QUESTION:

Given 2 sorted arrays A and B of size n each. Write an algorithm to find the median of the array obtained after merging the above 2 arrays(i.e. array of length 2n). The complexity should be O(log(n))

Input : ar1[] = {1, 12, 15, 26, 38}
ar2[] = {2, 13, 17, 30, 45}
Output : 16

Explanation :
After merging two arrays, we get
{1, 2, 12, 13, 15. 17, 26, 30, 38, 45}
Middle two elements are 15 and 17
Average of middle elements is (15 + 17)/2
which is equal to 16

EXP NO:	**MEDIAN FINDING**
DATE :	

AIM:

To find the median of two arrays after merging it.

PSEUDOCODE:

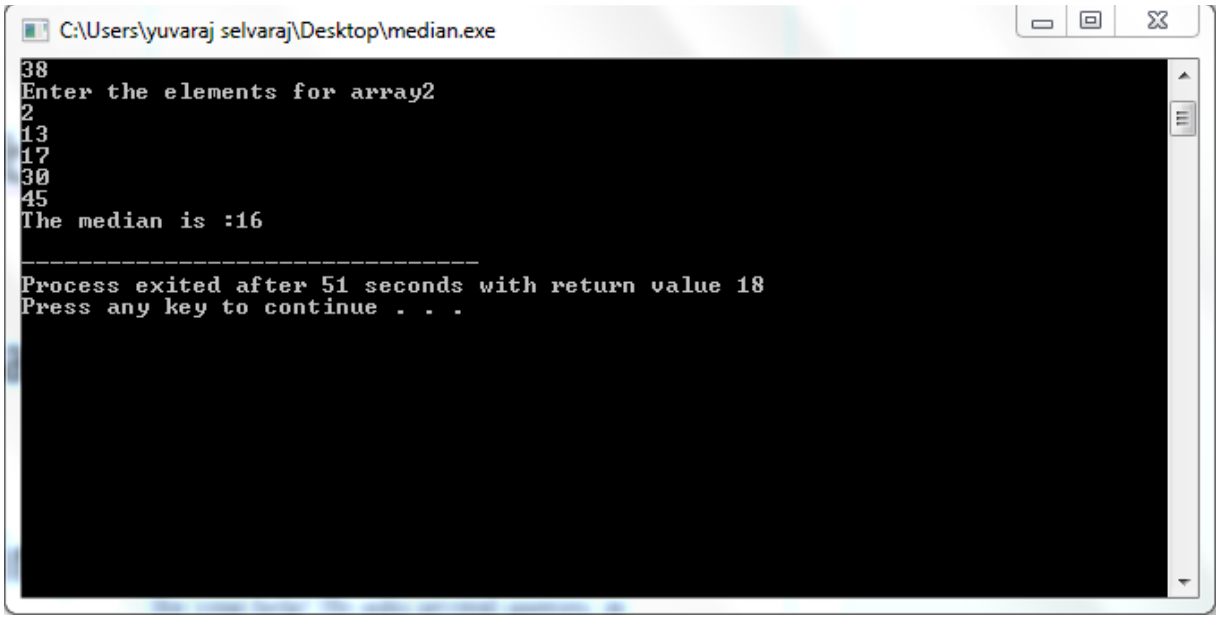
```
m1=median(arr1,n);
m2=median(arr2,n);
        if(m1<m2)
        {
                if(n%2==0)
                        return getmedian(arr1+(n/2-1),arr2,n-n/2+1);
                return getmedian(arr1+(n/2),arr2,n-n/2);
        }
        else
        {       if(n%2==0)
                        return getmedian(arr1,arr2+(n/2-1),n-n/2+1);
                return getmedian(arr1,arr2+(n/2),n-n/2);
        }
```

