**Batch: A1 Roll No.: 1911004**

**Experiment No. 4a**

**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:**

|  |  |
| --- | --- |
| Sr. No | Objective |
| CO 1 | Analyze the asymptotic running time and space complexity of algorithms. |
| CO 2 | Describe various algorithm design strategies to solve different problems and analyze  Complexity. |
| CO 3 | Develop string matching techniques |
| CO 4 | Describe the classes P, NP, and NP-Complete |

**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:**

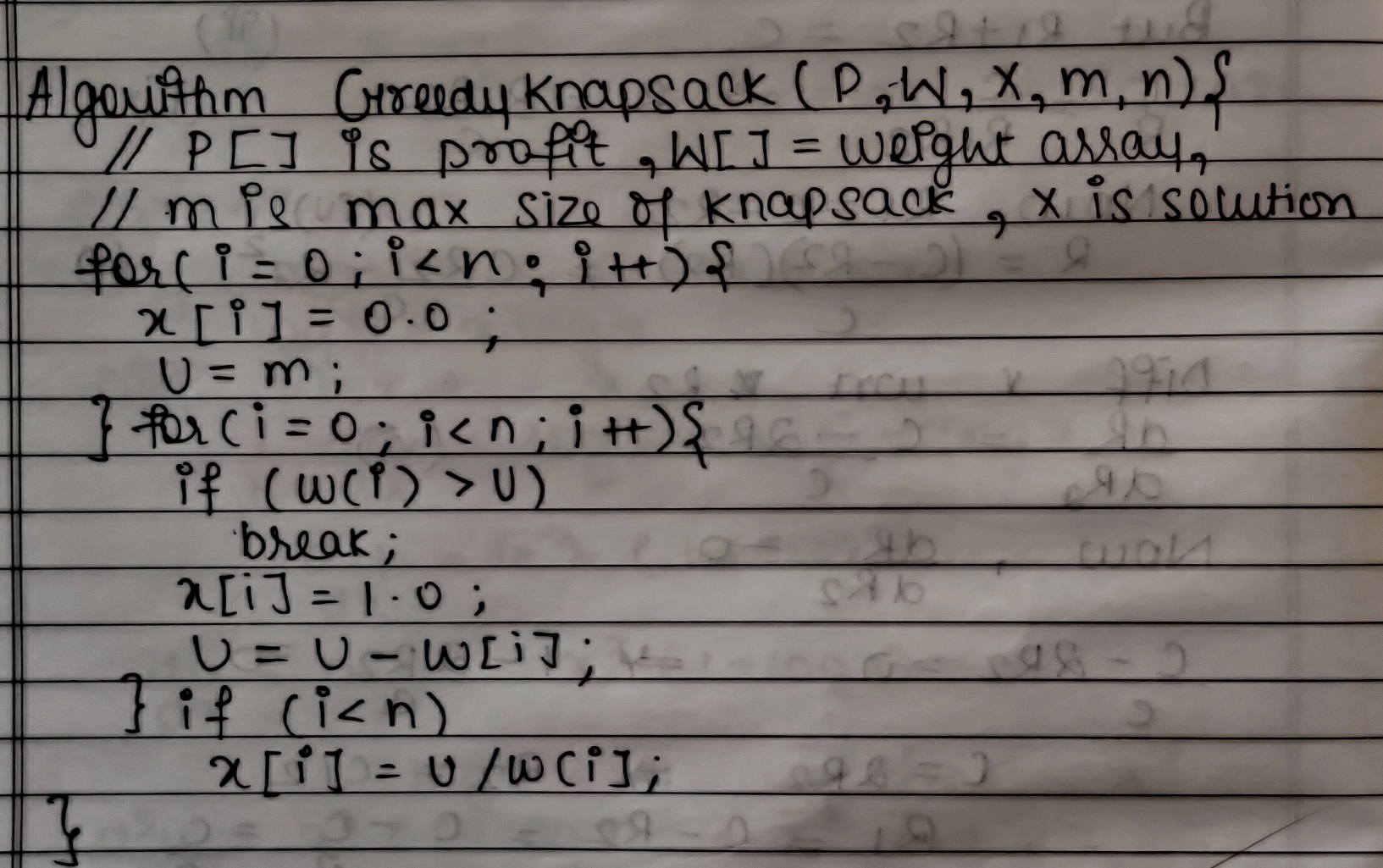
The knapsack problem represents constraint satisfaction optimization problems’ family. Based on nature of constraints, the knapsack problem can be solved with various problem solving strategies. Typically, these problems represent resource optimization solution.

