**Experiment No: 10**

**AIM:** Implementation 0/1 Knapsack Problem(Dynamic Programming) and obtaining its step count.

**THEORY:**

Given weights and profits of n items, we need to put these items in a **knapsack** of capacity W to get the maximum total profit in the knapsack. However, unlike fractional knapsack, we must either consider the entire object, or not consider it at all.

*In other words, given two integer arrays val[0..n-1] and wt[0..n-1] which represent values and weights associated with n items respectively. Also given an integer W which represents knapsack capacity, we need to compute the maximum value subset of val[] such that sum of the weights of this subset is smaller than or equal to W. You cannot break an item, either pick the complete item or don’t pick it (0-1 property).*

**ALGORITHM:**

PW = record{floatp; float w; }

**AlgorithmDKnap**(p,w,x,n,to)

{

// pair[]is an array of PWs.

b[0]:=1;pair[l].p:=pair[l].w:=0.0;// S0

t:=l;h:=1;// Start and end of S0

b[l] :=next:=2; // Next free spot in pair[]

**for** i :=1 to n – 1 **do**

{

// GenerateS

k :=t;

u :=Largest(pair,w, t, h, i,to);

for j :=t to u do

{

// GenerateS\\~l and merge. 13 pp :=pair[j].p+ p[i];ww :=pair[j].w+ w[i];

while ((k < h) and (pair[k].w< ww)) do

{

pair [next].p :=pair[k].p;

pair[next].w:=pair[k].w,

next:=next+ 1;k :=k + 1;

}

if ((k < h) and (pair[k].w < ww)) then

{

if pp < pair[k].p then pp :=pair[k].p;

k:=k+l;

}

if pp > pair[next-1].p then

{

pair[next].p:=pp;pair[next].w:=ww;

next:=next+ 1;

}

while ((k < h) and (pair[k].p< pair[next - 1].p))

do k:=k+ 1;

}

// Merge in remaining terms from Si-1

while (k < h) do

{

pair[next].p:=pair[k].p; pair[next].w:=pair[k].w;

next:=next+1; k :=k + 1;

}

// Initialize for Si+l.

t :=h + 1;h :=next-1; b[i + 1]:=next;

