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];
}
```

Analysis of Knapsack Problem algorithm:

Time complexity- only for sorting so O(n log(n))

Example: Knapsack Problem

1601012111 0 B2

fractional knapsack

Place values with highest profit, ratio or lowest weight as per the strategy
- If unable to place, put fractional part till weight is full & stop
- Repeat step 1 till all is exausted or knapsack full

e complexity : O(n log n) Sorting

Example

Weight	10	20	30
Profit	60	100	120

Capacity : 50

Maximize profit

30 20
120 100 220

Minimize Weight

10 20 $\frac{2}{3}$ 30
60 100 $\frac{2}{3}$ x120 = 240

ximize ratio

10	20	30
60	100	120
6.0	5.0	4.0

```
import java.util.*;
public class Main
{
	public static void main(String[] args) {
		double [] profit ={1,2,4,3,5};
		double [] weight = {2,1,4,5,7};
		double capacity=7;
		double[][] ratio = new double[5][5];
		for(int i=profit.length-1;i>=0;i--){

		  ratio[i][0]=profit[i]/weight[i];
		  ratio[i][1]=weight[i];
		}

		//sort ArrayList


		ratio=insetionSort(ratio);



		int i=0;
		while(capacity>0){

		   capacity=capacity-ratio[i][1];
		   if(capacity>0){
		   System.out.println("chosen object with weight "+ratio[i][1]);

		   }
		  i++;
		   if(i==5){//on capacity less than objects
		      break;
		   }
		}
	}
	public static double[][]                    insetionSort(double[][] arr1){
	   for(int i=1;i<arr1.length;i++){
	      double key=arr1[i][0];
	      int j=i-1;
	      while(arr1[j][0]<key){
	         //swap ratio
	         double temp=arr1[j+1][1];
	         arr1[j+1][1]=arr1[j][1];
	         arr1[j][1]=temp;
	         //swap weight
	          temp=arr1[j+1][0];
```

```
                arr1[j+1][0]=arr1[j][0];
                arr1[j][0]=temp;

                j--;
                if(j<0){
                   break;
                }
              }
            }
        //   for(int i=0;i<arr1.length;i++){
    //     System.out.print(arr1[i]+",");
    //   }
           return arr1;

        }

}
```

Output-

chosen object with weight 1.0
chosen object with weight 4.0

Conclusion:
Thus we have understood how to implement fractional knapsack using greedy strategy. We used the p/w ratio to implement the best possible profit. This has many applicaiotns including thread scheduling policies