DAA Assigment

22BCE7892

S Sharmeely Khairunnissa
22BCE7892

Q Write a program to solve 0/1 Knapsack problem using the following

a) greedy algorithm b) Dynamic programming algorithm

Aim: To implement the 0/1 Knapsack problem using greedy Algorithm and Dynamic Programming approaches and analyze their efficiency.

Real-time Applications of 0/1 Knapsack problem

1. Resource Allocation in Cloud Computing.
   - Used in scheduling and resource distribution in AWS Azure and GCP.

2. Budget Optimization in Marketing:
   - Companies allocate budgets for advertisements based on the best ROI within constraints.

Algorithm for 0/1 Knapsack Problem

(a) Greedy Algorithm (For Fractional Knapsack Not 0/1)

1. Sort items by their value/weight ratio is descending order.

2. Start picking items with highest value to weight ratio until the capacity is full.

3. If the item cannot fit completely, take the fraction that fits.

(b) Dynamic Programming Approach

1. Create a table dp[n+1][w+1] where n is number of items and w is the total weight capacity.

2. If no items or zero capa

3. If the weight of the item is less than or equal to w, compute:

$$dp[i][w] = \max(val[i-1] + dp[i-1][w-wt[i-1]],\ dp[i-1][w])$$

4. Otherwise, copy the previous row value:

$$dp[i][w] = dp[i-1][w]$$

5. Return $dp[n][W]$ as final result.

Time and Space Complexity

- Greedy Algorithm
  - Time Complexity: $O(n\log n)$
  - Space Complexity: $O(1)$

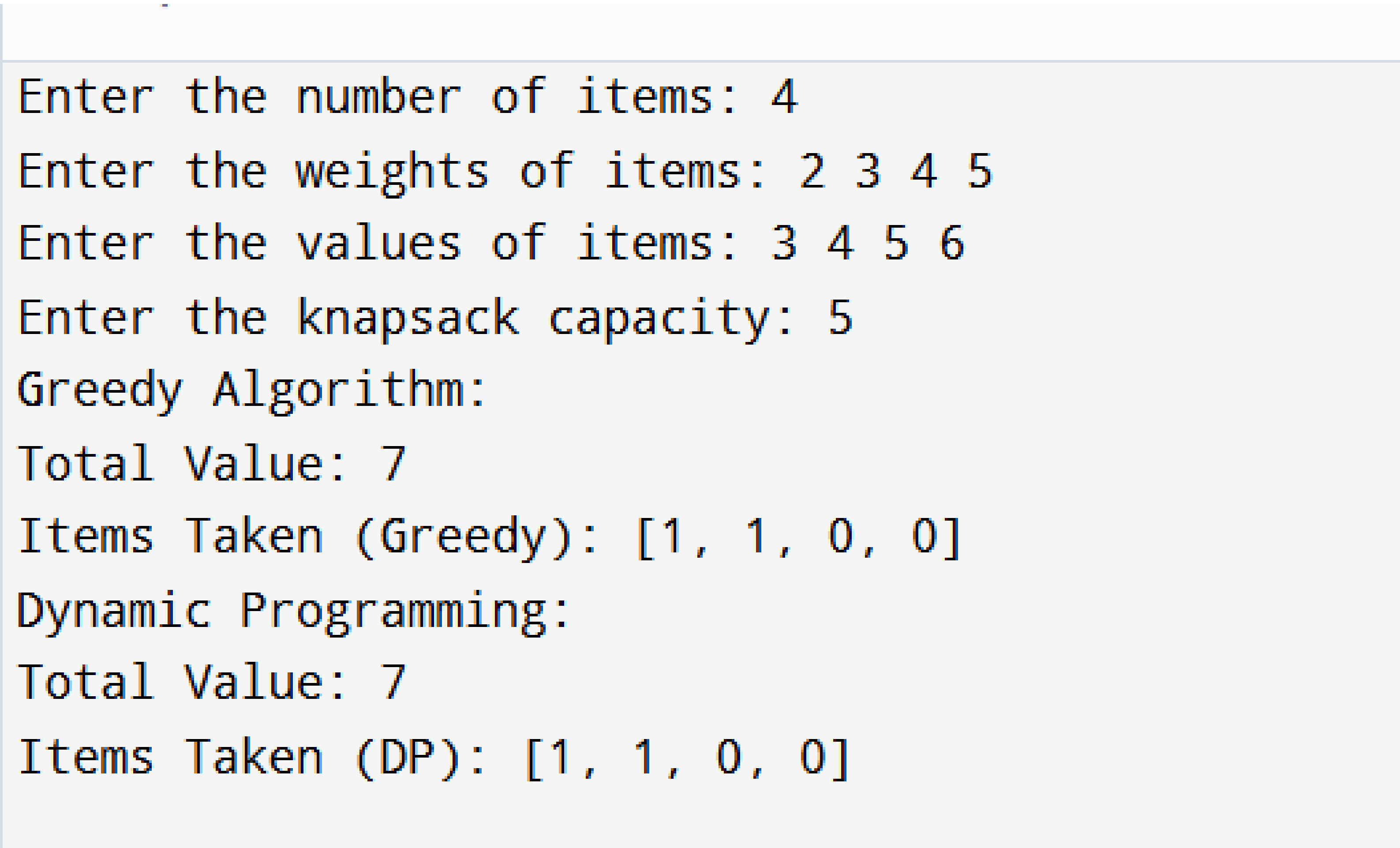
- Dynamic Programming Algorithm
  - Time complexity: $O(nW)$
  - Space Complexity: $O(nW)$