}

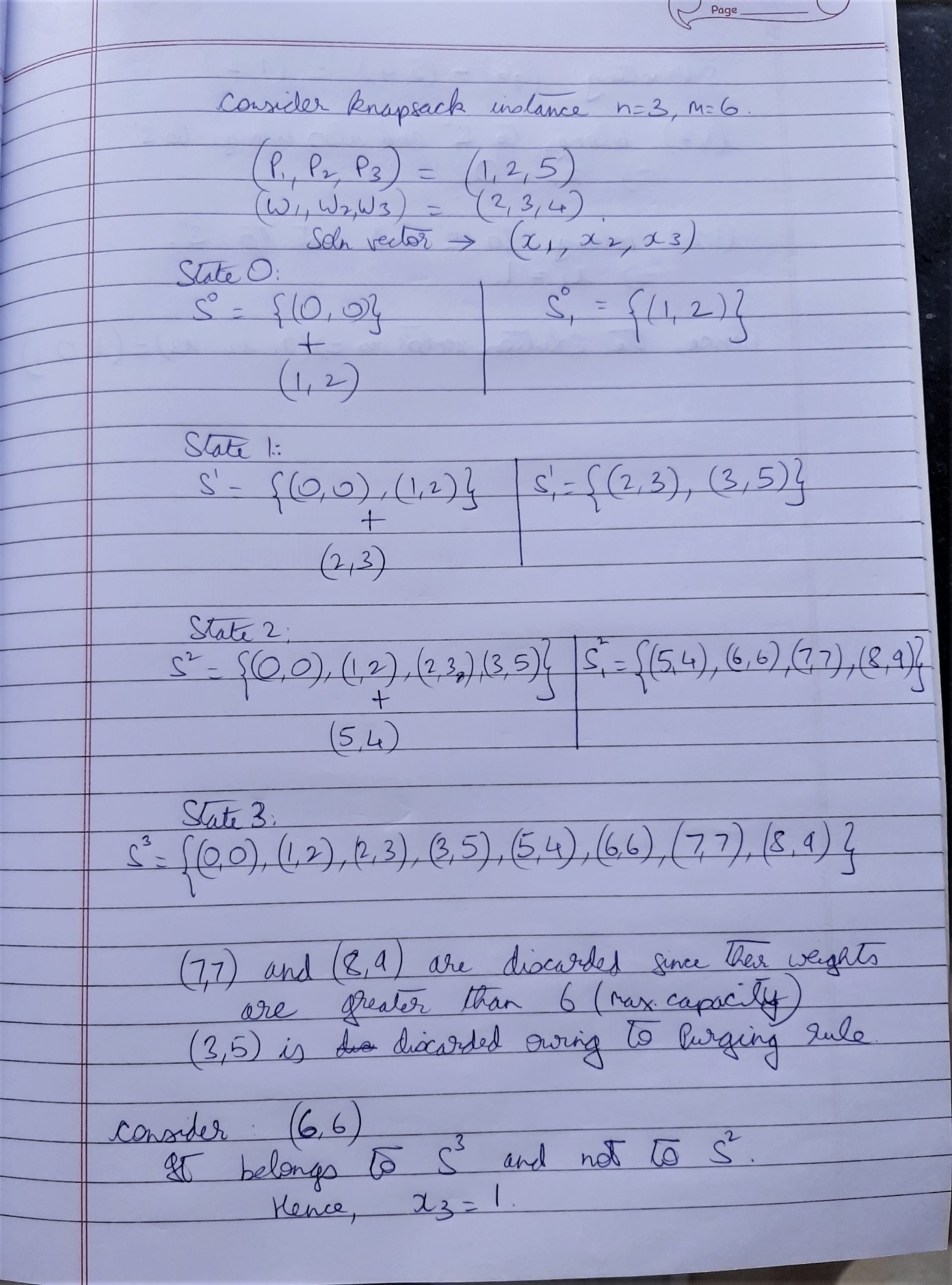
TraceBack(p,w,pair,x,to, n);