SOURCE CODE:

```
#include<stdio.h>
int median(int arr[],int n)
{
int middle=n/2;
        if(n%2==0)

                        return (arr[middle]+arr[middle+1])/2 ;
        else
                return arr[middle];
}
int max(int val1,int val2)
{
if(val1>val2)
        return val1;
else
        return val2;
}
int min(int val1,int val2)
{
if(val1>val2)
        return val2;
else
        return val1;
}
```

```
int getmedian(int arr1[],int arr2[],int n)
{
	int m1,m2;
	if(n==1)
	return (arr1[0]+arr2[0])/2;
	if(n==2)
	return (max(arr1[0],arr2[0])+min(arr1[1],arr2[1]))/2;
	m1=median(arr1,n);
	m2=median(arr2,n);
	if(m1<m2)
	{
		if(n%2==0)
			return getmedian(arr1+(n/2-1),arr2,n-n/2+1);
		return getmedian(arr1+(n/2),arr2,n-n/2);
	}
	else
	{	if(n%2==0)
			return getmedian(arr1,arr2+(n/2-1),n-n/2+1);
		return getmedian(arr1,arr2+(n/2),n-n/2);
	}
}
void main()
{
printf("Enter the number of values    :");
int n,i;
scanf("%d",&n);
int arr1[n],arr2[n];
printf("Enter the elements for array1\n");
for(i=0;i<n;i++)
{
	scanf("%d",&arr1[i]);
}
printf("Enter the elements for array2\n");
for(i=0;i<n;i++)
{
	scanf("%d",&arr2[i]);
}
printf("The median is :%d\n",getmedian(arr1,arr2,n));
}

\
```

OUTPUT:



RESULT:

Thus the program was compiled and executed successfully.

QUESTION:

To perform heap sort for the given array using heapify function

Input:

5

5

4

3

2

1

Output:

1

2

3

4

5

EXP NO:	**HEAP SORT**
DATE :	

AIM:

To Find the sorted array using heap sort

PSEUDOCODE:

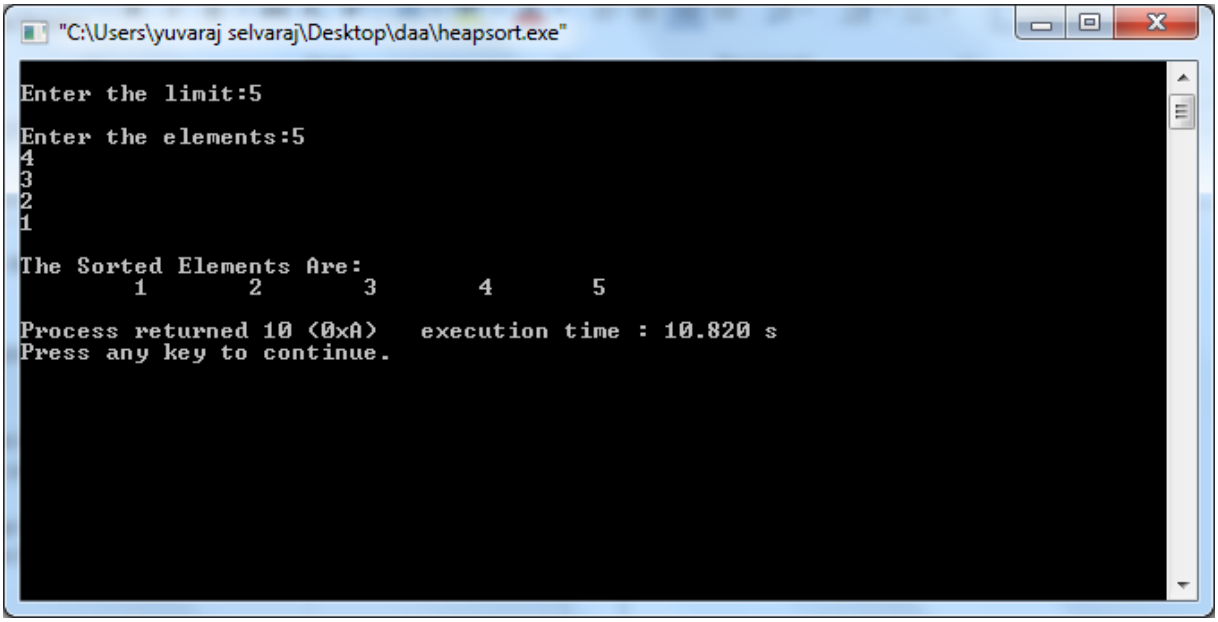
```
void heapify(int a[],int n) {

for (k=1;k<n;k++) {
item = a[k];
i = k;
j = (i-1)/2;
while((i>0)&&(item>a[j])) {
a[i] = a[j];
i = j;
j = (i-1)/2;
}
a[i] = item;
}}
```

