a current value, and a list which includes all the ints that have led to that current value. We then check to see if the current value is equal to n, and if it is we can return the string. If it is greater we stop. Else we call this function with each value of d_i added to the current value.

3.

My algorithm is correct, because it just checks all possible sums that can be generated using an array of ints that are less than our target. It's not terribly efficient as it has lots of recursive calls, but all they do is add and check values, which can happen in constant time for our assumption.

4.

```
#I've been writing a lot of Python so I sorta yanked the sytax from that.

#This assumes the main call for JJ is passed 0 as sum, and an empty array for cu

JJ(n, D, sum, curr_vals)

    if sum == n:
        return curr_vals


    for i = 0 to len(D):

        if (sum += D[i] <= n):
            temp = curr_vals
            temp.append(D[i])
            JJ(n = n, D = D, sum += D[i], temp)
```

5.

$O(\sum_{n=0}^{n} len(D)^n)$

This is the worst case, which has a D of all ones. The reason ones is the worst case, is because it will take n additions to reach n, and it will always reach n. It is then helpful to visualize this as a branch of recursive calls. Each one of those nodes is making n calls, to iterate over D and so that means that we get the number of nodes in an len(d)-ary tree, times n.

6.

$O(\sum_{n=0}^{n} len(D)^n)$

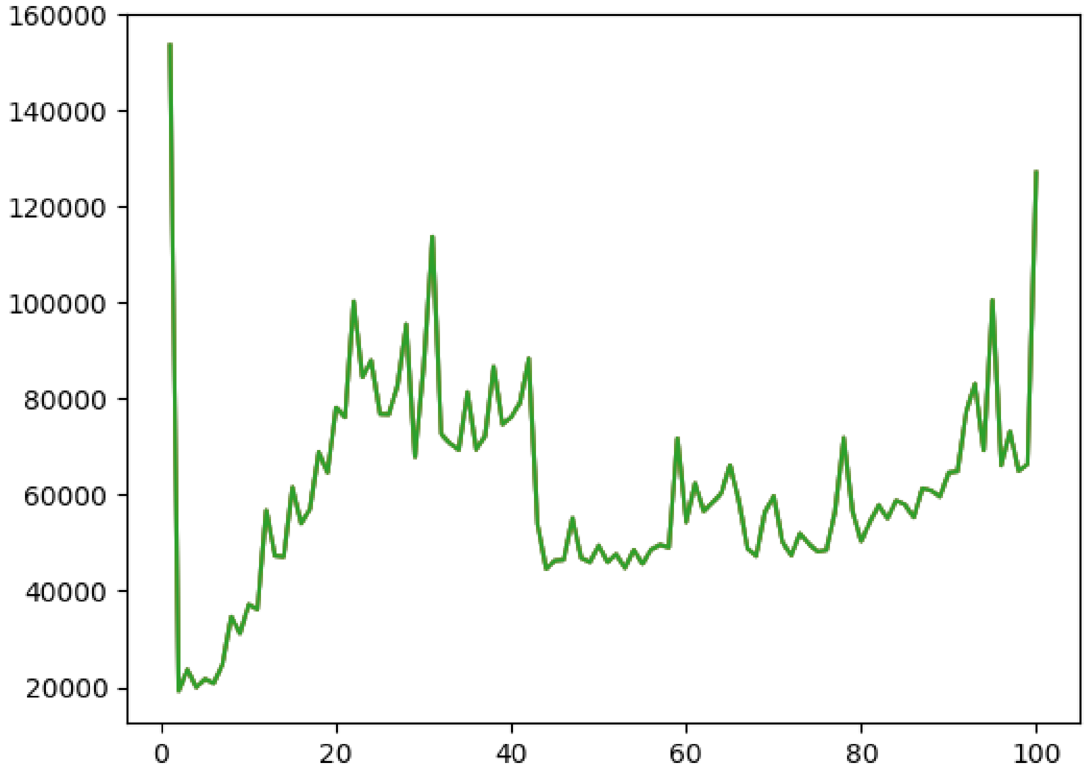
Again this is the worst case, where the entire sigma function comes from the number of nodes, when each element in the array is 1. The n in this function comes rather for the n number of elements in each array.

8.