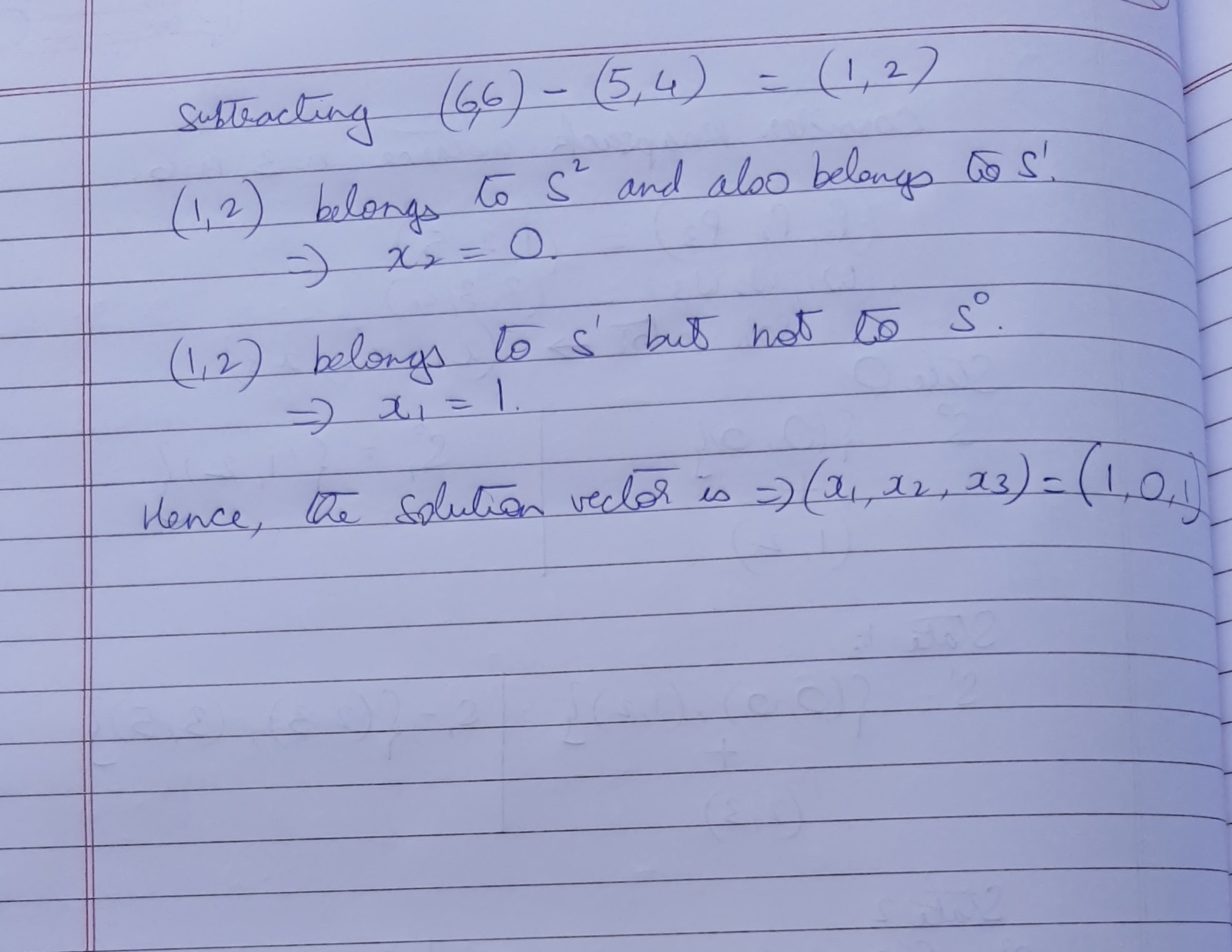
}

*Time Complexity*

• The time complexity of this algorithm is O (2^n).

*Problem Tracing*





PROGRAM IMPLEMENTATION:

#include<iostream>

using namespace std;

int count=0;

struct record

{

float p;

float w;

}pairs[100];

void show\_tuple(int N)

{

for(int i=0;i<N;i++)

cout<<"("<<pairs[i].p<<","<<pairs[i].w<<")"<<" ";

}

void Dknapsack(int n, float W, float \*pt, float \*wt)

{

pairs[0].p=pairs[0].w=0.0; //S0;

int N=1; //Number of tuples currently in pairs

int j=1; //start index to fill up more tuples are S0;

count+=4; //assignments above

cout<<"\n\nThe tuples are:\n";

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

{

int c=0; //start index of pairs[]

count+=2; //for loop and c

while(c<N)

{

pairs[j].p = pairs[c].p + pt[i];

pairs[j].w = pairs[c].w + wt[i];

j++; c++;

count+=5; //while loop and assignments

}

count++; //end while condition

show\_tuple(N);

N=j;count++;

cout<<endl;

}

count++; //end of for loop

show\_tuple(N);

cout<<endl;

int num = N,soln[n],m=n; //num is number of tuples. soln[] is the solution vector and n is no of objs

float large;

for(int i=0;i<num;i++)

{

count+=2; //if and for

if(pairs[i].p>large && pairs[i].w<=W)

large = pairs[i].p,count++;

}

count++; //end of for

while(num>0)

{

num/=2;

count+=2; //while and num

int i;

for(i=0;i<num && pairs[i].p!=large;i++);

count++; //if

if(i==num)

{

soln[--m]=1;

large = large - pt[m];

count+=2;

}

else

soln[--m]=0;

}

cout<<"\nThe solution vector: ( ";

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

cout<<soln[i]<<" ";

cout<<")";

float max=0;

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

if(soln[i])

max+=pt[i];

cout<<"\nThe Maximum profit is:"<<max<<endl<<endl;

}

int main()

{

int n;

cout<<"Enter number of objects:";

cin>>n;

float W, pt[n], wt[n];

cout<<"\nEnter the "<<n<<" profits: ";

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

cin>>pt[i];

cout<<"\nEnter the "<<n<<" weights: ";