W - is max weight capaci of knapsack

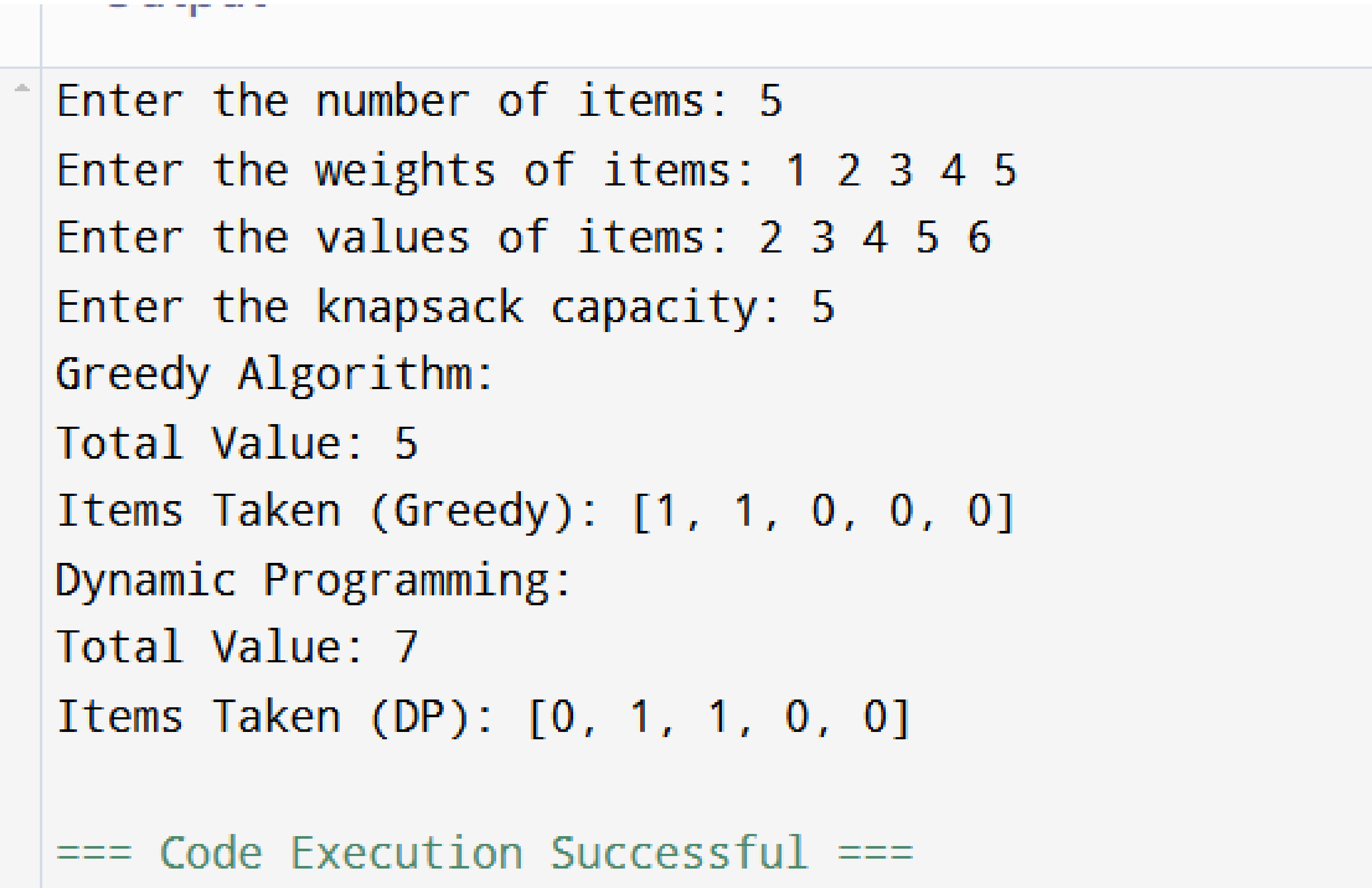
```
import Java.util.*;

class KnapsackSolver
{
  static class Item
   { int weight, value, index;
     public Item (int weight, int value, int index)
      { this.weight = weight;
        this.value = value;
        this.index = index;
      }
   }

  static int[] knapsackGreedy (int[] weights,
                               int[] values,
                               int capacity)

  { int n= weights.length;
    Item[] items = new Item[n];
    for (int i=0; i<n; i++)
      { items[i]= new Item(weights[i], values[i], i);
      }
  Arrays.sort (items, (a,b) -> Double.compare (double)
                        b.value / b.weight, (double)
                             a.value / a.weight))

   int totalValue=0,
   int[] knapsack = new int[n];
```

```
for (Item item :
  {
    if (capacity >= item.weight)
      {  Knap Sack [item.index] = 1;
         totalValue += item.value;
         capacity -= item.weight;

      }
  }
5  System.out.println("Greedy Algo In Total
   Value : " + totalValue);
  return knapSack;
}
Static int []   knapsack DP (int [] weights,
                              int [] values,
                              int capacity)

  {  int n = weights.length;
     int [] [] dp = new int [n+1][capacity+1];

  for (int i=1; i <=n; i++)
   { ~~for( int i=n; i>0;~~
   for ( int w = 0; w <= capacity; w++)
     { if (weight [i-1] <= w)
       { dp [i] [w] = Math.max (dp[i-1] [w],
                          dp [i-1].[w- weight [i-1]]
                          + values [i-1]);

   }
else {
    dp[i][w] = dp[i-1][w];

   }
  }
}
int totalValue = dp[n][capacity];
int [] knapsack = new int [n];
int w= capacity;

for (int i = n; i > 0; i--)
 {
   if (dp[i][w] != dp[i-1][w])
   {
      knapsack [i-1] = 1;
      w -= weigus[i-1];
   }
 }

 System.out.println("Dynamic Programming: \n Total Value: "
        + total Value );
 return knapsack;
}
public static void main (String[] args)
 {
   Scanner in = new Scanner (System.in);
  System.out.print("Enter th no of items: ");
   int n= in.nextInt();
   int [] weights = new int [n];
   int [] values = new int[n];

 System.out.print("Enter the values of items: ");
```

