Name: Vanessa D'mello
Roll No: 8863
Branch: SE Computers A (Batch A)
Experiment: 0/1 Knapsack

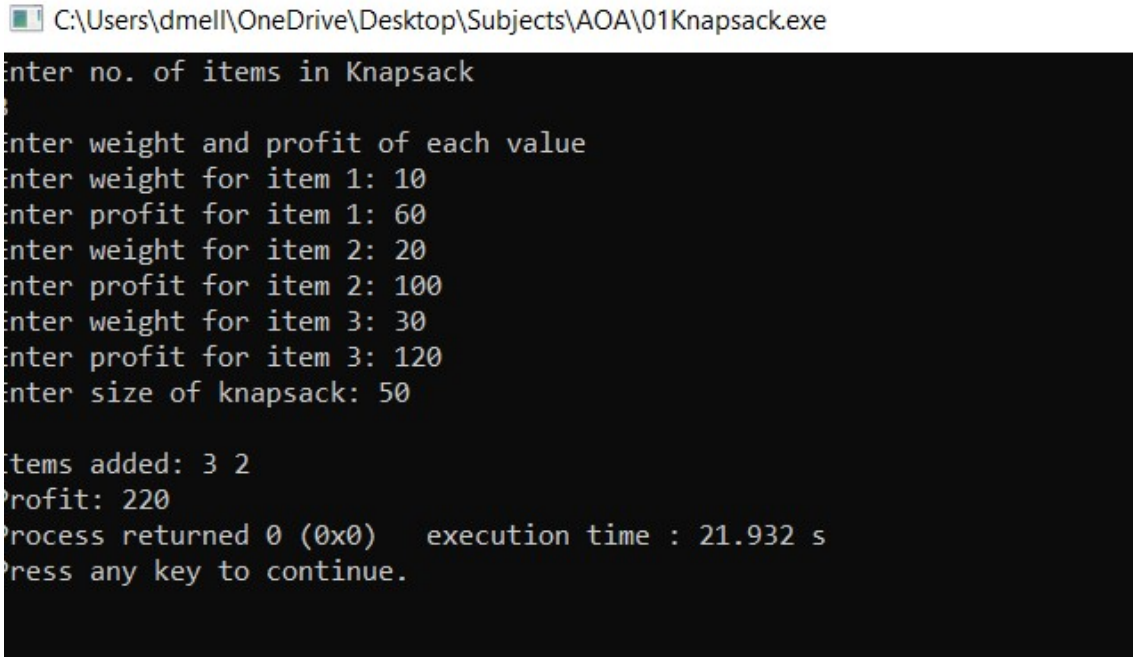
```
#include <stdio.h>
int max(int i, int j) {
	return i > j ? i : j;
}
int knapsack(int w[], int p[], int n, int M) {
	int v[n + 1][M + 1];
	for (int i = 0; i <= n; i++) {
		for(int j = 0; j <= M; j++) {
			if(i == 0 || j == 0)
			v[i][j] = 0;
			else {
				if(w[i - 1] > j)
					v[i][j] = v[i - 1][j];
				else
					v[i][j] = max(v[i - 1][j], p[i - 1] + v[i - 1][j - w[i - 1]]);
			}
		}
	}

	int i, j, totalProfit = 0;
	i = n;
	j = M;
	printf("\nItems added: ");
	while(i > 0 && j > 0) {
		if(v[i][j] != v[i - 1][j]) {
			printf("%d ", i);
			totalProfit += p[i - 1];
			j = j - w[i - 1];
		}
		i--;
	}
	return totalProfit;
}

int main() {
	int n;
	printf("Enter no. of items in Knapsack\n");
	scanf("%d", &n);
	int w[n];
	int p[n];
	printf("Enter weight and profit of each value\n");
	for(int i = 0; i < n; i++) {
```

```
            printf("Enter weight for item %d: ", i + 1);
            scanf("%d", &w[i]);
            printf("Enter profit for item %d: ", i + 1);
            scanf("%d", &p[i]);
        }
        int M;
        printf("Enter size of knapsack: ");
        scanf("%d", &M);
        int profit = knapsack(w, p, n, M);
        printf("\nProfit: %d", profit);
        return 0;
}
```

Output:



Name: Vanessa D'mello
Roll No.: 8863
Experiment: 0/1 Knapsack

Postlab

Greedy Approach	Dynamic Programming Approach
① In greedy Algorithm, we make whatever choice seems best at the moment in hope that it will lead to optimal solution	② In this approach, we make a decision at each step considering the current problem and solution to previously solved subproblem to calculate optimal solution.
② There is no ~~guar~~ guarantee of getting Optimal Solution	② There is a guarantee of generating optimal Solution
③ It is more efficient in terms of memory as it doesn't store the previously generated values	③ It requires a table for memoization and it increase its space complexity
④ The greedy method computes its solution by making its choices in a serial forward fashion	④ Dynamic Programming computes its solution bottom-up or top-down by analysing them from smaller optimal sub solution
eg Fractional Knapsack	0/1 Knapsack.