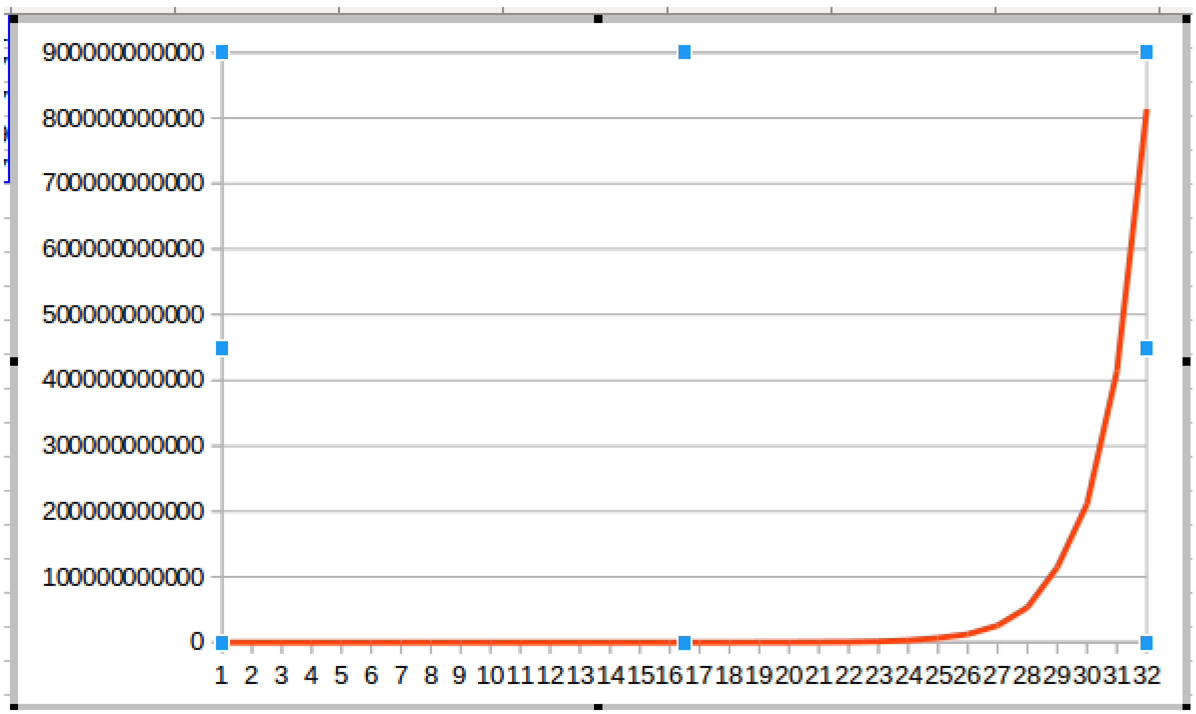
As we can see from the figures, the run time of this function is really really bad. It grows at increasing rates, and for higher values of len(D), I couldn't even get past 20 tests.