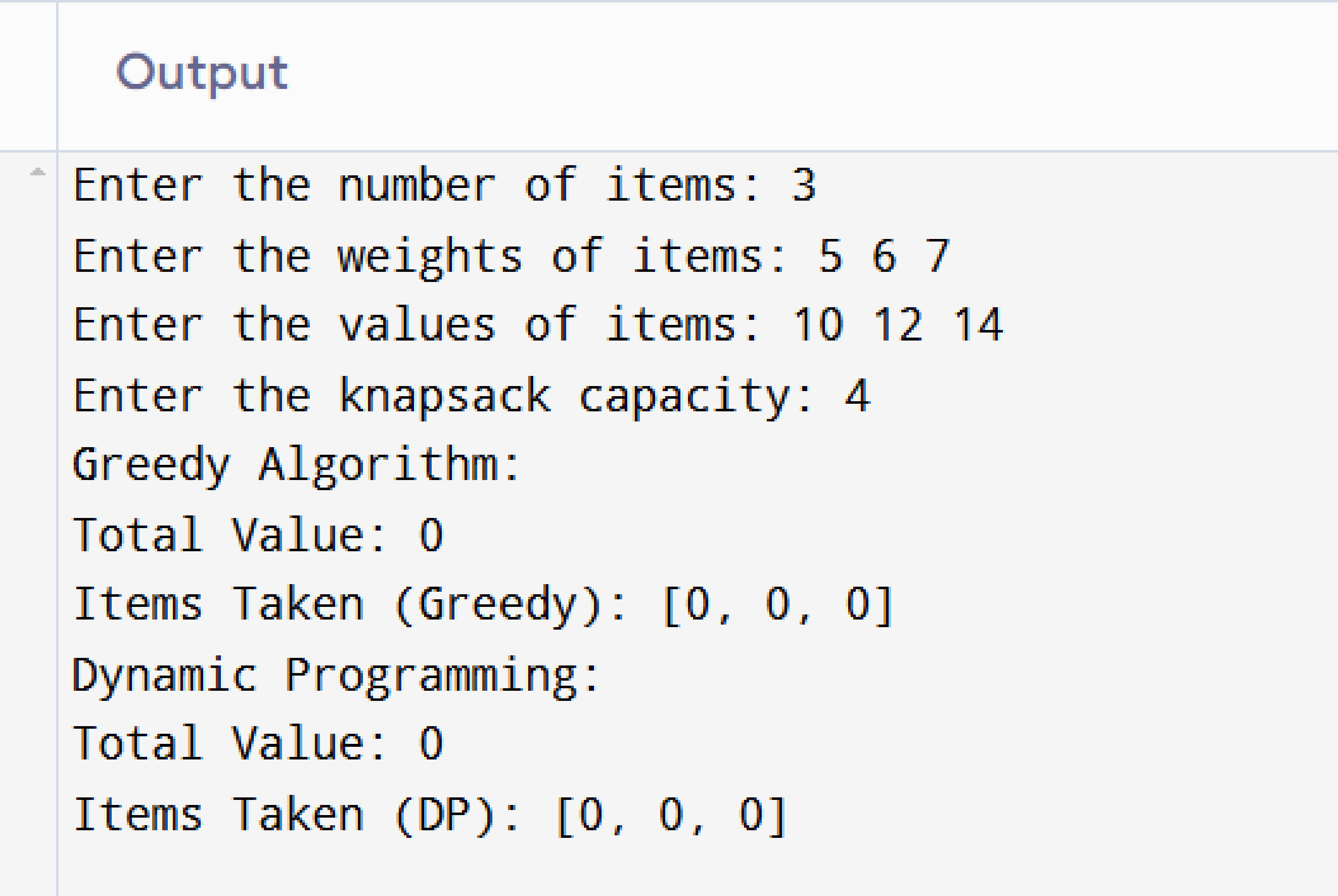
```
    for (int i = 0; i<n; i++)
    { weight ~~values~~[i] = in.next Int ();

    }
   System.out.print ("Enter the values of items :");
   for (int i=0; i<n; i++)
       values[i]= in.nextInt();
  System.out.print ("Enter the knapsack capacity:");
     int capacity = in.nextInt();

 int [] greedy knapsack = knapsack Greedy (weight, values, capaci
 System.out.println ("Items Taken (Greedy): "
                       + Arrays.to String (greedy knapsack));

 int [] dpknapsack = knapsack DP(weight, values, capacity)
 System.out.println ("Items Taken (DP):"+ Arrays.to String
                       (dpknapsack));

    in.close();
  }
}
```