SOURCE CODE:

```
#include<stdio.h>
void heapsort(int[],int);
void heapify(int[],int);
void adjust(int[],int);
main() {
int n,i,a[50];
printf("\nEnter the limit:");
scanf("%d",&n);
printf("\nEnter the elements:");
for (i=0;i<n;i++)
scanf("%d",&a[i]);
heapsort(a,n);
printf("\nThe Sorted Elements Are:\n");
for (i=0;i<n;i++)
printf("\t%d",a[i]);
printf("\n");
}
void heapsort(int a[],int n)
{
int i,t;
heapify(a,n);
for (i=n-1;i>0;i--) {
t = a[0];
```

```
a[0] = a[i];
a[i] = t;
adjust(a,i);
}}
void heapify(int a[],int n) {
int k,i,j,item;
for (k=1;k<n;k++) {
item = a[k];
i = k;
j = (i-1)/2;
while((i>0)&&(item>a[j])) {
a[i] = a[j];
i = j;
j = (i-1)/2;
}
a[i] = item;
}}
void adjust(int a[],int n) {
int i,j,item;
j = 0;
item = a[j];
i = 2*j+1;
while(i<=n-1) {
if(i+1 <= n-1)
if(a[i] <a[i+1])
i++;
if(item<a[i]) {
a[j] = a[i];
j = i;
i = 2*j+1;
}
else
break;
}
a[j] = item;
}
```

OUTPUT:



RESULT:

Thus the program was compiled and executed successfully.

QUESTION:
To multiply two large numbers using dvide and conquer strategy

Input:
100
100

Output:
10000

EXP NO:	**MULTIPLICATION OF TWO LARGE INTEGERS**
DATE :	

AIM:

To implement the program to find the product of two large integers.

PSEUDOCODE:

```
for (int i=n1-1; i>=0; i--)
   {
      int carry = 0;
      int n1 = num1[i] - '0';
      i_n2 = 0;
      for (int j=n2-1; j>=0; j--)
      {
         int n2 = num2[j] - '0';
         int sum = n1*n2 + result[i_n1 + i_n2] + carry;
         carry = sum/10;
         result[i_n1 + i_n2] = sum % 10;

         i_n2++;
      }
      if (carry > 0)
         result[i_n1 + i_n2] += carry;
      i_n1++;
   }
   int i = result.size() - 1;
   while (i>=0 && result[i] == 0)
     i--;
   if (i == -1)
     return "0";
   string s = "";
   while (i >= 0)
      s += std::to_string(result[i--]);

   return s;
```

