**Analysis of Change-Making Algorithms­­­­­­**

Final Output data for 3 algorithms (Recursion, Greedy Algorithm, Dynamic Programming):

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **RECURSION FOR CHANGE-MAKING PROBLEM** | | | | | |
| **INPUT** | | | | **OUTPUT** | |
| **COIN SYSTEM** | **DENOMINATION SET** | **CHANGES IN CENTS** | **NO. OF DENOMINATION** | **COINS COUNT** | **RUN TIME (Microseconds)** |
| US coin systems | [1,5,10,25] | 11 | 4 | 2 | 262 |
| [1,5,10,25] | 23 | 4 | 5 | 5626 |
| [1,5,10,25] | 31 | 4 | 3 | 21399 |
| [1,5,10,25] | 51 | 4 | 3 | 4189452 |
| [1,5,10,25] | 73 | 4 | 7 | 7198730751 |
| [1,5,10,25] | 83 | 4 |  |  |
| [1,5,10,25] | 91 | 4 |  |  |
| [1,5,10,25] | 99 | 4 |  |  |
| Weird coin systems | [1,5,10,23,25] | 69 | 5 | 3 | 1344105094 |
| **GREEDY ALGORITHM FOR CHANGE-MAKING PROBLEM** | | | | | |
| **INPUT** | | | | **OUTPUT** | |
| **COIN SYSTEM** | **DENOMINATION SET** | **CHANGES IN CENTS** | **NO. OF DENOMINATION** | **COINS USED** | **RUN TIME (Microseconds)** |
| US coin systems | [1,5,10,25] | 11 | 4 | 10, 1 | 11 |
| [1,5,10,25] | 23 | 4 | 10, 10, 1, 1, 1 | 7 |
| [1,5,10,25] | 31 | 4 | 25, 5, 1 | 6 |
| [1,5,10,25] | 51 | 4 | 25, 25, 1 | 6 |
| [1,5,10,25] | 73 | 4 | 25, 25, 10, 10, 1, 1, 1 | 6 |
| [1,5,10,25] | 83 | 4 | 25, 25, 25, 5, 1, 1, 1 | 10 |
| [1,5,10,25] | 91 | 4 | 25, 25, 25, 10, 5, 1 | 7 |
| [1,5,10,25] | 99 | 4 | 25, 25, 25, 10, 10, 1, 1, 1, 1 | 8 |
| Weird coin systems | [1,5,10,23,25] | 69 | 5 | 25, 25, 10, 5, 1, 1, 1, 1 | 8 |
| **DYNAMIC PROGRAMMING ALGORITHM FOR CHANGE-MAKING PROBLEM** | | | | | |
| **INPUT** | | | | **OUTPUT** | |
| **COIN SYSTEM** | **DENOMINATION SET** | **CHANGES IN CENTS** | **NO. OF DENOMINATION** | **COINS USED** | **RUN TIME (Microseconds)** |
| US coin systems | [1,5,10,25] | 11 | 4 | 10, 1 | 25 |
| [1,5,10,25] | 23 | 4 | 10, 10, 1, 1, 1 | 36 |
| [1,5,10,25] | 31 | 4 | 25, 5, 1 | 42 |
| [1,5,10,25] | 51 | 4 | 25, 25, 1 | 58 |
| [1,5,10,25] | 73 | 4 | 25, 25, 10, 10, 1, 1, 1 | 86 |
| [1,5,10,25] | 83 | 4 | 25, 25, 25, 5, 1, 1, 1 | 94 |
| [1,5,10,25] | 91 | 4 | 25, 25, 25, 10, 5, 1 | 97 |
| [1,5,10,25] | 99 | 4 | 25, 25, 25, 10, 10, 1, 1, 1, 1 | 107 |
| Weird coin systems | [1,5,10,23,25] | 69 | 5 | 25, 25, 10, 5, 1, 1, 1, 1 | 86 |

**Time graph of change-making recursive algorithm:**

**Pseudocode analysis of change-making recursive algorithm:**

RECURSION-CHANGE-MAKING(c, k, n)

