NAME:	Yash Ravindra Burad
UID:	2022301004
SUBJECT	DAA
EXPERIMENT NO :	05
THEORY	The Greedy algorithm could be understood very well with a well-known problem referred to as Knapsack problem. Although the same problem could be solved by employing other algorithmic approaches, Greedy approach solves Fractional Knapsack problem reasonably in a good time. Let us discuss the Knapsack problem in detail. The basic idea of the greedy approach is to calculate the ratio **profit/weight** for each item and sort the item on the basis of this ratio. Then take the item with the highest ratio and add them as much as we can (can be the whole element or a fraction of it). This will always give the maximum profit because, in each step it adds an element such that this is the maximum possible profit for that much weight. The fractional knapsack problem is also one of the techniques which are used to solve the knapsack problem. In fractional knapsack, the items are broken in order to maximize the profit. The problem in which we break the item is known as a Fractional knapsack problem. Example: **Input:** arr[] = {{60, 10}, {100, 20}, {120, 30}}, W = 50 **Output:** 240 **Explanation:** By taking items of weight 10 and 20 kg and 2/3 fraction of 30 kg. Hence total price will be 60+100+(2/3)(120) = 240 **Input:** arr[] = {{500, 30}}, W = 10 **Output:** 166.667

* Fractional knapsack problem :-

Item : 1, 2, 3, 4, 5, 6, 7
Profit (P) : 10, 15, 7, 8, 9, 4, 5
Weight (w) : 1, 3, 5, 4, 1, 3, 2.

W (weight of knapsack) : 15.
n (number of items) : 7.

→

Item	profits (p)	weights (w)
1	10	1
2	15	3
3	7	5
4	8	4
5	9	1
6	4	3
7	5	2

(I) calculate, profit / weight for all items.

Item	profits	weights	Profit / weight
1	10	1	10
2	15	3	5
3	7	5	1.4
4	8	4	2
5	9	1	9
6	4	3	1.3
7	5	2	2.5

(II) Arrange the items in nonascending order of profit / weight. (i.e., descending order)

Item	profits	weights	Profit / weight (↓ Desc.)
1	10	1	10
5	9	1	9
2	15	3	5
7	5	2	2.5
4	8	4	2
3	7	5	1.4
6	4	3	1.3

(III) Select the items based on above table, where, W of Knapsack is 15.

Item selected	Profit	Weight	Remaining weight
1	10	1	15-1 = 14
5	9	1	14-1 = 13
2	15	3	13-3 = 10
7	5	2	10-2 = 8
4	8	4	8-4 = 4
3	7	5	4-5 = -1
			... (Knapsack overload)

(a) Create the fractions of items.

In above example,

weight = 4, is remaining in the bag.

∴ w(4) will be added in bag and fractional profit will be, $\frac{4}{5} \times 7 = 5.6$.

Item	p	w	remaining weight
∴ 3	5.6	4	4-4 = 0
	$\Sigma p = 59.6$	$\Sigma w = 15$	

[∴ profit in our knapsack = 59.6] //

ALGORITHM	**Algorithm: Greedy-Fractional-Knapsack (w[1..n], p[1..n], W)** for i = 1 to n do x[i] = 0 weight = 0 for i = 1 to n

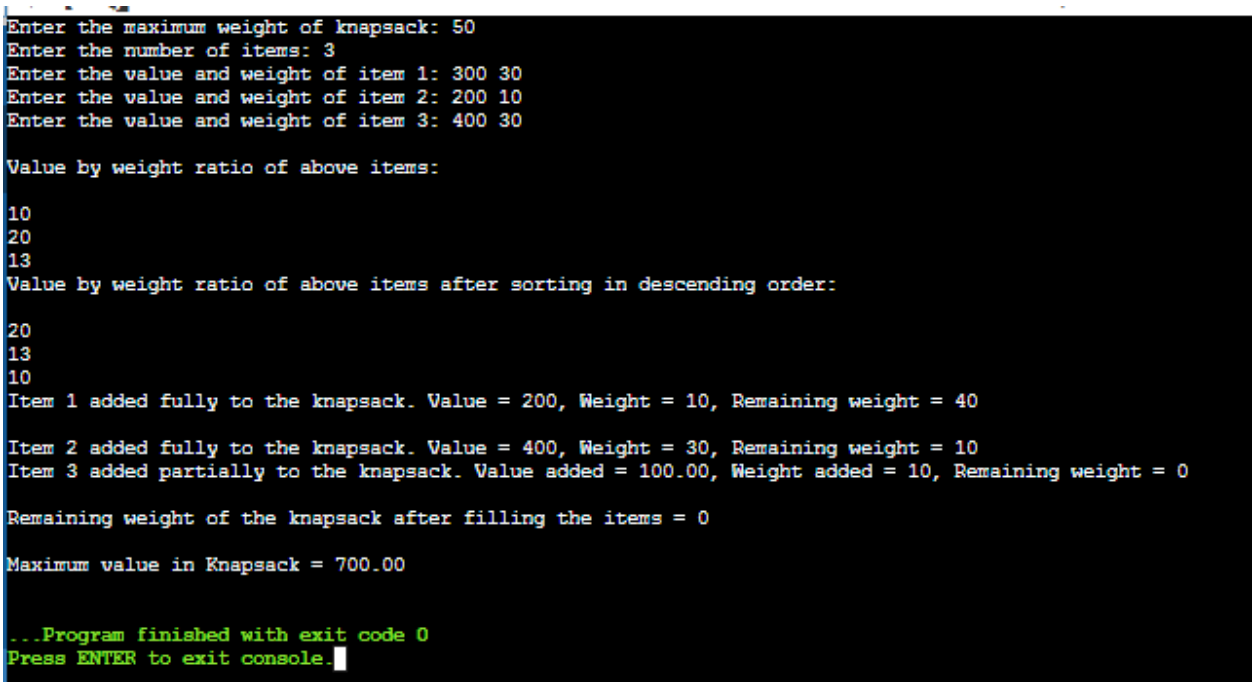
	if weight + w[i] ≤ W then
	x[i] = 1
	weight = weight + w[i]
	else
	x[i] = (W - weight) / w[i]
	weight = W
	break
	return x

