Even after using the inclusion-exclusion method, the time complexity is $O(2^n)$. This is because, even after deleting identical sub-issues through inclusion and exclusion (for example, include 8 and do not include 8), other types of identical problems are recalculated (For example, two coins C1,C2 may produce the sum V, although another combination of C3,C4 may exist with the same sum). So here we are revisiting the same kind of states again and over. To resolve this issue, we can apply the memorizing method, which reduces the time complexity to O(nV).

Memoization:

In the memorization table,

1. Columns of the table represent all possible sums from 0 to the target sum (V).
2. And the coins' denominations need to be sorted in ascending order.
3. Rows of the table represent the different coin denominations available.
4. The element in the position (i,j) represents – the min number of coins required to obtain the sum j using the first i number of coins.

Initial Conditions –

a. (0,0) value is set to be 0 because 0 value can be obtained using 0 number of coins
b. (i,0) value will be 0 for all i
c. (0,j) value will be set to a very large value (infinity) for all j >0
d. When ever value of ith coin is less than j , (i,j) = (i-1,j)

Methodology –

We will be using the below method for filling (i,j) of the table-

For (i,j) –

a. We can use the ith coin

(i-1, j-V(i))+1 : because when we use the ith coin , the value now reduces to j-V(i) , and we add 1 because we already used one coin

b. We we won't be using the ith coin

(i-1,j) : because if we donot use the ith coin, the value to be achieved remains the same as j but we will be now using remaining i-1 coins

If (value of coin i> j):(i,j) = (i-1,j)
else:(i,j) = min{(i-1,j-V(i))+1,(i-1,j) }

Memoization Table:

Let say we want to obtain the sum V=11 , using the set of coins – {1,2,5,6,8}

Denomination	No.of coins	Value											
		0	1	2	3	4	5	6	7	8	9	10	11
	0	0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
1	1	0	min(1+0,∞)	min(1+∞,∞)	min(1+∞,∞)	(1+∞,∞)	(1+∞,∞)	(1+∞,∞)	(1+∞,∞)	(1+∞,∞)	(1+∞,∞)	(1+∞,∞)	(1+∞,∞)
			1	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
2	2	0		(1+0,∞)	(1+1,∞)	(1+∞,∞)	(1+∞,∞)	(1+∞,∞)	(1+∞,∞)	(1+∞,∞)	(1+∞,∞)	(1+∞,∞)	(1+∞,∞)
			1	1	2	∞	∞	∞	∞	∞	∞	∞	∞
5	3	0					(1+0,∞)	(1+1,∞)	(1+1,∞)	(1+2,∞)	(1+∞,∞)	(1+∞,∞)	(1+∞,∞)
			1	1	2	∞	1	1	2	3	∞	∞	∞
6	4	0						(1+0,1)	(1+1,2)	(1+1,3)	(1+2,∞)	(1+∞,∞)	(1+1,∞)
			1	1	2	∞	1	1	2	2	3	∞	2
8	5	0								(1+0,2)	(1+1,3)	min(1+1,∞)	min(1+2,2)
			1	1	2	∞	1	1	2	1	2	2	2

Thus, with this table, we can perform backtracking to identify the coins that yield a sum of 11 with the smallest quantity of coins. Hence, by utilizing the memoization table, we are executing nV calculations. Consequently, we can decrease our time complexity from O(2^n) to O(nV) with the application of memoization.

Note: It is essential to sort the denominations prior to initializing the memoization table, which incurs its own time complexity for the sorting process.

Problem:

What if the repetition of the coins is allowed to make the sum ?