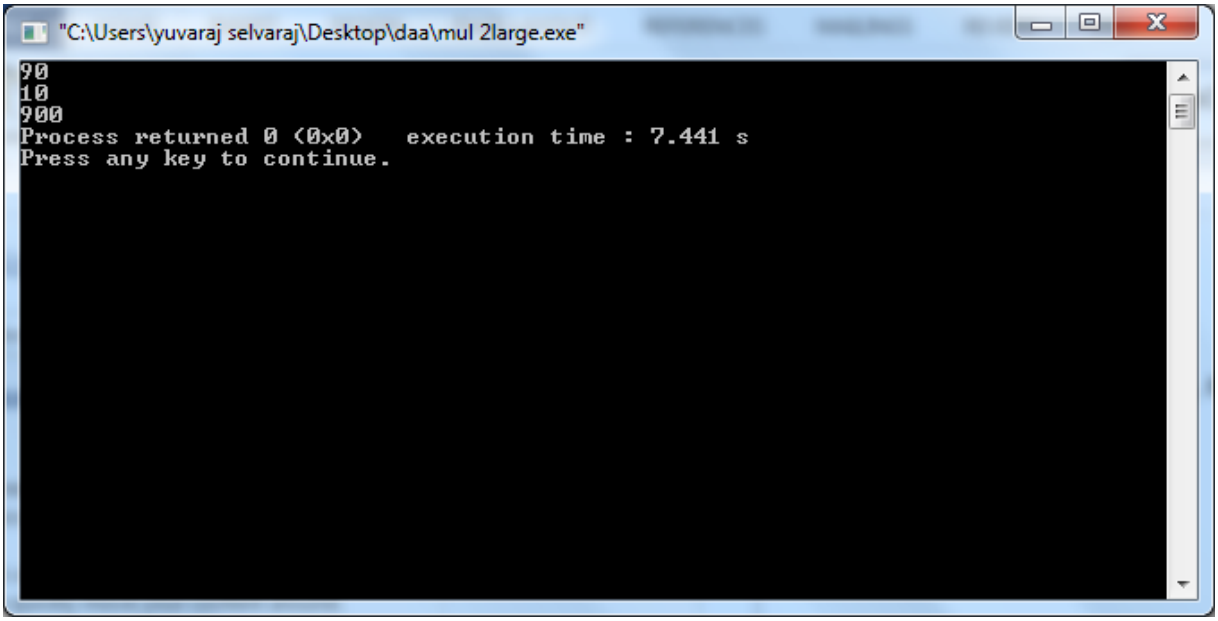
SOURCE CODE:

```
#include<bits/stdc++.h>
using namespace std;
string multiply(string num1, string num2)
{
   int n1 = num1.size();
```

```
    int n2 = num2.size();
    if (n1 == 0 || n2 == 0)
      return "0";
    vector<int> result(n1 + n2, 0);
    int i_n1 = 0;
    int i_n2 = 0;
    for (int i=n1-1; i>=0; i--)
    {
        int carry = 0;
        int n1 = num1[i] - '0';
        i_n2 = 0;
        for (int j=n2-1; j>=0; j--)
        {
            int n2 = num2[j] - '0';
            int sum = n1*n2 + result[i_n1 + i_n2] + carry;
            carry = sum/10;
            result[i_n1 + i_n2] = sum % 10;

            i_n2++;
        }
        if (carry > 0)
            result[i_n1 + i_n2] += carry;
        i_n1++;
    }
    int i = result.size() - 1;
    while (i>=0 && result[i] == 0)
      i--;
    if (i == -1)
      return "0";
    string s = "";
    while (i >= 0)
        s += std::to_string(result[i--]);

    return s;
}
int main()
{
    string str1 ;
    string str2 ;
```

```
    cin>>str1;
    cin>>str2;
    cout << multiply(str1, str2);
    return 0;
}
```

OUTPUT:



RESULT:

Thus the program was compiled and executed successfully.

QUESTION:

Given a cost matrix cost[][] and a position (m, n) in cost[][], write a function that returns cost of minimum cost path to reach (m, n) from (0, 0). Each cell of the matrix represents a cost to traverse through that cell.

Total cost of a path to reach (m, n) is sum of all the costs on that path (including both source and destination).

You can only traverse down, right and diagonally lower cells from a given cell, i.e., from a given cell (i, j), cells (i+1, j), (i, j+1) and (i+1, j+1) can be traversed.

You may assume that all costs are positive integers.

Input:

1	2	3
4	8	2
1	5	3

(2,2)

Output:

The output is an integer indicating the sum of the shortest path.

Ans:8

EXP NO:	MATRIX MIN COST
DATE :	

AIM:

To find the shortest path to go to a position in a given matrix

PSEUDOCODE:

