```java
package DynamicProgramming;

public class KnapsackRecur {

    public static void main(String[] args) {
        int val[] = new int[] { 2,3,1,4 };
        int wt[] = new int[] { 4,5,3,7 };
        int W = 5;
        int n = val.length;
       System.out.println("Max Profit using recursion = "+knapSackRec(W, wt, val, n));
       System.out.println(knapSackDP(W, wt, val, n));
    }

    private static int knapSackMem(int w, int[] wt, int[] val, int n,int[][]dp) {
        if(w==0 || n==0) {
            return 0;
        }
        if (dp[n][w] != -1)
            return dp[n][w];

        if(wt[n-1]>w) {
            return dp[n][w] = knapSackMem(w, wt, val, n-1,dp);
        }else {
            return dp[n][w] = Math.max(val[n-1] + knapSackMem(w-wt[n-1], wt, val, n-1,dp),knapSackMem(w, wt, val, n-1,dp));
        }

    }
```

```java

    static void knapSack(int W, int wt[], int val[], int N)
    {

        // Declare the table dynamically
        int dp[][] = new int[N + 1][W + 1];7

        // Loop to initially filled the
        // table with -1
        for(int i = 0; i < N + 1; i++)
            for(int j = 0; j < W + 1; j++)
                dp[i][j] = -1;


        System.out.println("Maximum Profit using Memoization = "+knapSackMem(W, wt, val, N, dp));
        for(int i = 0; i < N + 1; i++) {
            for(int j = 0; j < W + 1; j++) {
                    System.out.print(dp[i][j]+" ");
                }
            System.out.println();
        }

```