**EXPERIMENT N0-5**

**AIM: Implementation of Fractional Knapsack problem.**

**CODE:**

#include<stdio.h>

int max(int a, int b) { return (a > b)? a : b; }

int knapSack(int W, int wt[], int val[], int n)

{

int i, w;

int K[n+1][W+1];

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

{

for (w = 0; w <= W; w++)

{

if (i==0 || w==0)

K[i][w] = 0;

else if (wt[i-1] <= w)

K[i][w] = max(val[i-1] + K[i-1][w-wt[i-1]], K[i-1][w]);

else

K[i][w] = K[i-1][w];

}

}

return K[n][W];

}

int main()

{

int i, n, val[20], wt[20], W;

printf("Enter number of items:");

scanf("%d", &n);

printf("Enter size of knapsack:");

scanf("%d", &W);

printf("Enter value and weight of items:\n");

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

scanf("%d%d", &val[i], &wt[i]);

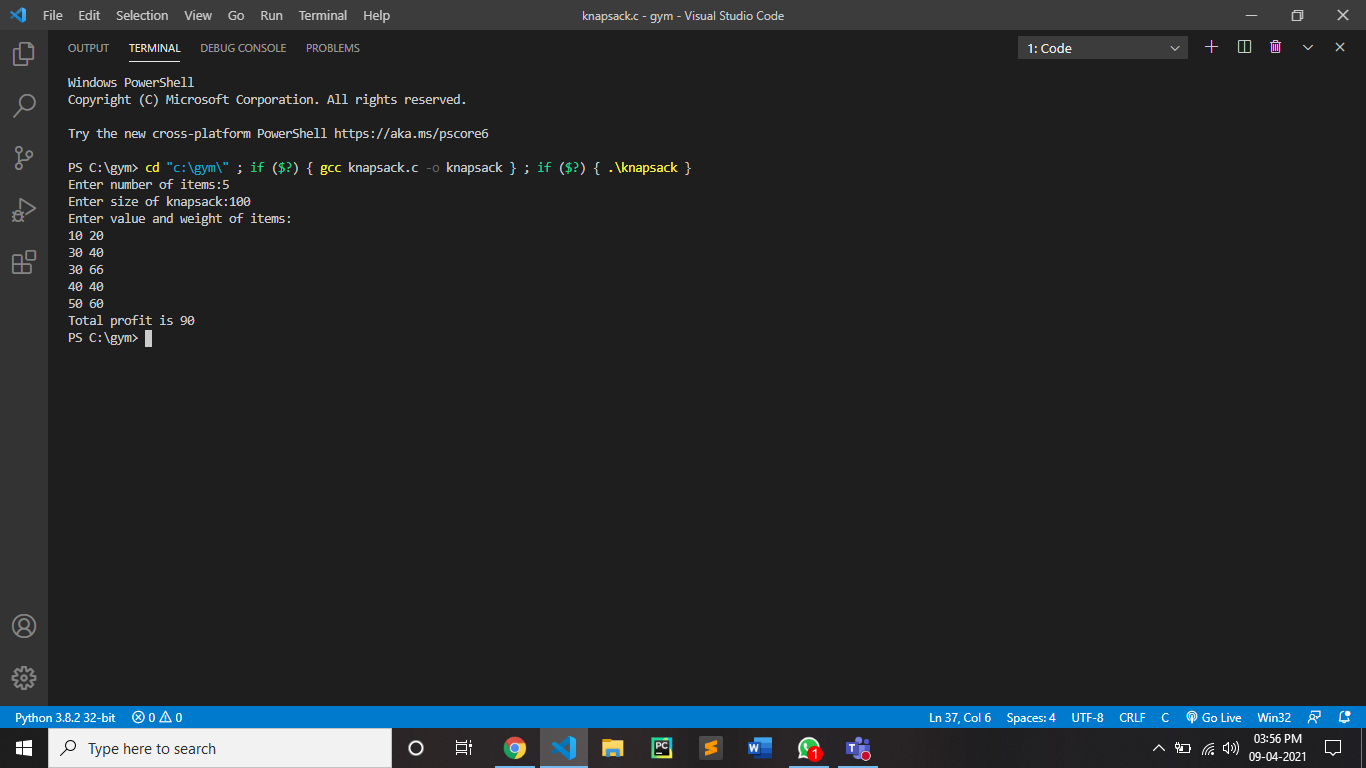
}

printf("Total profit is %d", knapSack(W, wt, val, n));

return 0;

}

**OUTPUT:**



CONCLUSION:

By performing above practical we can conclude that for n items, knapsack has 2^n choices. So brute force approach runs in O(2^n) time. We can improve performance by sorting items in advance. Using merge sort or heap sort, n items can be sorted in O(nlog2n) time. Merge sort and heap sort are non-adaptive and their running time is same in best, average and worst case. To select the items, we need one scan to this sorted list, which will take O(n) time. Total time required is T(n) = O(nlog2 n) + O(n) = O(nlog2 n)