PROGRAM:	#include <stdio.h>
	#include <stdlib.h>
	
	struct Item {
	int value;
	int weight;
	};
	
	int compare(const void *a, const void *b) {
	double r1 = (double)(((struct Item*)a)->value) / (double)(((struct Item*)a)->weight);
	double r2 = (double)(((struct Item*)b)->value) / (double)(((struct Item*)b)->weight);
	if (r1 < r2) {
	return 1;
	} else if (r1 > r2) {
	return -1;
	} else {
	return 0;
	}
	}
	
	double fractionalKnapsack(int W, struct Item arr[], int n) {
	printf ("\nValue by weight ratio of above items:\n");
	for (int i = 0; i < n; i++) {

```
        int valueByWeight = arr[i].value / arr[i].weight;
        printf ("\n%d",valueByWeight);

     }
    qsort(arr, n, sizeof(struct Item), compare);
    int curWeight = 0;
    double finalValue = 0.0;
     printf ("\nValue by weight ratio of above items after sorting in
descending order:\n");
     for (int i = 0; i < n; i++) {
        int valueByWeight = arr[i].value / arr[i].weight;
        printf ("\n%d",valueByWeight);

     }
    for (int i = 0; i < n; i++) {
        if (curWeight + arr[i].weight <= W) {
            curWeight += arr[i].weight;
            finalValue += arr[i].value;
            printf("\nItem %d added fully to the knapsack. Value = %d,
Weight = %d, Remaining weight = %d\n", i+1, arr[i].value,
arr[i].weight, W - curWeight);
        } else {
            int remain = W - curWeight;
            finalValue += arr[i].value * ((double) remain / (double)
arr[i].weight);
            printf("Item %d added partially to the knapsack. Value
added = %.2f, Weight added = %d, Remaining weight = 0 \n", i+1,
arr[i].value * ((double) remain / (double) arr[i].weight), remain);
            break;
        }
    }
   // int remainingWeight = W - curWeight;
    printf("\nRemaining weight of the knapsack after filling the
items = 0 \n");
```

```
    return finalValue;
}

int main() {
    int W, n;
    printf("Enter the maximum weight of knapsack: ");
    scanf("%d", &W);
    printf("Enter the number of items: ");
    scanf("%d", &n);

    struct Item arr[n];
    for (int i = 0; i < n; i++) {
        printf("Enter the value and weight of item %d: ", i+1);
        scanf("%d %d", &arr[i].value, &arr[i].weight);
    }

    double maxVal = fractionalKnapsack(W, arr, n);

    printf("\nMaximum value in Knapsack = %.2f\n", maxVal);

    return 0;
}
```

Output:



INFERENCE	Time complexity of Fractional knapsack problem: The time complexity of the above program is O(n log n), where n is the number of items in the knapsack. This is because the program uses the qsort function to sort the array of items by density, and the time complexity of qsort is O(n log n). The program also has a for loop that iterates over the items in the sorted array, which has a time complexity of O(n). Therefore, the overall time complexity of the program is O(n log n) + O(n), which simplifies to O(n log n). Motive to solve this problem: Maximize the profits with minimum weight in knapsack.
CONCLUSION:	In this practical, I leant about fractional knapsack problem using greedy approach, where I learnt that while putting the items into the knapsack, it is important to put maximum profit/value and less weight.