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**



**Implementation**

**Code**

import java.util.\*;

class Main {

public static void main(String[] args) {

System.out.println("Fractional KnapSack");

Scanner ob=new Scanner(System.in);

System.out.print("Enter no of Objects ");

int i,j,p,w,n = ob.nextInt();

OBJ obj[]=new OBJ[n];

OBJ maxPr[] = new OBJ[n]; OBJ minWt[] = new OBJ[n]; OBJ maxPW[] = new OBJ[n];

for(i=0;i<obj.length;i++){

System.out.print("Enter Profit,Weight for Object "+(i+1)+" ");

p = ob.nextInt();

w = ob.nextInt();

obj[i]=new OBJ((i+1),p,w,p/w);

System.out.println("");

}

System.out.print("Enter MAX Capacity ");

double cap = ob.nextDouble();

maxPr = obj; minWt = obj; maxPW = obj;

System.out.println("Max Profit");

mergeSort(maxPr,0,n-1,'p'); KnapSack(maxPr,cap,n,1);

System.out.println("Min Weight");

mergeSort(minWt,0,n-1,'w'); KnapSack(minWt,cap,n,0);

System.out.println("Max Profit/Weight");

mergeSort(maxPW,0,n-1,'r'); KnapSack(maxPW,cap,n,1);

}

public static void mergeSort(OBJ A[],int l,int u,char C){

int m,R=1;

if (l<u){

m = (l+u)/2;

mergeSort(A,l,m,C);

mergeSort(A,m+1,u,C);

A = merge(A,l,m,u,C);

}

}

public static void KnapSack(OBJ A[],double cap,int n,int flag){

double remW=cap,tot=0,p\_,r\_,w\_;

int i;

System.out.print("Object\t Profit\t Weight\t Ratio\t Remaining Weight\n");

if(remW>=0.0 && flag==1){

   for(i=n-1;i>=0;i--){

   r\_=A[i].p/A[i].w;

   w\_=A[i].w;

p\_=A[i].p;

if(A[i].w>remW){

w\_=remW/A[i].w;

p\_\*=remW/A[i].w;

}

remW-=A[i].w;

if(remW<0.0)

remW=0.0;

tot+= p\_;

System.out.println(A[i].id+"\t\t "+p\_+"\t\t "+w\_+" \t\t"+r\_+"\t\t "+remW);

}

}

if(remW>=0.0 && flag==0){

   for(i=0;i<n;i++){

   r\_=A[i].p/A[i].w;

   w\_=A[i].w;

p\_=A[i].p;

if(A[i].w>remW){

w\_=remW/A[i].w;

p\_\*=remW/A[i].w;

}

remW-=A[i].w;

if(remW<0.0)

remW=0.0;

tot+= p\_;

System.out.println(A[i].id+"\t\t "+p\_+"\t\t "+w\_+" \t\t"+r\_+"\t\t "+remW);

}

}

System.out.println("\t Max Profit = "+tot);

}

public static OBJ[] merge(OBJ A[],int l,int m,int u,char C){

int n1 = m-l+1,n2 = u-m,i,j,k;double rL,rR;

OBJ[] L = new OBJ[n1+1],R = new OBJ[n2+1];

for(i=0;i<n1;i++)

L[i]=A[l+i];

for(j=0;j<n2;j++)

R[j]=A[m+j+1];

L[n1]= new OBJ(0,100000,100000,0); R[n2]= new OBJ(0,100000,100000,0);

i=0;j=0;k=l;

if(C=='p'){

while(i<n1 && j<n2){

if (L[i].p<=R[j].p){

A[k]=L[i];

i++;

}else{

A[k]=R[j];

j++;

}k++;

}

}else if(C=='w'){

while(i<n1 && j<n2){

if (L[i].w<=R[j].w){

A[k]=L[i];

i++;

}else{

A[k]=R[j];

j++;

}k++;

}

}else{

while(i<n1 && j<n2){

if (L[i].r<=R[j].r){

A[k]=L[i];

i++;

}else{

A[k]=R[j];

j++;

}k++;

}

}

while(i<n1){

A[k]=L[i];

k++;

i++;

}

while(j<n2){

A[k]=R[j];

k++;

j++;

}

return A;

}

public static void prt(OBJ obj[],int l,int u,int way){

System.out.print("Object   Profit   Weight   Ratio   \n");

if(way==1){

for(int i=l;i<=u;i++)

