CSE 321 Homework 3

1) Loyalty program

```
def max_discount(set_of_stores, index=0, current_list=None):
    if current_list == None:
        current_list = []
    if index == len(set_of_stores):
        return calc_discount(current_list)        # O(1)
    include = max_discount(set_of_stores, index+1, current_list + [set_of_stores[index]])   # O(1)
    exclude = max_discount(set_of_stores, index+1, current_list)
    return max(include, exclude)                  # O(1)
```

* Find all the subsets of stores then calculate the discount each of them and compare with each other.

By forward substitution

$T(n) = 2 \cdot T(n-1) + O(1)$

$T(n) = 2 \cdot (2 \cdot T(n-2) + O(1)) + O(1)$

$T(n) = 2 \cdot (2 \cdot [2 \cdot T(n-3) + O(1)] + O(1)) + O(1)$

$T(n) = 2^k \cdot T(n-k) + [2^{k-1} + 2^{k-2} + \dots + 2 + 1] \cdot O(1)$

$k = n$

$T(n) = 2^n \cdot T(0) + (2^{n-1} + 2^{n-2} + \dots + 2 + 1) O(1)$ → $O(1)$ → base case

$T(n) = 2^n + (2^n - 1) O(1)$

Geometric Series → $2^n - 1$

$T(n) \in 2^n$

2) Distributed computer system

The algorithm generates all possible combinations of use-process pairs and processors.

Then it iterates through all the processors and calculates the cost for every combination.

It calculates for every combination and compares them with each other at every iteration. Then it returns the minimum costed combination.

For best case, average case and worst case the time complexity is same.

Because we use a brute force algorithm and we need to find all the possible solutions and pick the optimum one in it.

Generating possible schedules $\rightarrow n^n$

Assign process to processor $\rightarrow$ iterate through the processes $\rightarrow n$

Time complexity $\Rightarrow T(n) \in O(n^n n) \Rightarrow T(n) \in O(n^{n+1}) \in O(n^n)$

---

3) find_optimal_product_sequence(ports):

```
min_cost_sequence = []
min_cost = ∞
all_sequences = get_permutations(ports)        → n! × n

for sequence in all_sequences:
    cost = cost_of_sequence(sequence)          → assuming O(1)
    if cost < min_cost:
        min_cost = cost
        min_cost_sequence = sequence           } O(1)

return min_cost_sequence
```

$T(n) = n! \times n + O(1)$

$T(n) \in O(n! n)$

For best case, average case and worst case.

4) Min coins

```
def min_coins(coins - amount):
    if amount == 0:        } Base cases
        return 0
    elif amount < 0:       }
        return ∞

    minCount = ∞  → The thing we want to calculate

    for coin in coins:  → iterate through coins and call recursive function to evaluate each combination.
        if coin <= amount:
            count = 1 + min_coins(coins, amount - coin)  → Recursive part with decreased amount.

            if count < minCount:  } update the minCount with better one
                minCount = count

    return minCount
```

Best case

If the amount is some with the largest coin it is the best case.

It makes 1 recursive call and n iterations so $T(n) \in O(n)$

Average and worst case

$T(n) \in O(mn)$ m: amount, n: number of coins → n iterates over the coins. m → exploring with recursive calls.

5) This function is just like merge sort divides a list into 2 pieces. and recursively evaluates them. and finds the minimum in the base case case.

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

By master's theorem:

$a=2$, $b=2$ $f(n) = O(1)$

$n^{\log_b a} = n^1$ $f(n) < n$ so: $T(n) \in O(n)$