Q.1.A. Approach of selecting the activity of least dura.

Activity	start time	Finish time	Duration
A_0	0	1	1
A_1	1	3	2
A_2	3	6	3
A_3	6	10	4
A_4	10	11	1

Let us consider a activity set as shown above. The optimal sol^n for this activities should be

$$Optimal = \{ A_0, A_1, A_2, A_3, A_4 \}.$$

The greedy approach described i.e. that of selecting activities with least duration will give

$$Greedy = \{ A_0, A_4 \}.$$

Thus, this greedy approach doesn't work.

T=0	1	2	3	4	5	6	7	8	9	10	11
optimal → A_0	A_1	A_1	A_2	A_2	A_2	A_3	A_3	A_3	A_3	A_4	
Greedy → A_0										A_4	

Q.1.B. Approach of selecting activities with fewest overlaps

→ Consider following scenario:

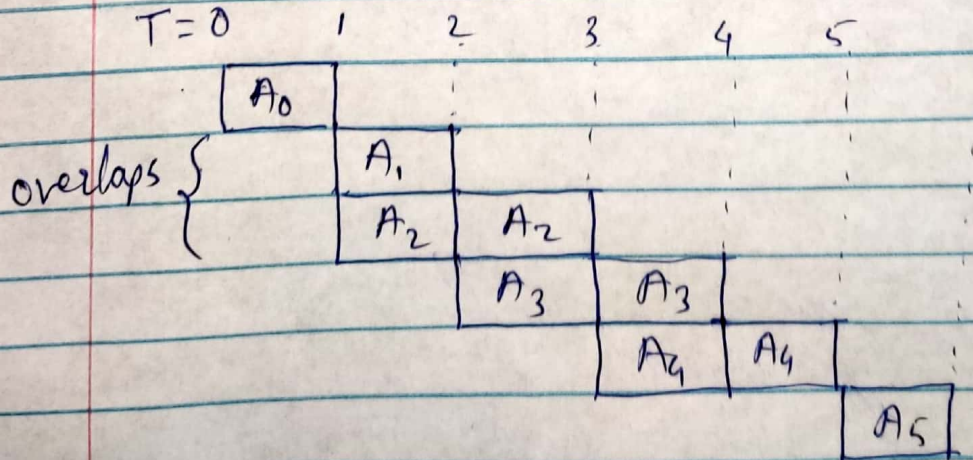
Activity	start time	finish time	no. of overlaps.
A_0	0	1	0
A_1	1	2	2
A_2	1	3	2
A_3	2	4	2
A_4	3	5	2
A_5	5	6	0.

The optimal sol^n for this case is
$\{A_0, A_1, A_3, A_5\}$ or/AND $\{A_0, A_2, A_4, A_5\}$.

The greedy approach of selecting activity with fewest overlap gives us

greedy sol = $\{A_0, A_5\}$.

Hence, this greedy approach doesn't yeild max set.



Q. 1. C. Approach of selecting activity of earliest start time

consider following set of activities

Activity	start time	finish time	Duration.
A_0	0	10	10
A_1	1	3	2
A_2	3	6	3
A_3	6	10	4