System.out.println(obj[i].id+"\t\t "+obj[i].p+"\t\t "+obj[i].w+"\t\t "+obj[i].r);

}else{

for(int i=u;i>=l;i--)

System.out.println(obj[i].id+"   "+obj[i].p+"   "+obj[i].w+"   "+obj[i].r);

}

}

}

class OBJ{

int id, p, w;

double r;

OBJ(int id,int p,int w,double r){

this.id=id;

this.p=p;

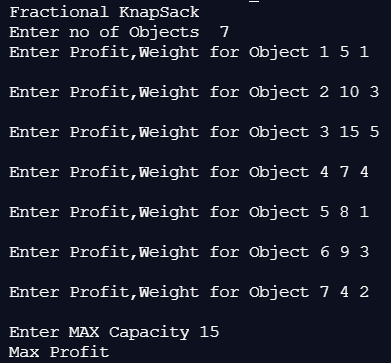
this.w=w;

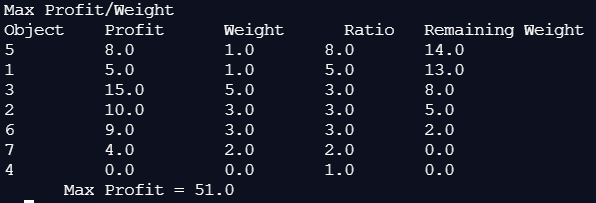
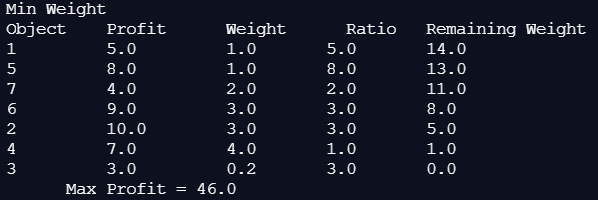
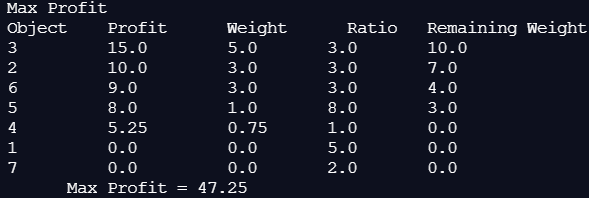
this.r=r;