Methodology –

If (value of coin i> j): (i,j) = (i-1,j)

else: (i,j) = min{(i,j-V(i))+1,(i-1,j) }

Denomination	No.of coins	Values											
		0	1	2	3	4	5	6	7	8	9	10	11
1	1	0	1	2	3	4	5	6	7	8	9	10	11
2	2			min(1+0,2)	min(1+1,3)	min(1+1,4)	(1+2,5)	(1+2,6)	(1+3,7)	(1+3,8)	(1+4,9)	(1+4,10)	(1+5,11)
		0	1	1	2	2	3	3	4	4	5	5	6
5	3						(1+0,3)	(1+1,3)	(1+1,4)	(1+2,4)	(1+2,5)	(1+1,5)	(1+2,6)
		0	1	1	2	2	1	2	2	3	3	2	3
6	4							(1+0,2)	(1+1,2)	(1+1,3)	(1+2,3)	(1+2,2)	(1+1,3)
		0	1	1	2	2	1	1	2	2	3	2	2
8	5									(1+0,2)	(1+1,3)	(1+1,2)	min(1+2,2)
		0	1	1	2	2	1	1	2	1	2	2	2

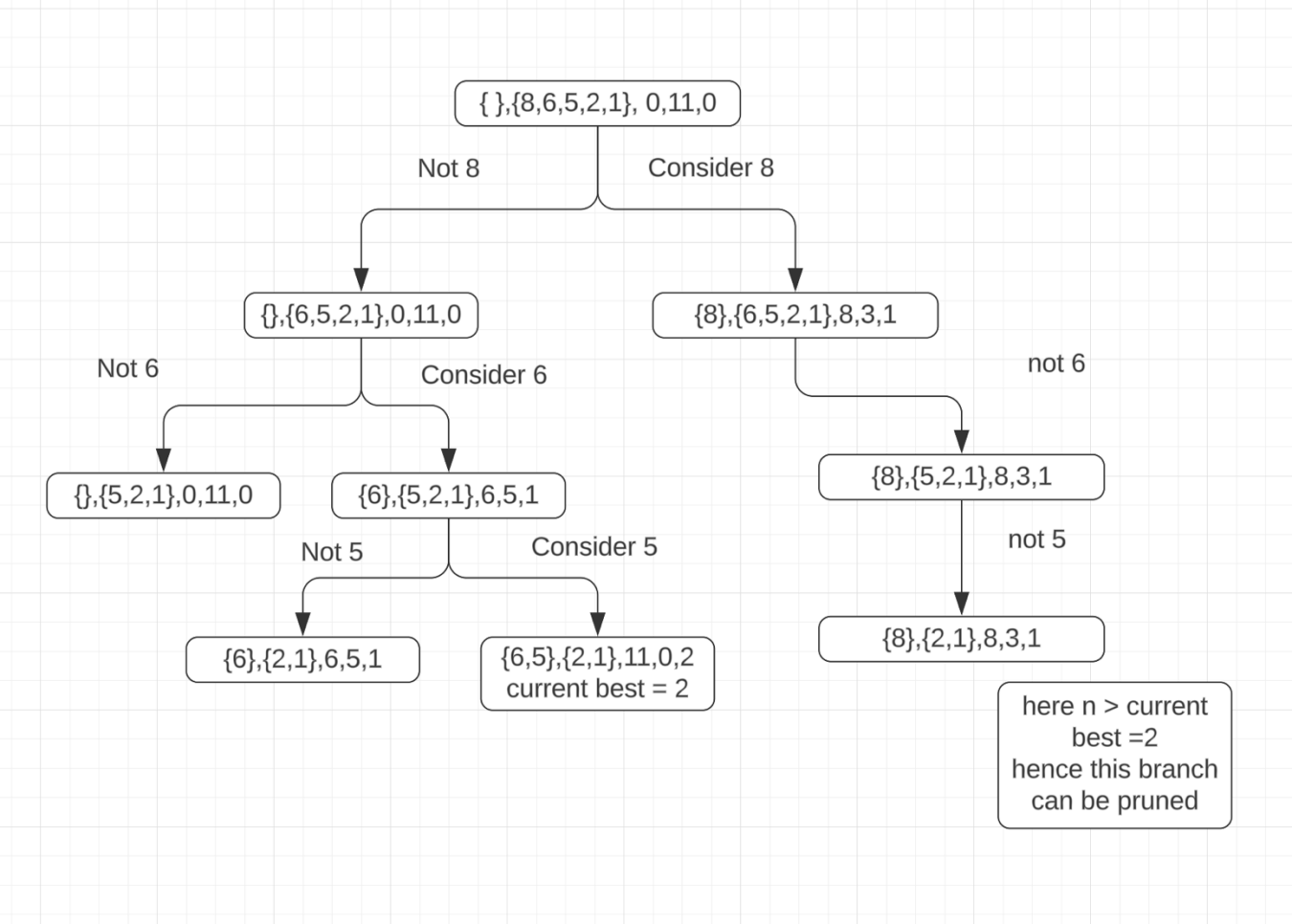
Thus, by utilizing the memoization table, we are executing nV calculations.
The time complexity is O(nV).