1. if (n == 0):

2. return 0

3. minimum\_coins = ထ

# Loop trying each coin that has less value than n

4. for i=1 to k

5. if c[i] <= n

6. temp\_minimum\_coins = RECURSION-CHANGE-MAKING(c, k, n-c[i])

# if minimum coins can minimized

7. if temp\_minimum\_coins + 1 < minimum\_coins

8. minimum\_coins = temp\_minimum\_coins + 1

9. return minimum\_coins

**Theoretical analysis of change-making recursive algorithm:**

🡺 This algorithm takes as input an array c[1 . . k] of denominations and integer k which is number of denomination and n which represent total changes in cents and algorithm returns minimum number of coins used. If n = 0, no coins will return, and so RECURSION-CHANGE-MAKING algorithm returns 0 in line 2. Line 3 initializes the minimum coins to ထ , so that for the first time when RECURSION-CHANGE-MAKING return value on line 6 then line 7 will correctly compute.

This algorithm needs to find the minimum number of coins required to make change for **n**, so whichever sub-problem provide the change using the minimum number of coins, we will add 1 to it (because we have selected one coin) and return the value. Here smaller sub-problems will be solved recursively.

Each coin has 2 options whether it selected or not. So,

**Time Complexity = O**(n^k), where k is the number of coins given.

**Space Complexity = O**(n) for the recursive call stack.

**Correctness of output for input (ii) for change-making recursive algorithm:**

Recursive algorithm for total cent 11 gives minimum coin count 2 that means algorithm taking 10 and 1 coins and it return the result very less time because value of n is small.

For 23 it returns minimum coin 5 which is correct because 23 = 10+10+1+1+1. So, it takes two 10’s coins and three 1’s coin and now run time increased to 5626 which is many times more than from 262 which is for input 11.

For test data (ii) where k=5 and n=69 algorithm returns minimum coin 3 which is correct because 69 = 23 +23 +23. So, it takes three 23’s coins and now run time increased to 1344105094 which is many times more than from 4189452 which is for input 51 because k and n both increases here. So, it verifies that time complexity is O(n^k).

**Time graph of change-making greedy algorithm:**

**Pseudocode of change-making greedy algorithm:**

GREEDY-CHANGE-MAKING (c, k, n)

1. deno = c

2. i = k

3. numCoins = 0

4. res = []

4. while( i >= 1):

5. if n/deno[i] >= 1:

6. numCoins = numCoins + int(n/deno[i])

7. res = add coin used

7. n = n - (int(n/deno[i]) \* deno[i])

8. i = i - 1

9. return res

**Time Complexity:** O(k).  
**Auxiliary Space:** O(k).

**Theoretical analysis of change-making greedy algorithm:**

🡺 This algorithm takes as input an array c[1 . . k] of denominations and integer k which is number of denomination and n which represent total changes in cents and algorithm returns minimum number of coins used. If n = 0, 0 coins will return. Greedy algorithms always choose highest possible coin as while loop start from maximum denomination, So it might not give best minimum coin count for other coin system.

**Cost/Timetable: -**

|  |  |  |
| --- | --- | --- |
| **Line no** | **cost** | **times** |
| **1.** | **C1** | **1** |
| **2.** | **C2** | **1** |
| **3.** | **C3** | **1** |
| **4.** | **C4** | **K+1** |
| **5.** | **C5** | **K** |
| **6.** | **C6** | **K** |
| **7.** | **C7** | **K** |
| **8.** | **C8** | **K** |
| **9.** | **C9** | **1** |

**Total cost = sum of all above mentioned cost = C1\*1+C2\*1+C3\*1+C4\*(k+1)+C5\*k+C6\*k+C7\*k+C8\*k+C9\*1**

**Adding all constant together**

**= k + c**

**So, the total cost of above algo is O(k).**

**Correctness of output for input(ii) for greedy algorithm:**

Greedy returns 25, 25, 10, 5, 1, 1, 1, 1 which is equal to 69 but it is not best solution

best solution should be three coins of 23.

The output is not best in case of greedy because it always takes the highest denomination coin first.