}

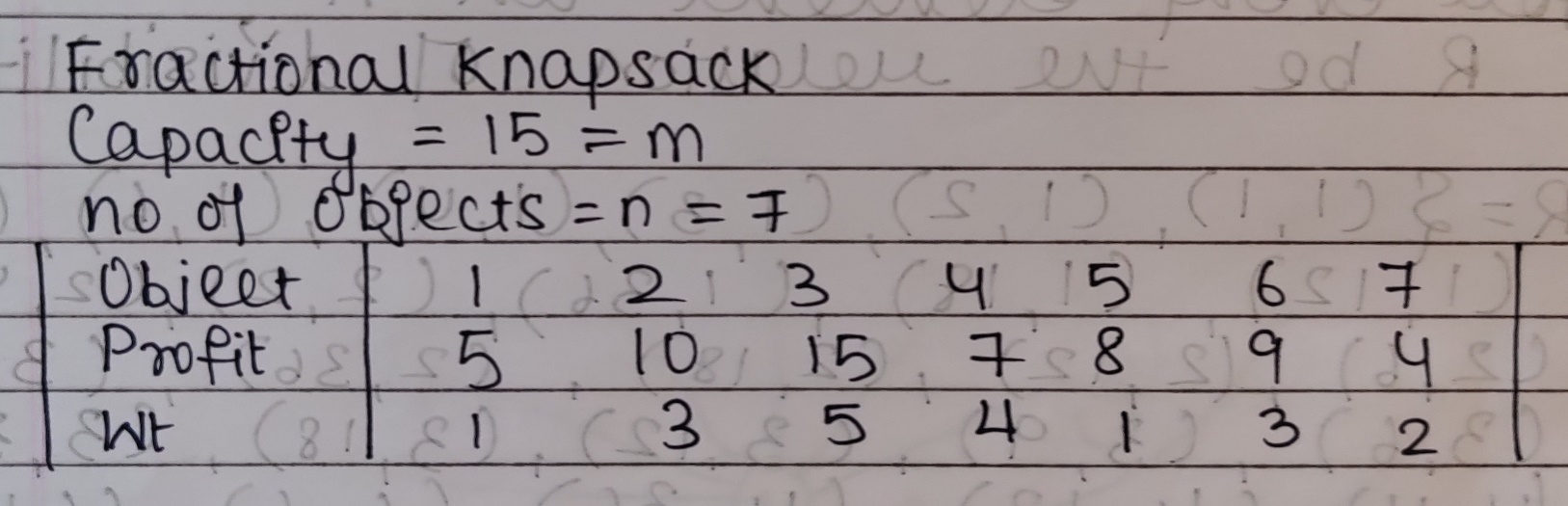
}

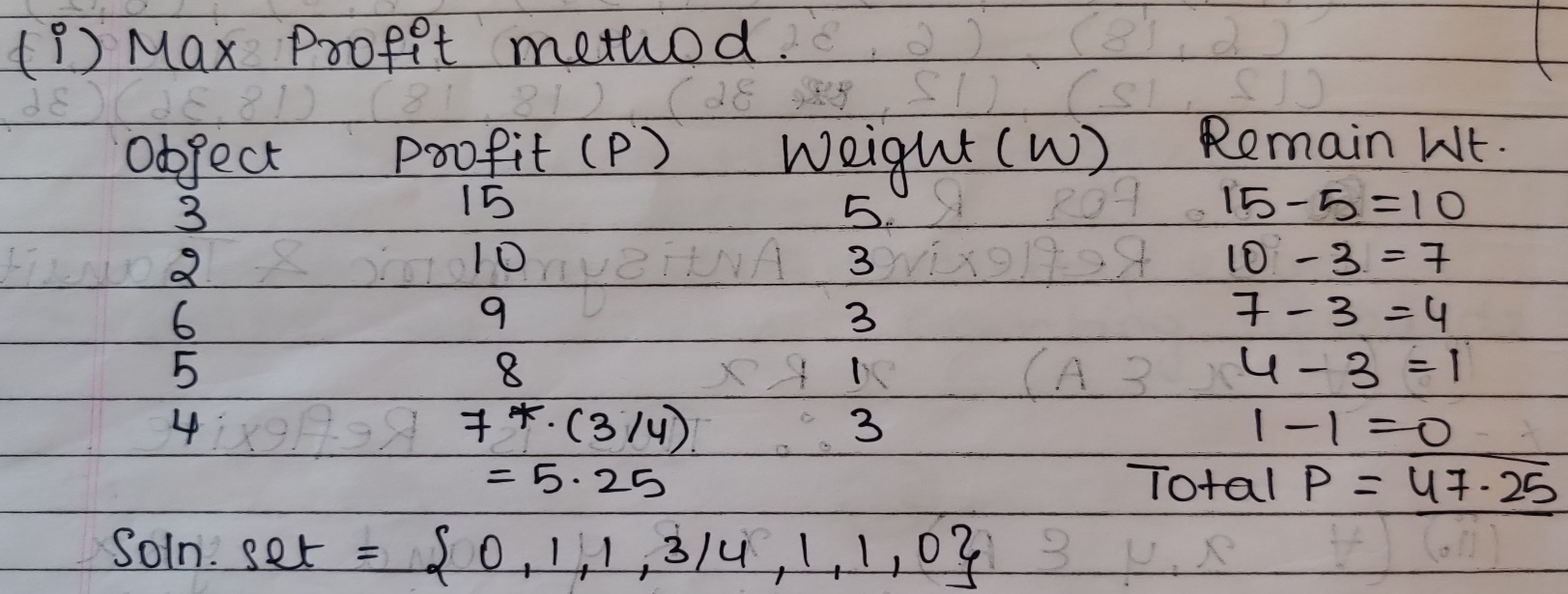
**Output**

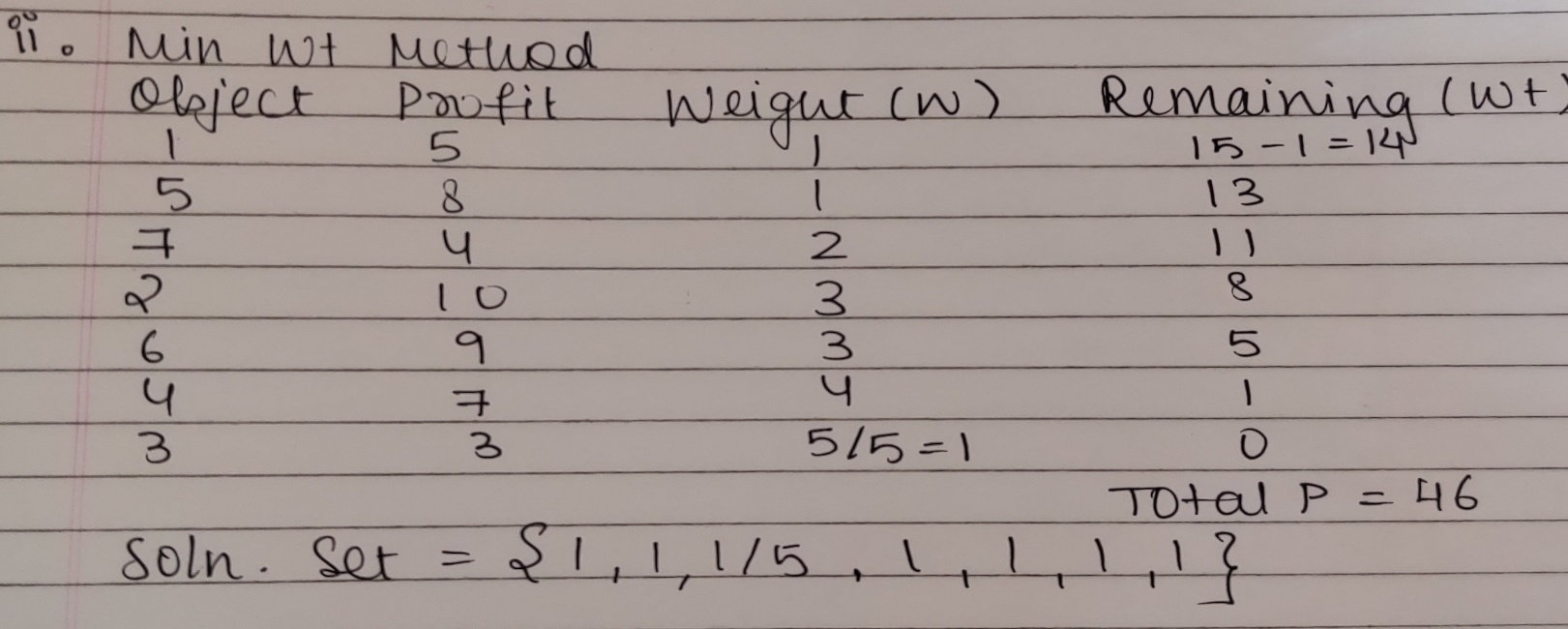


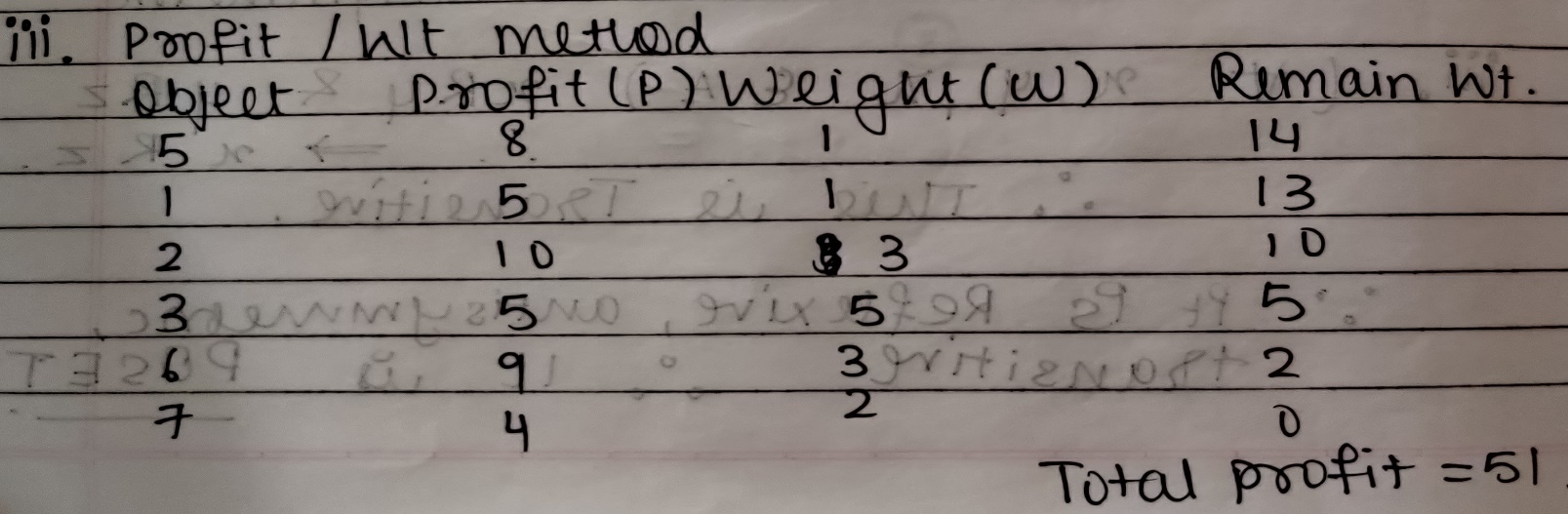


