
| **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:**

| CO2 | 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**](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:**

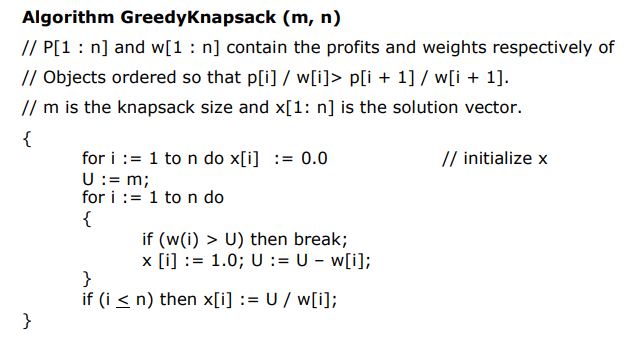
The knapsack problem represents constraint satisfaction optimization problems’ family. Based on nature of constraints, the knapsack problem can be solved with various problem saolving strategies. Typically, these problems represent resource optimization solution.

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**

****

**Code:**

import java.lang.\*;

import java.util.Arrays;

import java.util.Comparator;

import java.util.Scanner;

// Greedy approach

public class FractionalKnapSack {

// Function to get maximum value

private static double getMaxValue(ItemValue[] arr, int capacity) {

// Sorting items by profit/weight ratio;

Arrays.sort(arr, new Comparator<ItemValue>() {

@Override

public int compare(ItemValue item1, ItemValue item2) {

double cpr1 = (double) item1.profit / item1.weight;

double cpr2 = (double) item2.profit / item2.weight;

if (cpr1 < cpr2)

return 1;

else

return -1;

}

});

double totalValue = 0d;

for (ItemValue i : arr) {

int curWt = i.weight;

int curVal = i.profit;

if (capacity - curWt >= 0) {

// This weight can be picked whole

capacity = capacity - curWt;

totalValue += curVal;

} else {

// Item cant be picked whole

double fraction = (double) capacity / curWt;

totalValue += (curVal \* fraction);

capacity = (int) (capacity - (curWt \* fraction));

break;

}

}

return totalValue;

}

// Item value class

static class ItemValue {

int profit, weight;

// Item value function

public ItemValue(int val, int wt) {

this.weight = wt;

this.profit = val;

}

}

// Driver code

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.println("Enter the number of items: ");

int n = scanner.nextInt();

ItemValue[] arr = new ItemValue[n];

for (int i = 0; i < n; i++) {

System.out.println("Enter profit and weight for item " + (i + 1) + ": ");

int profit = scanner.nextInt();

int weight = scanner.nextInt();

arr[i] = new ItemValue(profit, weight);

}

System.out.println("Enter the capacity of the knapsack: ");

int capacity = scanner.nextInt();

double maxValue = getMaxValue(arr, capacity);

// Function call

System.out.println("Maximum value we can obtain = " + maxValue);

}

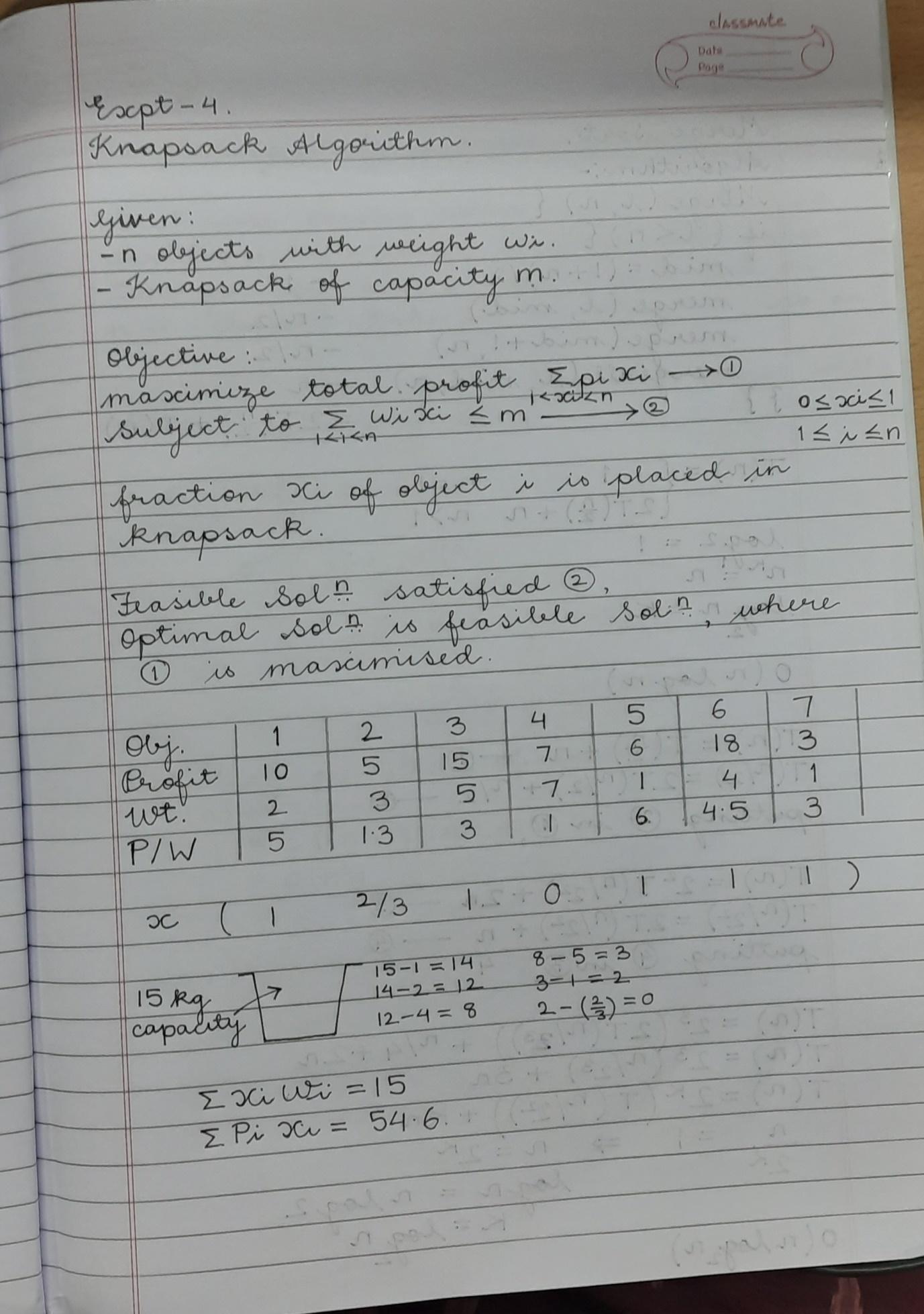
}

**Time Complexity:**

* **Sorting of the array takes O(nlogn) time if we consider a merge or a quick sort**
* **Greedy Knapsack takes O(n) time**
* **Therefore, total time complexity is O(n + nlogn) = O(nlogn) Space Complexity:**
* **O(3n + 2) = O(n) as we only traverse through the arrays.**

**Example: Knapsack Problem**

**Analysis of Knapsack   Problem algorithm:**



**Conclusion:**

Through this experiment we learnt about the fractional knapsack problem and learnt to implement it in Java.