**Time graph of bottom-up dynamic programming algorithm:**

**Pseudocode of DP-BOTTOM-UP-CHANGE-MAKING** **Algorithm:**

DP-CHANGE-MAKING (c, k, n)

1. let T be a new array

2. for 1 to n+1

3. T[i] = infinity

4. let R be a new array

5. for 1 to n+1

6. R[i] = -1

7. T[0] = 0

8. for j to k

9. for i to n + 1

10. if i >= c[j]:

11. if T[i - c[j]] + 1 < T[i]:

12. T[i] = 1 + T[i - c[j]]

13. R[i] = j

**Theoretical analysis of change-making bottom-up DP algorithm:**

🡺 This algorithm takes as input an array c[1 . . k] of denominations and integer k which is number of denomination and n which represent total changes in cents and algorithm returns minimum number of coins used. Dynamic programming stores the best result of sub problems and use it. So, it always gives the best result in very less computation time.

**Cost/Timetable: -**

|  |  |  |
| --- | --- | --- |
| **Line no** | **cost** | **times** |
| **1.** | **C1** | **1** |
| **2.** | **C2** | **n+2** |
| **3.** | **C3** | **n+1** |
| **4.** | **C4** | **1** |
| **5.** | **C5** | **n+2** |
| **6.** | **C6** | **n+1** |
| **7.** | **C7** | **1** |
| **8.** | **C8** | **k+1** |
| **9.** | **C9** | **K(n+2)** |
| **10.** | **C10** | **k(n+1)** |
| **11.** | **C11** | **k(n+1)** |
| **12.** | **C12** | **k(n+1)** |
| **13.** | **C13** | **k(n+1)** |

**Total cost =** sum of all above mentioned cost

= C1\*1+C2\* (n+2)+C3\*(n+1)+C4\*1+C5\*(n+2)+C6\*(n+1)+C7\*1+C8\*(k+1)+C9\*( K(n+2)) )+C10\*( K(n+1)) )+C11\*( K(n+1)) )+C12\*( K(n+1)) )+C13\*( K(n+1))

Adding all constant together

**T(n) = kn+n+c**

**So, the total cost of above algo is O(nk).**

**Time Complexity:** O(nk).  
**Auxiliary Space:** O(n).

**Correctness of output for input(ii) for change-making dp algorithm:**

Dynamic programming returns 23, 23, 23 which is equal to 69, So dynamic programming gives minimum coin count in weird coin system as well.

**Comparison of different solution:**

By the theoretical and practical analysis, we have seen that the performance of greedy algorithm is much faster than the recursion but it’s not reliable as it does not work for every currency also the recursive solution will not work for larger value as the complexity is way higher than greedy.

Dynamic programing always gives optimal minimum coins count and executes within few microseconds which is much faster than recursion.

Below is the time comparison graph for greedy and dp algorithms for different inputs:

**Are the theoretical and actual performance results consistent?**

**Recursive Change Making:** For recursive algorithm time taken for weird coin system for 69 should be more than from 73 of US coin system because it has time complexity O(n^k) but in result output 69 has less than time from 73 this may happen because of if condition.

**Greedy Change Making**: Though performance of greedy algo is efficient but the output is not optimal and thus it’s not reliable.

**Dynamic Programming:** This algorithm is reliable and gives as the correct output.

**Analysis and Discussion:**

After analyzing of greedy and recursive algorithms, we can conclude that the change making problem cannot be exactly solved by these two algorithms because of their limitation which are mentioned here: -

·   Greedy is faster than recursion but work for only some coin system like US coin system while in weird coin system it does not give minimum coin count the reason for this is the greedy always goes for maximum number thus sometimes avoid the optimal path.

·      Recursion always give minimum coin count, but the complexity of recursive solution is exponential the time of this algorithm will increase with the size of n and in practical life, we cannot let the user wait for hours to get his change.

·      So, the dynamic programming will be the best solution for coin change because it gives correct output each time and take feasible time which is much less than recursion, although in coin system like US the greedy approach will work faster than DP.