K. J. Somaiya College of Engineering, Mumbai-77
(A Constituent College of Somaiya Vidyavihar University)
Department of Computer Engineering

Batch:-B-2	**Roll No:-** 16010122151
Experiment No.5	
Grade: AA / AB / BB / BC / CC / CD /DD	
Signature of the Staff In-charge with date	

Title: Implementation of Knapsack Problem using Greedy strategy

---

Objective: To learn the Greedy strategy of solving the problems for different types of problems

---

CO to be achieved:

CO 2 Describe various algorithm design strategies to solve different problems and analyse Complexity.

Books/ Journals/ Websites referred:

1. **Ellis horowitz, Sarataj Sahni, S.Rajsekaran," Fundamentals of computer algorithm", University Press**
2. **T.H.Cormen ,C.E.Leiserson,R.L.Rivest and C.Stein," Introduction to algortihtms",2nd Edition ,MIT press/McGraw Hill,2001**
3. **http://lcm.csa.iisc.ernet.in/dsa/node184.htm**
4. **http://students.ceid.upatras.gr/~papagel/project/kruskal.htm**
5. **http://www.personal.kent.edu/~rmuhamma/Algorithms/MyAlgorithms/GraphAlgor/kruskalAlgor.html**
6. **http://lcm.csa.iisc.ernet.in/dsa/node183.html**
7. **http://students.ceid.upatras.gr/~papagel/project/prim.htm**
8. **http://www.cse.ust.hk/~dekai/271/notes/L07/L07.pdf**

Pre Lab/ Prior Concepts:
Data structures, Concepts of algorithm analysis

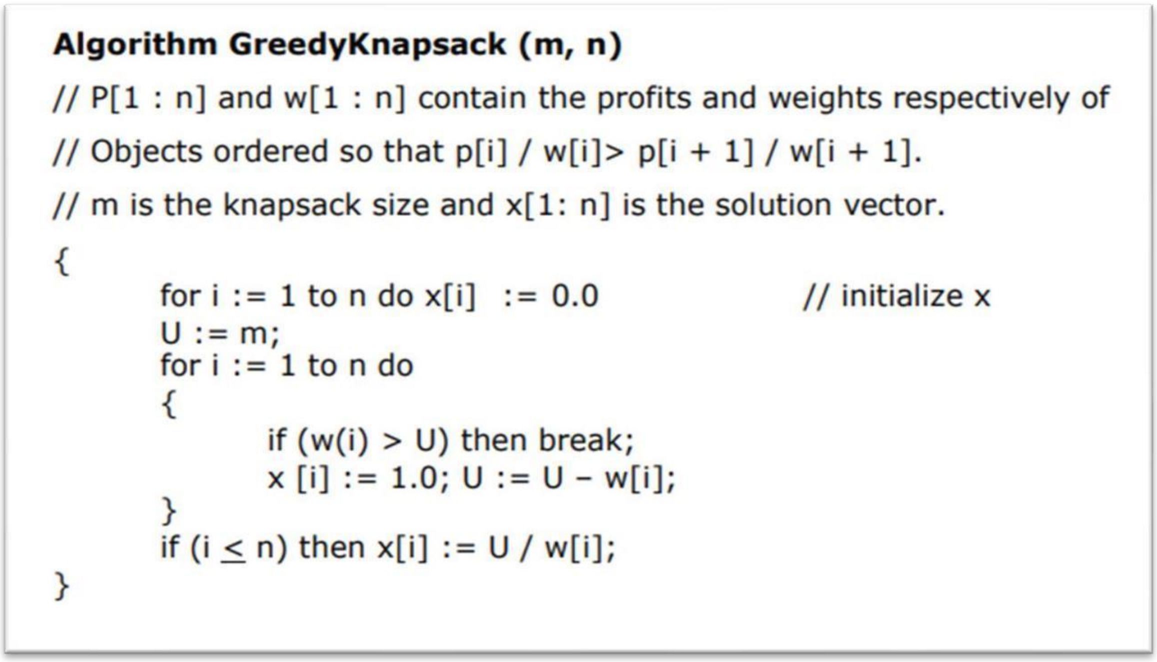
Historical Profile:

The knapsack problem represents constraint satisfaction optimization problems’ family. Based on nature of constraints, the knapsack problem can be solved with various problem saolving strategies. Typically, these problems represent resource optimization solution.

