Emre Oytun
2001040041093 E.Oytun

1.

→ Firstly, check if the given array is empty.
— $\Theta(1)$

→ Initialize an empty setOfStores array. — $\Theta(1)$

→ Initialize maxSequence ← None
maxDiscount ← $-\infty$ — $\Theta(1)$

→ Call the recursive helper method by giving setOfAllStores, setOfStores, 0 (as curIdx) as parameters. — $\Theta(1)$

→ In recursive method:

- Check if all elements considered. — $\Theta(1)$
- If it's the case, then it's the base case:
  - → Calculate the total discount for generated sequence and compare it with maxDiscount. — $\Theta(1)$
  - → If necessary, copy the generated sequence to the maxSequence global variable. — $\Theta(n)$
    If not necessary, then just return. — $\Theta(1)$
- If it's not the base case:
  - → Add the current element to the setOfStores array and make a recursive call for next element.
  - → Remove the newly added current element from the setOfStores array and make a recursive call for next element.

$$T(n) = 2T(n-1) + 1 \qquad T(n-1) = 2T(n-2) + 1$$

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

$$= 2^3 T(n-3) + 2^2 + 2 + 1$$

$$\vdots$$

$$= 2^k T(n-k) + 2^{k-1} + 2^{k-2} + \cdots + 1$$

$T(0)$: $n-k=0$, $k=n$

$$T(n) = 2^k T(n-k) + \sum_{i=0}^{k-1} 2^k = 2^k T(n-k) + \frac{2^k - 1}{2-1}$$

$$T(n) = 2^n T(0) + 2^n - 1$$

Best – case: There is no copy at all when $n=0$.

So, $B(n) = 2^n + 2^n - 1 \in \Theta(2^n)$

Worst – case: There is always a copy when $n=0$.

So, $W(n) = 2^n \cdot n + 2^n - 1 \in \Theta(n \cdot 2^n)$

Average – case: Let assume probability of copying n each time when $n=0$ is p.

For $T(0)$: $\sum (\text{probability}) * (\text{cost}) = p \cdot n + (1-p) \cdot 1 \in \Theta(n)$

Therefore; $A(n) = 2^n \Theta(n) + 2^n - 1 \in \Theta(n \cdot 2^n)$

$$A(n) \in \Theta(n \cdot 2^n)$$

2.

→ Firstly, assignments and remaining Process arrays are initialized. — $\Theta(n)$

→ Then recursive helper method is called.

→ Algorithm generates all possible pairs by trying to assign all remaining processes one-by-one to the current processor.

→ When there is no process/processor remains $T(0)$:
- it calculates the new cost — $\Theta(n)$
- copies new sequence if necessary — $\Theta(n)$

* In each step, there are n recursive calls for $T(n-1)$ and there are swap operations before and after each call.

$$T(n) = nT(n-1) + n \qquad T(n-1) = (n-1)T(n-2) + (n-1)$$

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

$$= n(n-1)(n-2)T(n-3) + n(n-1)(n-2) + n(n-1) + n$$

$$\vdots$$

$$= n(n-1)(n-2)\cdots(n-k)T(n-(k+1)) + n(n-1)(n-2)\cdots(n-k) + n(n-1)(n-2)\cdots(n-k+1) + \cdots + n$$

$T(0)$

$n-(k+1)=0$

$k+1=n$

$k=n-1$

$$T(n) = n!\,T(0) + n! + \frac{n!}{1} + \frac{n!}{1.2} + \cdots$$

$$\cdots + \frac{n!}{1.2\cdots(n-1)}$$

$$T(n) = n!\,T(0) + n!\left(1 + \frac{1}{2!} + \cdots + \frac{1}{(n-1)!}\right) + n!$$

$$T(n) = n!\,T(0) + n!\left(\sum_{i=1}^{n-1} \frac{1}{i!}\right) + n!$$

* Let analyze $\sum_{i=1}^{n-1} \frac{1}{i!}$:

Upper bound: All terms are smaller or equal to 1.
There are $(n-1)$ terms.

So, $\sum_{i=1}^{n-1} \frac{1}{i!} \leq (n-1).1$

$$\sum_{i=1}^{n-1} \frac{1}{i!} \in O(n)$$

Lower Bound:

$$\sum_{i=1}^{n-1} \frac{1}{i!} = \sum_{i=1}^{\lfloor \frac{n-1}{2} \rfloor} \frac{1}{i!} + \sum_{i=\lfloor \frac{n-1}{2} \rfloor + 1}^{n-1} \frac{1}{i!} \geqslant \sum_{i=1}^{\lfloor \frac{n-1}{2} \rfloor} \frac{1}{i!}$$

$$\sum_{i=1}^{\lfloor \frac{n-1}{2} \rfloor} \frac{1}{i!} = 1 + \frac{1}{1.2} + \cdots + \frac{1}{1.2\cdots\lfloor \frac{n-1}{2} \rfloor}$$

↳ All terms $\geqslant 1$
There are $\lfloor \frac{n-1}{2} \rfloor$ terms.