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

cin>>wt[i];

cout<<"\nEnter the capacity of the knapsack:";

cin>>W;

Dknapsack(n,W,pt,wt);

cout<<"\nStepCount="<<count<<endl;

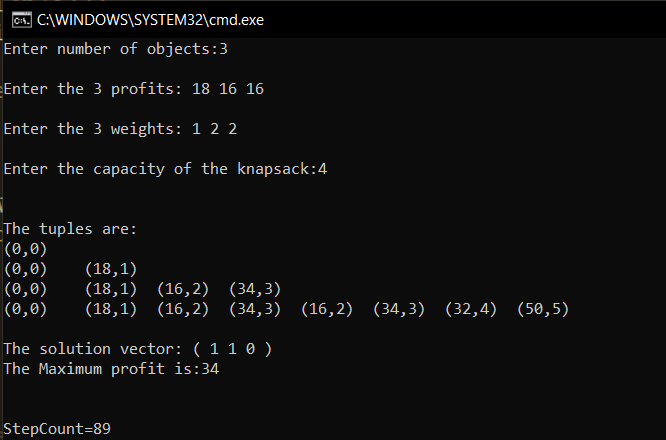
return 0;

}

OUTPUTS:

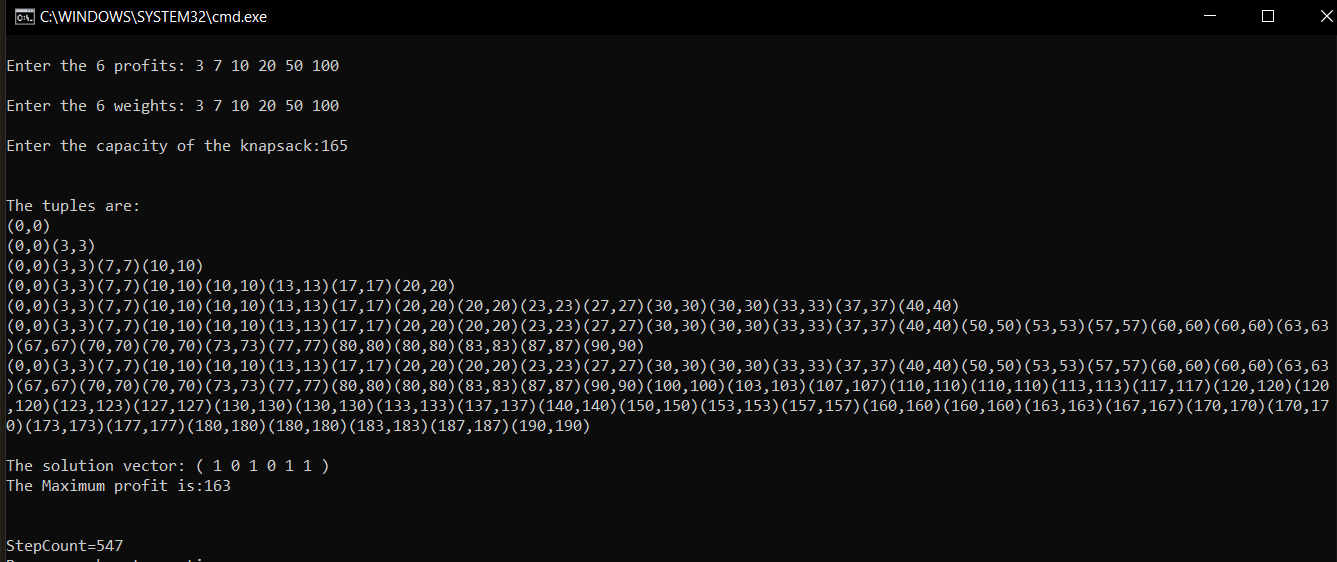
1. When n=3

**Count=89**



1. When n=6

**Count=547**

****

**Conclusion**:

* **Time complexity of 0/1 Knapsack problem was found to be O (2^n)**
* **The 0/1 Knapsack problem is essential when the objects to be included are to be considered as a whole**