

Assignment 1

15.4.2021

Name: Jiawei Wang

ID: s1239587

Question 1 (15 marks)

$$\begin{aligned}
 (a) \quad (265, 380) &\rightarrow (265, 115) \rightarrow (150, 115) \rightarrow (35, 115) \\
 &\rightarrow (35, 80) \rightarrow (35, 45) \rightarrow (35, 10) \\
 &\rightarrow (25, 10) \rightarrow (15, 10) \rightarrow (5, 10) \\
 &\rightarrow (5, 5) \Rightarrow \text{g.c.d} = 5
 \end{aligned}$$

$$\begin{aligned}
 (b) \quad (265, 380) &\rightarrow (265, 115) \rightarrow (115, 35) \rightarrow (35, 10) \\
 &\rightarrow (10, 5) \Rightarrow \text{g.c.d} = 5
 \end{aligned}$$

Question 2 (35 marks)

$$(a) \text{ line 3 takes average } \frac{0+1+2+\dots+n-1}{n} = \left\lfloor \frac{n-1}{2} \right\rfloor \text{ additions. } - O(n) \text{ time}$$

line 4 takes $O(1)$ time to store the sum to $Y[i][j]$

$$\begin{aligned}
 \text{So the for-loop in line 1 and line 2 takes } & - O(1) \text{ time} \\
 & \frac{(0+1+2+\dots+n-1)}{\text{line 1 and line 2}} \cdot \frac{O(n+1)}{\text{line 3 and line 4}} \\
 = \frac{n(n-1)}{2} \cdot O(n+1) &= O(n^3) \text{ time.}
 \end{aligned}$$

Therefore, the time complexity of function is $O(n^3)$

(b) The bottleneck is that there are a lot of repeated calculations.

For example: the value of adding all the entries $X[\lfloor \frac{n}{2} \rfloor]$ to $X[n-1]$

We do this addition for about $\lfloor \frac{n}{2} \rfloor$ times, we can reuse the sum we already put into Y .

(c) Please check q2.py. function func1()

Question 2 (cont'd)

(d) the pseudo-code of q2.py func1c):

```

1. for i ← 0 to n-1 do
2.   for j ← i to n-1 do
3.     if j is 0
4.       then Y[i][j] = X[i]
5.     end if
6.     else Y[i][j] = Y[i][j-1] + X[j]
7.   end for
8. end for

```

from line 3 to line 6:
takes $O(1)$ time

So the for-loop in line 1 and line 2:
takes $(0+1+2+\dots+n-1) \cdot O(1)$

$$= \frac{n(n-1)}{2} \cdot O(1) = O(n^2) \text{ time}$$

Therefore, the time complexity of func1c) is $O(n^2)$.

Question 3 (25 marks)

(a) local minimum.
left complete binary tree: 0
right complete binary tree: 1

(b) Please check q3.py. function: traverse1c)

(c) In function traverse1c). I used a property of the local minimum number:
— We must have at least one local minimum number in every possible route from the top of the tree to bottom.

So we just need to search one route to find one of the local minimum number.

In the worst case, I traverse from top to bottom, and the local minimum is on the leaf. I called traverse1(node.left) recursively till I hit the leaf.
Then we have:

In the worst case:

$$T(n) = T\left(\frac{n}{2}\right) + O(1)$$

Since $n^{\log_a a} = n^{\log_2 2} = 1$, we can apply case 2 of the master theorem.

We can conclude that the time complexity of our function is $O(n^{\log_2 2}) = O(\log n)$.

Question 4 (25 marks)

(a) please check q4.py.

(b) The algorithm in (a) is a greedy algorithm.

— Every step we try to find the largest value of coin to minimize the total amount of coins.

Because of that, we can make sure our final output is the minimal list of coins in descending order.

In each amount of coins, first we calculate the number of it by using `//` then we subtract the total value of this amount coins to make sure the final coin value equals \$value.

When `value // 1` is zero, which means the final coin value equals \$value.