So, $\sum_{i=1}^{\lfloor \frac{n-1}{2} \rfloor} \frac{1}{i!} \geqslant \frac{n-1}{2}.1$.

Therefore, $\sum_{i=1}^{n-1} \frac{1}{i!} \geq \frac{n-1}{2}$

$$\sum_{i=1}^{n-1} \frac{1}{i!} \in \Omega(n)$$

Since, $\sum_{i=1}^{n-1} \frac{1}{i!} \in \Omega(n)$

$$\sum_{i=1}^{n-1} \frac{1}{i!} \in O(n) \quad \Big/ \quad \sum_{i=1}^{n-1} \frac{1}{i!} \in \Theta(n)$$

* In best case ($n=0$), there are always n operations and n optional operations according to comparison. Therefore;

T(0) worst-case: $W(n) = 2n \in \Theta(n)$
T(0) best-case: $B(n) = n \in \Theta(n)$
T(0) average-" : $A(n) = W(n) = B(n) \in \Theta(n)$

$T(n) = n! \cdot \Theta(n) + n! \cdot \Theta(n) + n!$

Therefore, $\boxed{T(n) \in \Theta(n \cdot n!)}$

* It holds for best-case, worst-case and average case since there'll be no difference in $T(n)$.

3.

→ Firstly, remainingParts and sequence arrays are initialized by iterating through indexes. — $\Theta(n)$

→ Then minCost is initialized as infinite for forward comparision.

Assumption: There's no energy cost for the first element since there's no change from one part to another.

→ Since there is no energy cost for first elements, first recursive calls are made by taking all elements as first item. There're 2 swap operations inside.

— $nT(n-1)$ calls

— $+n$ for constant-time swap operations done in each operation.

→ Recursive helper function is also doing the same as above. In each call, it tries all possibilities for next assemble part, but this time there is a cost of moving from the current part to the next part.

→ When there's only 1 element remains $T(1)$:

— it compares the total cost with min cost

— copies new sequence if necessary — $\Theta(n)$

$T(n) = nT(n-1) + n$ $\qquad T(n-1) = (n-1)T(n-2) + (n-1)$

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

$= n(n-1)(n-2)T(n-3) + n(n-1)(n-2) + n(n-1) + n$

$\vdots$

$= n(n-1)(n-2)\cdots(n-k)T(n-(k+1)) +$

$n(n-1)(n-2)\cdots(n-k) + n(n-1)(n-2)\cdots(n-k+1) +$

$\cdots + n$

$T(1)$

$n-(k+1)=1$

$n = k+2$

$\boxed{k=n-2}$

$$T(n) = \frac{n!}{1}T(1) + \frac{n!}{1} + \frac{n!}{2} + \cdots + \frac{n!}{1\cdot 2\cdots(n-1)}$$

$$T(n) = n!\,T(1) + n!\sum_{i=1}^{n-1}\frac{1}{i!}$$

Analysis of this part is made in the question 2's answer. It's $\Theta(n)$.

* In base case $T(1)$:

- There's always a comparision. — $\Theta(1)$.
- There's a copy operation accordingly.

Best-case: No copy in $T(1)$. Only comparision.

$$B(n) = n!\,\Theta(1) + n!\,\Theta(n) \in \Theta(n \cdot n!)$$

Worst-case: There's always a copy in $T(1)$.

$$W(n) = n!\,\Theta(n) + n!\,\Theta(n) \in \Theta(n \cdot n!)$$

Therefore, average-case complexity is $\Theta(n \cdot n!)$.

Because: $A(n) = B(n) = W(n) \in \Theta(n \cdot n!)$ //

4.

Algorithm minCoinsChange (coins[0:n-1], targetAmount):

1) Checks if there is no target amount anymore.
(targetAmount == 0) Returns 0, if it is the case.

2) Initializes minChange variable as $+\infty$.

3) Iterates through coins as below:

for every Coin in coins:

Check if Coin is smaller or equal to targetAmount:

If it's the case, then make a recursive call by subtracting the Coin's amount from the target amount. The purpose here is to get the count when the Coin is included.

result = minCoinsChange(coins, targetAmount − Coin)

If result is not −1 (So there is a valid solution), and if result+1 < minChange (So the change amount is smaller than the minChange):

Then assign result+1 to minChange.