1 Figures

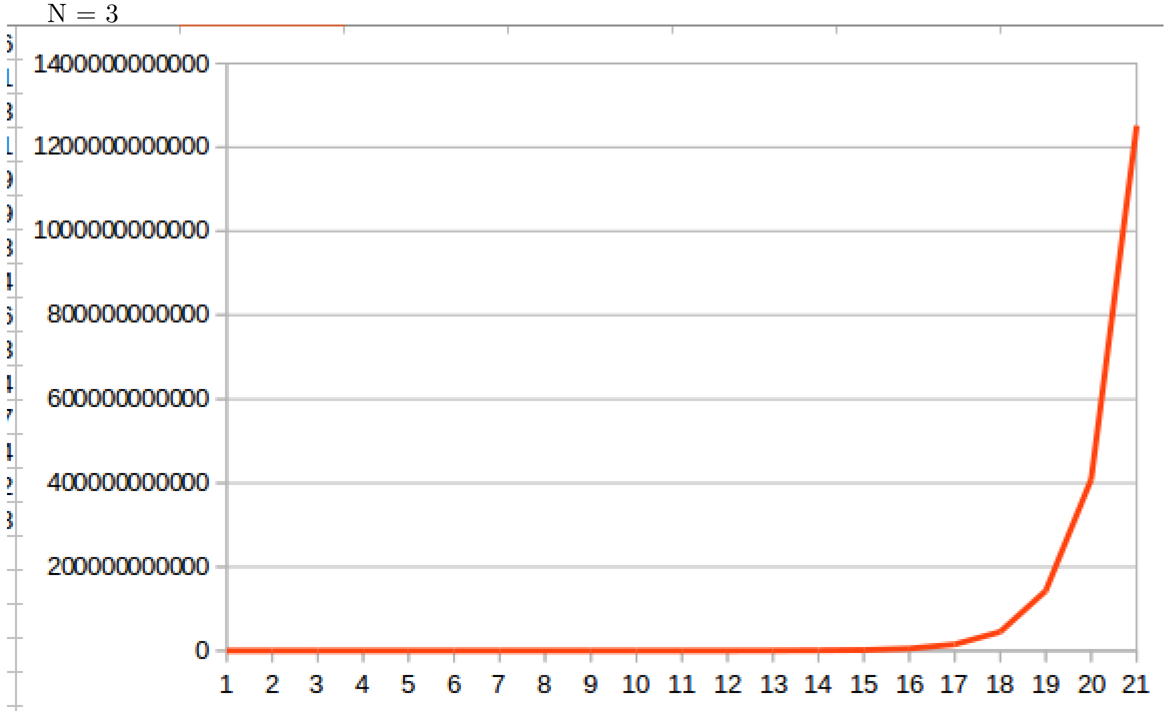
N = 1



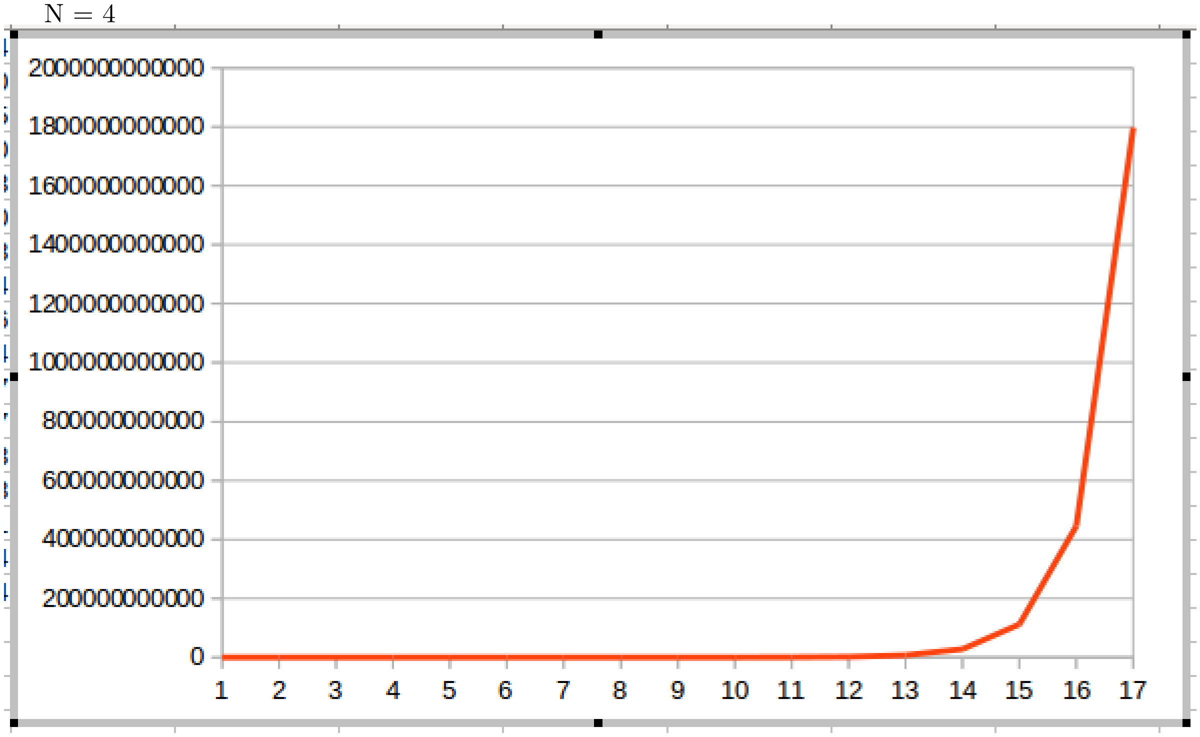
N = 2


900000000000
800000000000
700000000000
600000000000
500000000000
400000000000
300000000000
200000000000
100000000000
0
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32

N = 3


1400000000000
1200000000000
1000000000000
800000000000
600000000000
400000000000
200000000000
0
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21

N = 4


2000000000000
1800000000000
1600000000000
1400000000000
1200000000000
1000000000000
800000000000
600000000000
400000000000
200000000000
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17