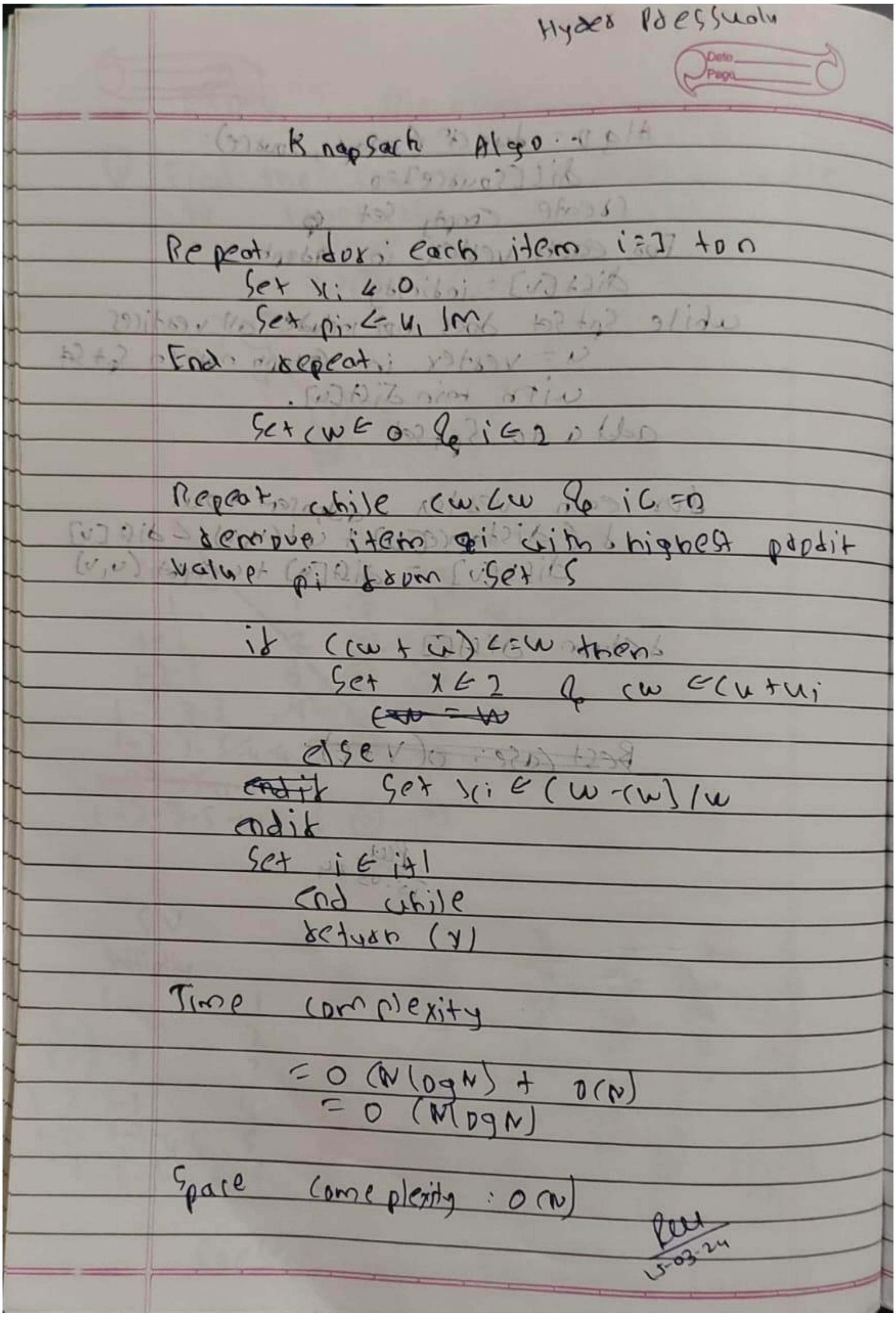
Given a set of n inputs. · Find a subset, called feasible solution, of the n inputs subject to some constraints, and satisfying a given objective function. · If the objective function is maximized or minimized, the feasible solution is optimal. · It is a locally optimal method.