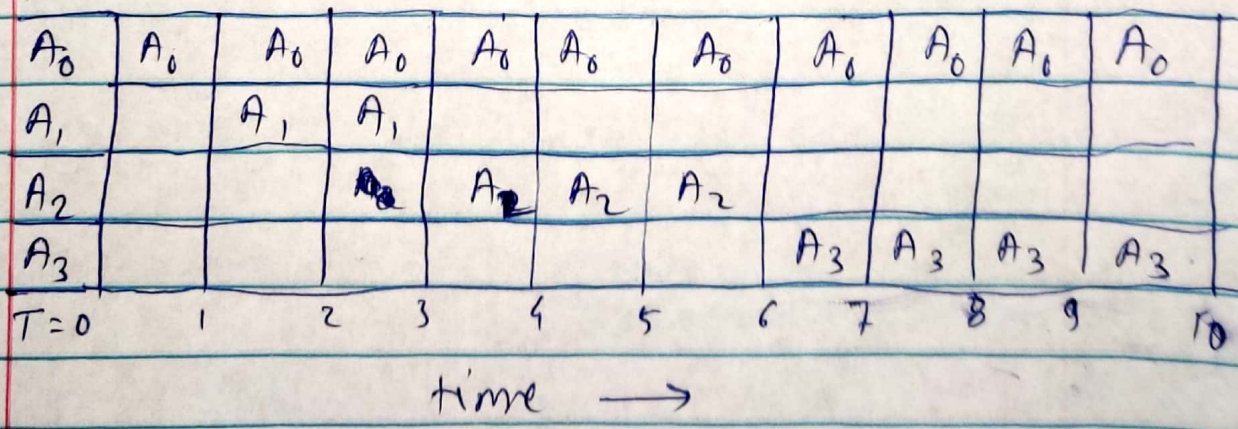
The optimal solution for this activity is:

$Set_{optimal} = \{ A_1, A_2, A_3 \}$.

The greedy approach of selecting the activity with earliest start time gives us

$Set_{greedy} = \{ A_0 \}$.

Thus, this greedy approach doesn't give max set



Q. 2. A.

In this case, we apply best possible soln to the current problem without considering the global solution at all.

For American coin currencies {25, 10, 5, 1} cents, at each step we will ~~apply~~ substract the largest possible coin denomination & increase the count by 1. till amount is zero.

```
int greedyCoins()
    denominations = {25, 10, 5, 1}.
    count, index = 0.
    while (amount > 0)
        if (amount >= denominations[index])
            amount = amount - denominations[index]
            ~~coins~~ count ++
        else
            index = index + 1

    return count
```

Prog:

Let us consider a scenario:
whenever the amount is greater than 25,
~~The~~ amount = amount - denominations[index]
will substract 25 out of amount.

This way, ~~the~~ we can represent a portion of amount with one quarter.
This will continue till amount is less than quarter.

In other words,
if n is the number of times quarter was substracted from an amount K;
K can be represented by at max n quarters + remaining amount.

~~The above loop continues~~
As the remaining amount cannot be represented by quarter, it is represented by next lower denomination i.e. dime. using same logic as above.
The loop continues till remaining amount is greater than zero.
The addition of number of quarters, dimes, nickel, penny gives the final amount.

Q.2. B. Let the denominations be

$c^0, c^1, c^2 \ldots\ldots c^k$ & amount be n

The amount n can be represented by which is similar approach to the previous ques.

$n = k_1 c^p + \text{reminder}$ s.t. $c^p \leq n$.

also, the reminder can be represented as

$\text{reminder} = k_2 c^b + \text{rem}'$ whereas $c^b \leq \text{reminder}$

$\therefore n = k_1 c^p + k_2 c^b + \text{rem}'$ & $p > b$.

$\vdots$

$n = k_1 c^p + k_2 c^b + k_3 c^{b'} \ldots\ldots c^0$.

$\therefore$ Thus, 'n' is a function of c where

$\sum_{j=1}^{n} k_j$ is the number of coins required.

as 'n' can be represented by only one combination of 'k' & 'c', the result will always be an optimal sol^n

Q.2.c. let the coins denominations be: 1, 4, 6. & amount be 8.

The greedy algorithm will: give

	coins
8 - 6	1
2 - 1	1
1 - 1	1
	3 coins

whereas the same amount can be represented by 4+4 i.e. using 2 coins.

Thus, greedy Sol^n doesn't yeild optimal solution for amount = 8 when denominations = {1, 4, 6}.

q.2. d.

```
int GreedyCoins (int amount arr denominations[])
  int coins = 0
  int index = 0  //denominations are sorted in descending order
  if (amount ≥ denominations [index])
        amount = amount - denomination [index]
        coins = coins + 1

  else,
        index = index + 1

  return coins
```