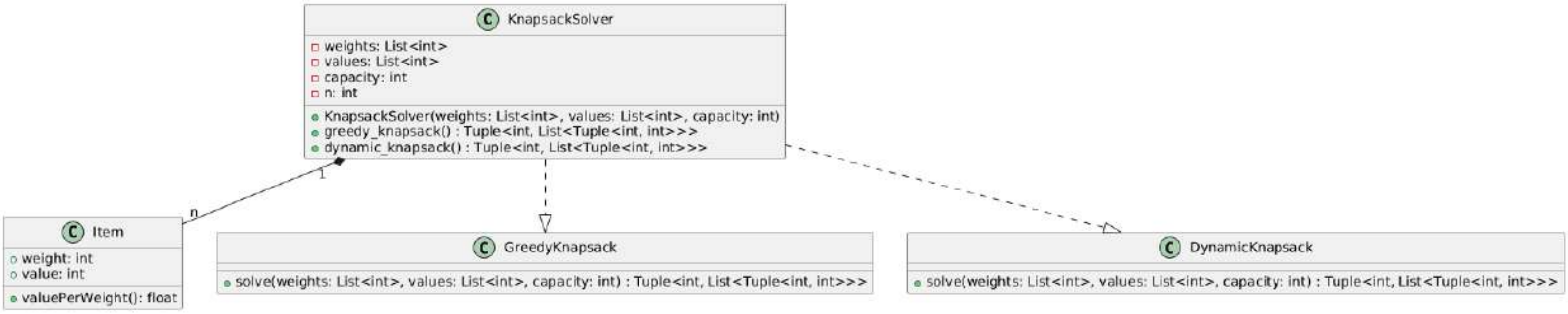
```
Enter the number of items: 4
Enter the weights of items: 2 3 4 5
Enter the values of items: 3 4 5 6
Enter the knapsack capacity: 5
Greedy Algorithm:
Total Value: 7
Items Taken (Greedy): [1, 1, 0, 0]
Dynamic Programming:
Total Value: 7
Items Taken (DP): [1, 1, 0, 0]
```



```
Enter the number of items: 5
Enter the weights of items: 1 2 3 4 5
Enter the values of items: 2 3 4 5 6
Enter the knapsack capacity: 5
Greedy Algorithm:
Total Value: 5
Items Taken (Greedy): [1, 1, 0, 0, 0]
Dynamic Programming:
Total Value: 7
Items Taken (DP): [0, 1, 1, 0, 0]


=== Code Execution Successful ===
```



```
Enter the number of items: 3
Enter the weights of items: 5 6 7
Enter the values of items: 10 12 14
Enter the knapsack capacity: 4
Greedy Algorithm:
Total Value: 0
Items Taken (Greedy): [0, 0, 0]
Dynamic Programming:
Total Value: 0
Items Taken (DP): [0, 0, 0]
```


KnapsackSolver
weights: List<int>
values: List<int>
capacity: int
n: int
KnapsackSolver(weights: List<int>, values: List<int>, capacity: int)
greedy_knapsack() : Tuple<int, List<Tuple<int, int>>>
dynamic_knapsack() : Tuple<int, List<Tuple<int, int>>>
1
n
Item
weight: int
value: int
valuePerWeight(): float
GreedyKnapsack
solve(weights: List<int>, values: List<int>, capacity: int) : Tuple<int, List<Tuple<int, int>>>
DynamicKnapsack
solve(weights: List<int>, values: List<int>, capacity: int) : Tuple<int, List<Tuple<int, int>>>