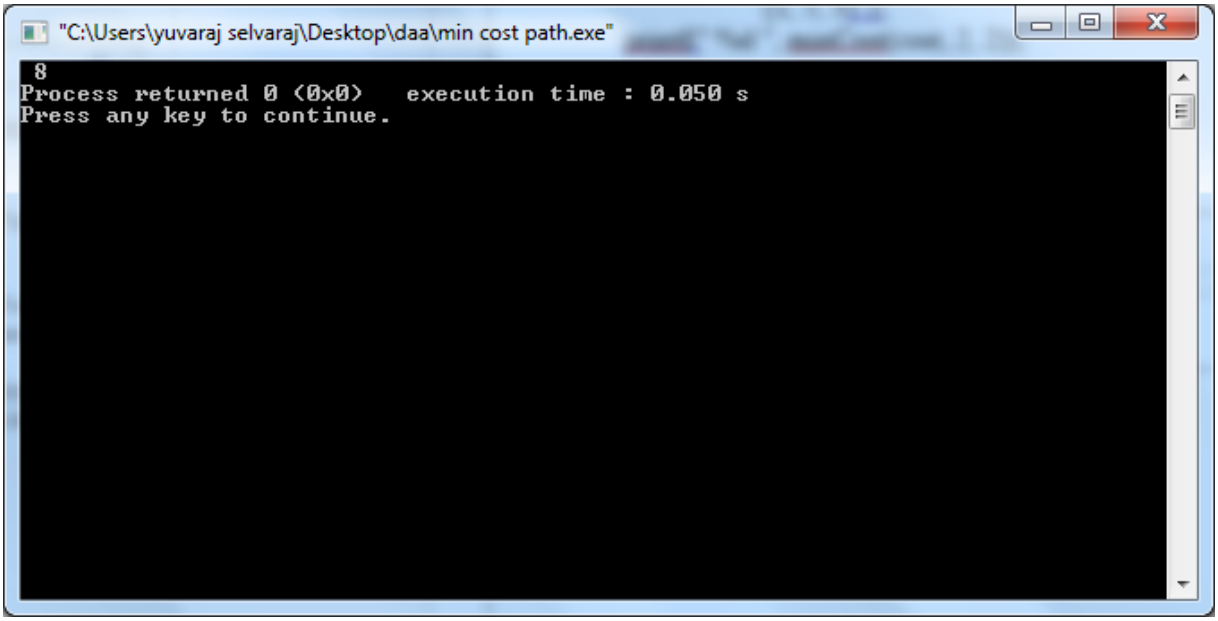
```
    for (i = 1; i <= m; i++)
      tc[i][0] = tc[i-1][0] + cost[i][0];


    for (j = 1; j <= n; j++)
      tc[0][j] = tc[0][j-1] + cost[0][j];

    for (i = 1; i <= m; i++)
      for (j = 1; j <= n; j++)
         tc[i][j] = min(tc[i-1][j-1],
                        tc[i-1][j],
                        tc[i][j-1]) + cost[i][j];

    return tc[m][n];
```

SOURCE CODE:

```
#include<stdio.h>
#include<limits.h>
#define R 3
#define C 3

int min(int x, int y, int z);

int minCost(int cost[R][C], int m, int n)
{
    int i, j;

    int tc[R][C];

    tc[0][0] = cost[0][0];


    for (i = 1; i <= m; i++)
      tc[i][0] = tc[i-1][0] + cost[i][0];


    for (j = 1; j <= n; j++)
      tc[0][j] = tc[0][j-1] + cost[0][j];

    for (i = 1; i <= m; i++)
```

```
        for (j = 1; j <= n; j++)
            tc[i][j] = min(tc[i-1][j-1],
                           tc[i-1][j],
                           tc[i][j-1]) + cost[i][j];

     return tc[m][n];
}
 int min(int x, int y, int z)
{
   if (x < y)
      return (x < z)? x : z;
   else
      return (y < z)? y : z;
}

int main()
{
   int cost[R][C] = { {1, 2, 3},
                      {4, 8, 2},
                      {1, 5, 3} };
   printf(" %d ", minCost(cost, 2, 2));
   return 0;
}
```

OUTPUT:



RESULT:

Thus the program was compiled and executed successfully

QUESTION:
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, find out 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

```
value[] = {60, 100, 120};
weight[] = {10, 20, 30};
W = 50;

Solution: 220
```

```
Weight = 10; Value = 60;
Weight = 20; Value = 100;
Weight = 30; Value = 120;
Weight = (20+10); Value = (100+60);
Weight = (30+10); Value = (120+60);
Weight = (30+20); Value = (120+100);
Weight = (30+20+10) > 50
```

EXP NO:	0-1 KNAPSACK PROBLEM
DATE :	

AIM:

