NEHA GUPTA
B00818087

Homework - 4

Question 1

Activity	1	2	3	4	5
Start Time	0	2	7	4	10
Finish Time	3	9	11	9	15
Duration	3	7	4	5	5

According to this, activity a_1 and a_3 are having least duration, remaining overlapping and hence cannot be selected.

On the other hand, optimal solution will be

a_1, a_4 and a_5.

Hence proved that we do not get optimal solution by solving it with least duration first method.

i) always selecting the compatible activity that overlaps the fewest other remaining activities

Activity	1	2	3	4	5
Start	0	2	7	4	10
Stop	3	9	11	9	15
Overlaps	1	3	3	2	1

Solution contains a_1 and a_5

Optimal solution is a_1 a_4 a_5

Hence proved that we don't get optimal solution by selecting activities that overlaps the fewest other remaining activities first method.

ii) Example

Activity	1	2	3	4	5	6
Start	10	2	7	4	10	0
Stop	3	9	11	9	15	12

Solution a_6

Optimal solution a_1 a_4 a_6

Hence proved that we dont get optimal soution by selecting activities with earliest start time.

Question-2

a) Greedy algorithm to make change using quaters, dems, nickels and pennies.

Initialize result and count array as empty.

find largest denomination i.e. 25, smaller than amount.

add the found denomination to result array and increment count array.

Subtract value of found denomination from V.

If amount == 0, break;

Optimal solution for n cents will exclude one coin, c, which will be the largest coin value.

$n \geq 25$, $c = 1$ which will give us least number of coins.

$10 \leq n < 25$, $c = 10$ which are replacing pennies and nickels, to get less no. of coins.

Hence, greedy algorithm is optimal solution.

(b) Let S_k be solution for C^k. Hence S_{k-1} will be the optimal solution for C^{k-1}.

(c) Assume $n = 22$

$C_0 = 1 \quad C_1 = 11 \quad C_2 = 20$

22 is obtained by c_1 coin and c_0 2 coins
but
optimal solution is $2c_1$ coins

(d)

```
greedyNk (amount, coin[], k)
    if amount == 0
        return 0

    for (i=0 to k) {
    temp = get(ci);
        if (amount > temp)
            return 1+ (greedyNk (
                amount - temp,
                coin, k);

        }
    return 0;
```

In every loop (recursive call) we are reducing the space and hence algorithm runs for $O(nk)$.