**Example: Knapsack Problem**



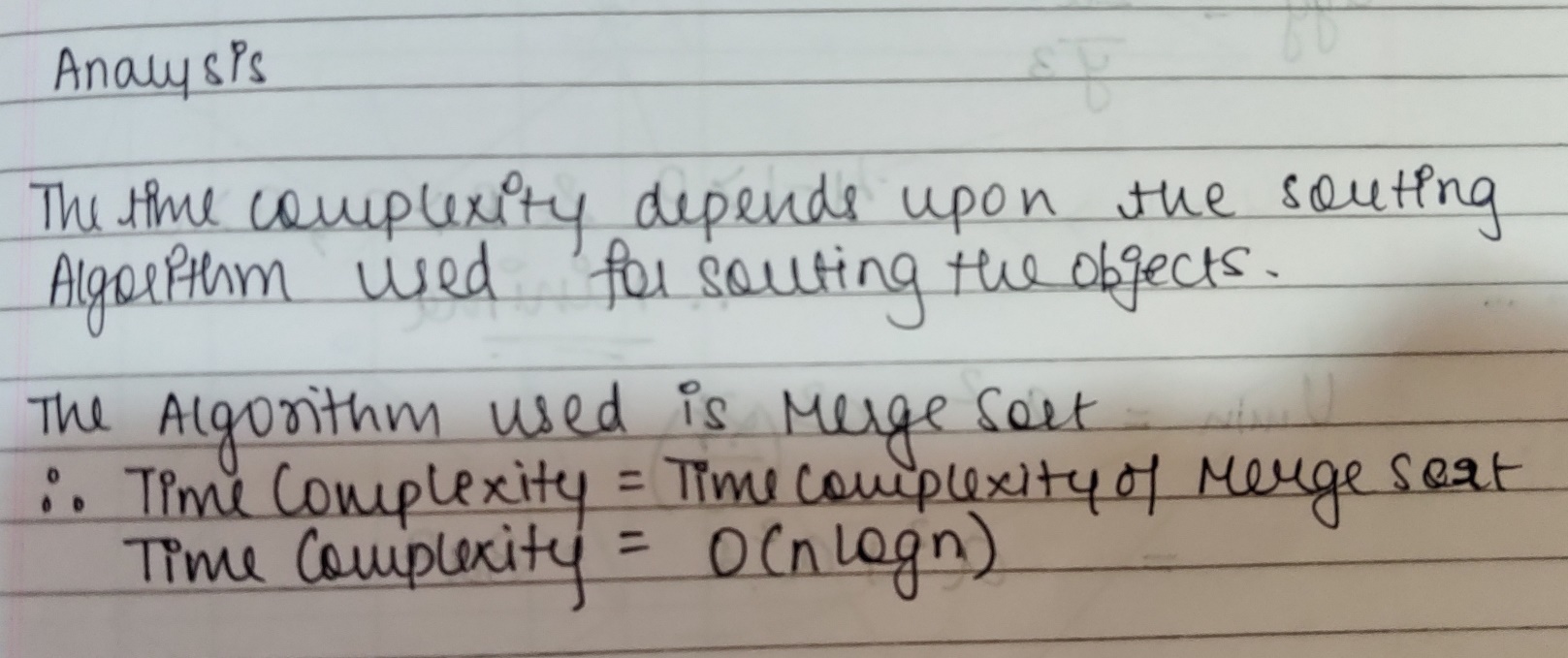






Soln Set = {1,1,1,0,1,1,1} for Profit/Wt method

**Analysis of Knapsack Problem algorithm:**



**CONCLUSION:**

We successfully implemented Fractional Knapsack in Java ; thus obtaining correct necessary output with

maximum profit.