To find the max profit either by taking the cost or dropping

PSEUDOCODE:

```
for(i = 1; i <= n; i++)
{
for(j = 1; j <= Capacity; j++)
{
if((j - Weight[i]) < 0)
Knapsack01[i][j] = Knapsack01[i-1][j];
else
Knapsack01[i][j] = Max(Knapsack01[i-1][j], Profit[i] +
Knapsack01[i-1][j-Weight[i]]);
}
}for(i = 0; i <= n; i++)
{
for(j = 0; j <= Capacity; j++)
printf("%d ", Knapsack01[i][j]);
printf("\n");
}
return Knapsack01[n][Capacity];
```

SOURCE CODE:

```
#include<stdio.h>
int Knapsack_01(int Item[], int Profit[], int Weight[], int Capacity, int n);
void main()
{
int n, i;
printf("Enter the value of n : ");
scanf("%d", &n);
int Item[n], Profit[n], Weight[n], Capacity;
printf("Enter %d items Profit and Weight :\n", n);
for(i=1; i<=n; i++)
{
Item[i] = i;
printf("Enter item %d Profit and Weight : ", Item[i]);
scanf("%d %d", &Profit[i], &Weight[i]);
}
printf("Enter the Knapsack Capacity : ");
scanf("%d", &Capacity);
printf("Maximum Profit attained with Knapsack of capacity %d with %d Items is %d\n",
Capacity, n, Knapsack_01(Item, Profit, Weight, Capacity, n));
```

```
}
int Knapsack_01(int Item[], int Profit[], int Weight[], int Capacity, int n)
{
int Knapsack01[n + 1][Capacity + 1], i, j;
for(i = 0; i <= n; i++)
Knapsack01[i][0] = 0;
for(j = 0; j <= Capacity; j++)
Knapsack01[0][j] = 0;
for(i = 1; i <= n; i++)
{
for(j = 1; j <= Capacity; j++)
{
if((j - Weight[i]) < 0)
Knapsack01[i][j] = Knapsack01[i-1][j];
else
Knapsack01[i][j] = Max(Knapsack01[i-1][j], Profit[i] +
Knapsack01[i-1][j-Weight[i]]);
}
}for(i = 0; i <= n; i++)
{
for(j = 0; j <= Capacity; j++)
printf("%d ", Knapsack01[i][j]);
printf("\n");
}
return Knapsack01[n][Capacity];
}
int Max(int a, int b)
{
return (a>b) ? a : b;
}
```

OUTPUT:

```
"C:\Users\yuvaraj selvaraj\Desktop\daa\knapsackdp.exe"
Enter the value of n : 3
Enter 3 items Profit and Weight :
Enter item 1 Profit and Weight : 60
10
Enter item 2 Profit and Weight : 100
20
Enter item 3 Profit and Weight : 120
30
Enter the Knapsack Capacity : 50
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 60 60 60 60 60 60 60 60 60 60 60 60 60 60 60 60 60 60 60 60
60 60 60 60 60 60 60 60 60 60 60 60 60 60 60 60 60 60 60 60 60
0 0 0 0 0 0 0 0 0 0 60 60 60 60 60 60 60 60 60 60 100 100 100 100 100 100 100 10
0 100 100 160 160 160 160 160 160 160 160 160 160 160 160 160 160 160 160 160 16
0 160 160 160
0 0 0 0 0 0 0 0 0 0 60 60 60 60 60 60 60 60 60 60 100 100 100 100 100 100 100 10
0 100 100 160 160 160 160 160 160 160 160 160 160 180 180 180 180 180 180 180 18
0 180 180 220
Maximum Profit attained with Knapsack of capacity 50 with 3 Items is 220

Process returned 73 (0x49)   execution time : 58.616 s
Press any key to continue.
```

RESULT:

Thus the program was compiled and executed successfully.

QUESTION:

Given weights and values of n items, we need put these items in a knapsack of capacity W to get the maximum total value in the knapsack.

In the 0-1 Knapsack problem, we are not allowed to break items. We either take the whole item or don't take it.

