**Experimental Analysis on Naïve Recursive and Greedy Algorithms solving Change-Making Problem**

**Introduction:**

In this report, I am detailing the analysis done by me over the Naïve Recursive and Greedy Algorithms approaches in solving the Change-Making problem on different coin systems.

**Importance:**

As a developer, we need to solve the real-world problems such as handling vending machines, finding cut-rod problems. There will be lots of approaches to solve a problem but there exists only one significant approach to solve in an efficient way. So, it is important to understand which approach we should follow in solving the problem using minimal resources and we can understand by doing the experiments!

**Theoretical Analysis:**

Let’s start with the theoretical analysis in which we will go through the pseudocodes of naïve recursive algorithm and Greedy algorithm that try to solve coin-change problem.

*Change-Making Problem:* The problem involves of making change for n cents using the fewest number of coins. Assume that each coin’s value is an integer. Assume that the coin system consists of k denominations of coin, and they are d1, d2, …, dk.

Note: Let us assume that given denomination set can solve any given n amount

Example: let us say n = 9,k = 2, d = [2,4]. We cannot solve n using d. we assume such weird coin system input will not be given to the algorithms.

*Pseudocode of Naïve-Recursive approach:*

**Coinchange\_Rec(n, d, k):**

1. count = +ve infinity # Considering positive infinity value as max default count
2. if n <= 0 # returns 0 if considered coin combination will not give required value
3. return 0
4. for each\_coin in range(0,k):
5. if d[each\_coin] <= n )
6. count = min(count,Coinchange\_Rec(n-d[each\_coin],d ,k)+1)

#Calling recursion to solve sub problems

1. return count

where *n* is amount for which change is required, *d* is denomination list and *k* denote number of denominations. The program returns the *count* value which gives the minimum number of coins used to give change to the users.

The time complexity for this algorithm will be T(n) = Θ (k\*2n )

*Pseudocode of Greedy approach:*

ChangeMaking\_greedy(n,d,k):

1. sort d array in decreasing order based on density #Takes Θ(k log2k)
2. coins\_used = []
3. for i in range(0,k):
4. if n//d[i] >= 1:
5. coins\_used.extend([d[i]]\*(int(n/d[i]))) #appending valid coin values to the list
6. n = n - (n//d[i]\*d[i]) #reducing the input amount value by maximum number of - - times current coin can be used
7. return coins\_used

where n is amount for which change is required, d is denomination list and k denote number of denominations. The program returns the coins used list that vending machine gives to the users.

The Line number 1 will take Θ(k log2k) time for sorting

The time complexity from line 2 to 7 will be T(n) = Θ (k)

The entire algorithm will take Θ(k log2k) time for execution as we ignore Θ (k)

**Practical Analysis:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Algorithm | Type of Input | Input Size | Count | Execution Time in Nano Seconds |
| **Naïve recursive Algorithm** | US Coin System | 11 | 2 | 201900 |
| 23 | 5 | 990800 |
| 31 | 3 | 11684900 |
| 51 | 3 | 9961724200 |
| 73 | 7 | 6.22605E+12 |
| Weird Coin System | 11 | 2 | 217700 |
| 23 | 1 | 1305400 |
| 31 | 4 | 12947600 |
| 51 | 3 | 9827521200 |
| 69 | 3 | 4.12707E+12 |
| **Greedy Algorithm** | US Coin System | 11 | [10, 1] | 173800 |
| 23 | [10, 10, 1, 1, 1] | 249800 |
| 31 | [25, 5, 1] | 86800 |
| 51 | [25, 25, 1] | 241900 |
| 73 | [25, 25, 10, 10, 1, 1, 1] | 211400 |
| 83 | [25, 25, 25, 5, 1, 1, 1] | 148200 |
| 91 | [25, 25, 25, 10, 5, 1] | 208200 |
| 99 | [25, 25, 25, 10, 10, 1, 1, 1, 1] | 216400 |

*Below are the Graphs plotted for the above data*

The above figure represents the output of Naïve Recursive approach for US Coin System

The above figure represents the output of Naïve Recursive approach for Weird Coin System

From the above figures we can observe that the time growth is exponential when n increases.

The above figure represents the output of Greedy Algorithm for US Coin System

For n = 69 the output is [25, 25, 10, 5, 1, 1, 1, 1] and the time taken is 211400 nano seconds for the weird coin system by the Greedy

**Correctness:**

The important phase is to talk about the correctness of actual results given by both the algorithms.

Naïve Recursive algorithm beats the Greedy algorithm in correctness as Greedy algorithm will always not give the correct output.

For example, from the above results of weird coin system, n = 69, d = [1,5,10,23,25], the correct output is,

Coins required = [23,23,23] which is count 3, Naïve recursive gave the correct output. Whereas Greedy gave output as [25, 25, 10, 5, 1, 1, 1, 1] which is not optimal.

Hence, we can conclude that Naïve recursive can give the output accurately but Greedy algorithm fails!

**Comparison:**

From the above results we can observe that Naïve recursive approach gives the correct output but does take the exponential time and more space (as recursion uses stack memory) which is not a good thing for any organization to use.

Greedy algorithm works for some type of coins and will not for some coin systems but takes linear time. We can say that Greedy approach is also not a good way for using in any coin system blindly.

**Performances:**

As per my understanding the theoretical and practical analysis are consistent, and the only thing that I was surprised is why don’t we make naïve recursive a better algorithm by making it to concede less time and of course it is giving correct output!

Greedy algorithm is absolutely working fine for the US Coin System but sadly not working for some weird coin systems, but its performance is appreciable as it takes linear time.

**Conclusion:**

To conclude with the analysis of results, we still need a better approach to find the solution for applying it to vending machines with which we can blindly use without worrying of any kind of denominations.