Pruning:

In the inclusion-exclusion method, we aimed to eliminate the issue of similar subproblems by avoiding re computation with the same set of coins. We now employ pruning to enhance the algorithm.

Lets consider, we start depth-first traversing following a particular path in the tree.

Suppose a -> b -> c -> d -> e and let the minimum number of coins found in that traversal be n. So, if while traversing through another path if I find the number of coins to be greater than or equal to n and still the pending cost p is nonzero , then we can eliminate the entire space and need not continue as the solution we are getting through that path is worst than the solution we got earlier (i.e. we prune the entire branch). And this method is known as branching and bounding.



Pseudo Code:

```
Coins(U,P,x,y,n){
        if (y = 0) return (U, n)
        if (y < 0) return (NULL, ∞)
        if (y < 0) return (NULL, ∞)
        if(C.B<=n ) return (NULL, ∞)
        Pmin = NULL, dmin = ∞
        for(i=1 to n) {
        ⟨s1,m1⟩ = Coins(U+{Ci},P-{Ci}, U+Ci, P-Ci, n+1)
        ⟨s0,m0⟩ = Coins(U,P-{Ci}, U, P, n)
        if (m1<m0) {
                Umin ⟵ U+{Ci}
                m⟵ m1
                        }
        Else{
                Umin ⟵ U
                m⟵ m0 }
                }
If(m<C.B) update C.B⟵m
return (Umin,m) }
```

Choice of split in divide and conquer:

The time complexity in divide and conquer mainly depends on the choice of split .

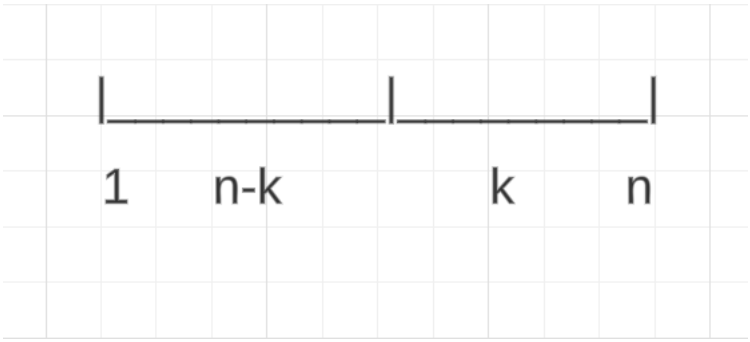
T(n) = T(k)+T(n-k)+f(n)

T(n) = a.T(n/b)+f(n)

T(1) = c

Where a ≥ 1, b> 1,c> 0

and f(n) is asymptotically positive.



As per master's theorem,

$$T(n) \in \begin{cases} \Theta(n^{\log_b a}) & \text{Case 1: if } a > b^d \text{ or } d < \log_b a \\ \Theta(n^d \log n) & \text{Case 2: if } a = b^d \text{ or } d = \log_b a \\ \Theta(n^d) & \text{Case 3: if } a < b^d \text{ or } d > \log_b a \end{cases}$$