minChange = result + 1

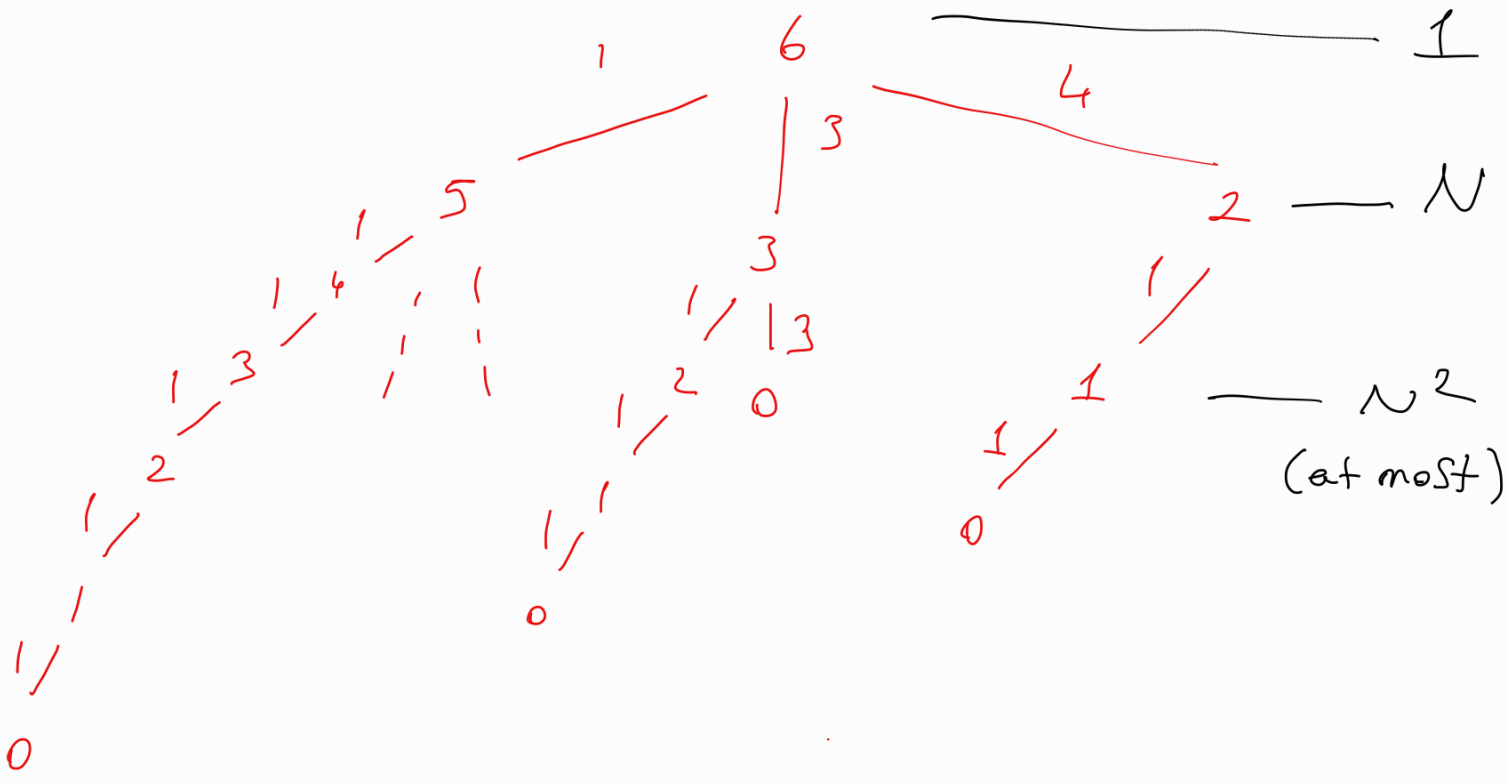
if minChange != ∞ and minChange != −1:

return minChange (If there is no result found)

return −1

Analysis: Coins = {1, 3, 4}

Target = 6



* T = target
  N = number of coins

* At the worst-case, the tree's depth is T and at most the tree is full.

Total number of calls:

$$1 + N + N^2 + \cdots + N^T = \sum_{i=0}^{T} N^i$$

$$\sum_{i=0}^{T} N^i = \frac{N^{T+1} - 1}{N - 1} \in O(N^T)$$ //

(5) * Since it makes recursive calls for both left and right half parts:

$$T(n) = 2T(n/2) + 1$$

→ 1 represents the constant time. Otherwise, the exact number is different of course.

By Master's Theorem:

$a = 2$, $b = 2$ $\rightarrow$ $\log_b a = \log_2 2 = 1$ (n^1)

$f(n) = n^0$ $\rightarrow$ $n^1 > n^0$, so $T(n) \in \Theta(n)$

→ $T(n) \in \Theta(n)$ is true if $n = 2^k$ format.

By Interpolation:

→ $T(n)$ is non-decreasing ✓

→ n is non-decreasing ✓

→ $c \cdot n \in \Theta(n)$, n is Θ-invariant under scaling ✓

Therefore, $\boxed{T(n) \in \Theta(n)}$ holds for all n.