New Concepts to be learned:
Application of algorithmic design strategy to any problem, Greedy method of problem solving Vs other methods of problem solving, optimality of the solution, knapsack problem and their applications

Knapsack Problem Algorithm



```
Algorithm GreedyKnapsack (m, n)
// P[1 : n] and w[1 : n] contain the profits and weights respectively of
// Objects ordered so that p[i] / w[i]> p[i + 1] / w[i + 1].
// m is the knapsack size and x[1: n] is the solution vector.
{
        for i := 1 to n do x[i]  := 0.0                // initialize x
        U := m;
        for i := 1 to n do
        {
                if (w(i) > U) then break;
                x [i] := 1.0; U := U – w[i];
        }
        if (i ≤ n) then x[i] := U / w[i];
}
```



Hyder Pressuala

Date
Page

Knapsack Algo

```
Repeat for each item i=1 to n
    Set x_i ← 0
    Set p_i ← v_i / w_i
End repeat

    Set cw ← 0 & i ← 1

Repeat while cw < w & i <= n
remove item qi with highest profit value pi from set S

    if (cw + wi) <= w then
            Set x ← 1 & cw ← cw + wi
              ~~cw = w~~
        else
    ~~endif~~   Set xi ← (w - cw)/w
    endif
    Set i ← i+1
        End while
        return (x)
```

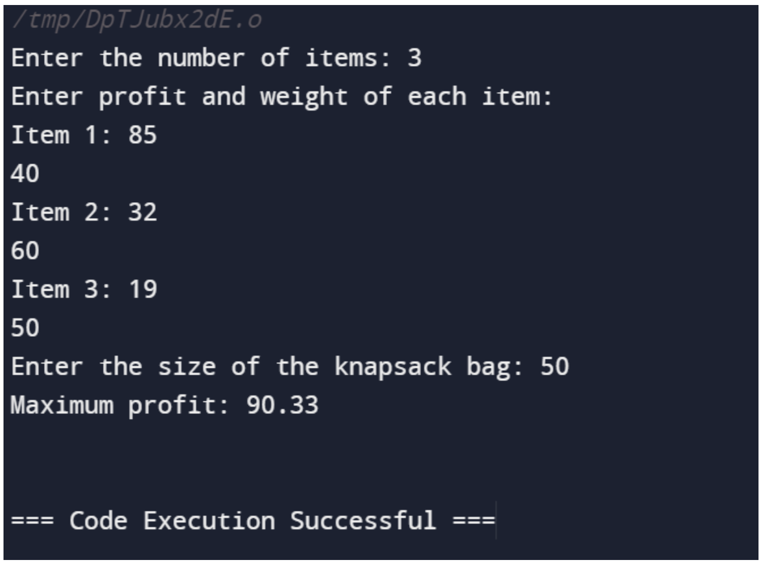
Time complexity

$= O(N \log N) + O(N)$
$= O(N \log N)$

Space complexity : $O(N)$

[illegible]
15-03-24

```c
#include <stdio.h>
#include <stdlib.h>

struct Item {
    int profit;
    int weight;
};

int compare(const void *a, const void *b) {
    double ratio_a = (double)(((struct Item *)a)->profit) / (((struct Item *)a)-
>weight);
    double ratio_b = (double)(((struct Item *)b)->profit) / (((struct Item *)b)-
>weight);
    if (ratio_a > ratio_b) return -1;
    else if (ratio_a < ratio_b) return 1;
    return 0;
}

double fractionalKnapsack(int W, struct Item arr[], int n) {
    qsort(arr, n, sizeof(struct Item), compare);

    double finalvalue = 0.0;

    for (int i = 0; i < n; i++) {
        if (arr[i].weight <= W) {
            W -= arr[i].weight;
            finalvalue += arr[i].profit;
        } else {
            finalvalue += (double)arr[i].profit * W / arr[i].weight;
            break;
        }
    }

    return finalvalue;
}

void inputItems(struct Item arr[], int n) {
    printf("Enter profit and weight of each item:\n");
    for (int i = 0; i < n; i++) {
        printf("Item %d: ", i + 1);
        scanf("%d %d", fiarr[i].profit, fiarr[i].weight);
    }
}

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

    struct Item *arr = (struct Item *)malloc(n * sizeof(struct Item));
    if (arr ==  NULL) {
        printf("Memory allocation failed!\n");
        return 1;
    }

    inputItems(arr, n);

    printf("Enter the size of the knapsack bag: ");
    scanf("%d", fiW);

    double max_val = fractionalKnapsack(W, arr, n);
    printf("Maximum profit: %.2f\n", max_val);

    free(arr);
    return 0;
}
```



```
/tmp/DpTJubx2dE.o
Enter the number of items: 3
Enter profit and weight of each item:
Item 1: 85
40
Item 2: 32
60
Item 3: 19
50
Enter the size of the knapsack bag: 50
Maximum profit: 90.33


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

Conclusion:- Successfully implemented Knapsack problem.