Input:

```
Items as (value, weight) pairs
arr[] = {{60, 10}, {100, 20}, {120, 30}}
Knapsack Capacity, W = 50;
```

Output:

```
Maximum possible value = 220
by taking items of weight 20 and 30
```

EXP NO:	**FRACTIONAL KNAPSACK**
DATE :	

AIM:

To find the max profit using fractional knapsack

PSEUDOCODE:

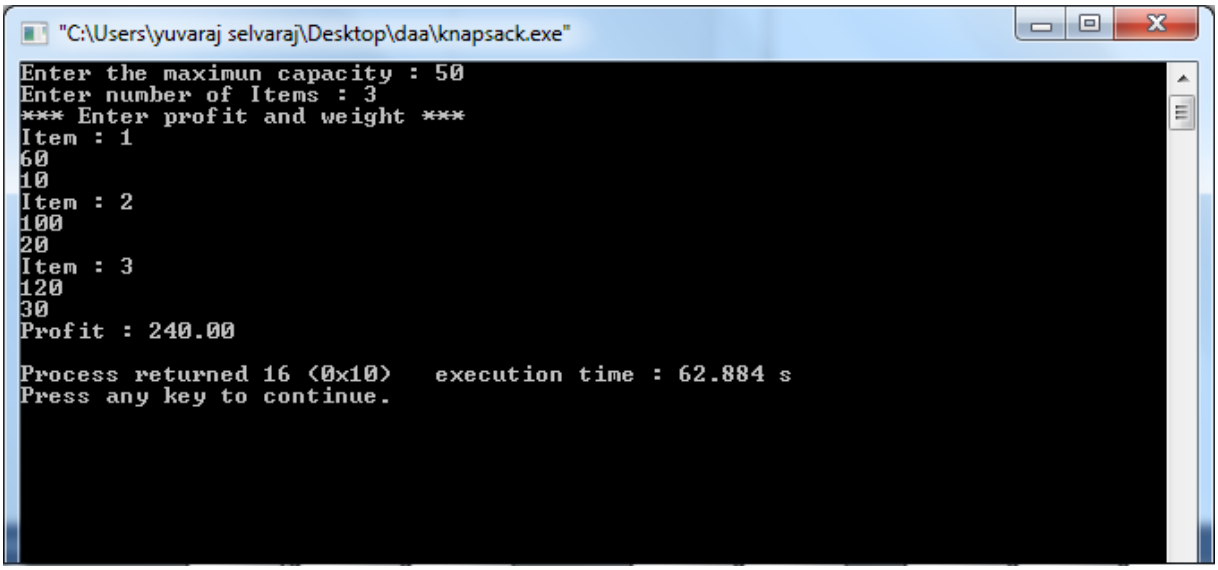
```
 while(rc){
if(w[i]<=rc){
 x[i]=1;
 profit+=p[i]*x[i];
 rc-=w[i];
 }
 else{
 x[i]=(float)rc/(float)w[i];
 profit+=p[i]*x[i];
 rc=0;
 }
 i++;
}
```

SOURCE CODE:

```
#include<stdio.h>
void main(){
int i,j,n;
int max;
int rc;
printf("Enter the maximun capacity : ");
scanf("%d",&max);
printf("Enter number of Items : ");
scanf("%d",&n);
float x[n];
int w[n];
int p[n];
float ratio[n];
float profit=0;
printf("*** Enter profit and weight ***\n");
for(i=0;i<n;i++){
 x[i]=0;
 printf("Item : %d\n",i+1);
 scanf("%d %d",&p[i],&w[i]);
 ratio[i]=(float)p[i]/(float)w[i];
```

```
}
for(i=0;i<n;i++){
 for(j=0;j<n;j++){
  if(ratio[i]>ratio[j]){
   int temp1=w[i];
   w[i]=w[j];
   w[j]=temp1;
   int temp2=p[i];
   p[i]=p[j];
   p[j]=temp2;
   float temp3=ratio[i];
   ratio[i]=ratio[j];
   ratio[j]=temp3;
  }
 }
 }
rc=max;
i=0;
while(rc){
 if(w[i]<=rc){
  x[i]=1;
  profit+=p[i]*x[i];
  rc-=w[i];
 }
 else{
  x[i]=(float)rc/(float)w[i];
  profit+=p[i]*x[i];
  rc=0;
 }
 i++;
}
printf("Profit : %.2f\n",profit);
}
```

OUTPUT:



RESULT:

Thus the program was compiled and executed successfully.