Sort a given N integers using Quick sort technique and compute its time taken.

```
#include <stdio.h>
#include <stdlib.h>
void merge(int low,int mid,int high,int array[20],int mer[20])
{
int i = low;
int j = mid+1;
int k = 0;
while(i<=mid && j<=high)
{
if(array[i]<array[j])
{
mer[k] = array[i];
i++;
k++;
}
else
{
mer[k] = array[j];
j++;
k++;
}
}
while (i <= mid)
{
mer[k] = array[i];
i++;
k++;
}
```

```
while (j <= high)
{
mer[k] = array[j];
j++;
k++;
}
for(int i=0;i<k;i++)
{
array[low+i] = mer[i];
}
}
void merge_sort(int low,int high,int array[20],int merged[20])
{
if(low<high)
{
int mid = (low+high)/2;
merge_sort(low,mid,array,merged);
merge_sort(mid+1,high,array,merged);
merge(low,mid,high,array,merged);
}
}
int main()
{
int n,array[30];
printf("Enter no of elements:");
scanf("%d",&n);
printf("Enter elements:");
for(int i=0;i<n;i++)
{
```

```
scanf("%d",&array[i]);
}
int merged[30];
merge_sort(0,n-1,array,merged);
printf("Sorted array:");
for(int i=0;i<n;i++)
{
printf("%d ",array[i]);
}
}
#include <stdio.h>
void swap(int *a, int *b) {
 int t = *a;
 *a = *b;
 *b = t;
}
int partition(int a[], int l, int h) {
 int pivot = a[l];
 int i = l, j = h;
 while (i < j) {
 while (a[i] <= pivot && i <= h) {
 i++;
 }
 while (a[j] > pivot) {
 j--;
 }
 if (i < j) {
 swap(&a[i], &a[j]);
 }
```

```
 }
 swap(&a[l], &a[j]);
 return j;
}
void quickSort(int a[], int l, int h) {
 if (l < h) {
 int pi = partition(a, l, h);
 quickSort(a, l, pi - 1);
 quickSort(a, pi + 1, h);
 }
}
int main() {
 int a[20], n, i;
 printf("Enter size of array\n");
 scanf("%d", &n);
 printf("Enter data elements: ");
 for (i = 0; i < n; i++) {
 scanf("%d", &a[i]);
 }
 printf("Unsorted Array\n");
 for (i = 0; i < n; i++) {
 printf("%d\t", a[i]);
 }
 quickSort(a, 0, n - 1);
 printf("\nSorted array in ascending order: \n");
 for (i = 0; i < n; i++) {
 printf("%d\t", a[i]);
 }
 return 0;
```

}

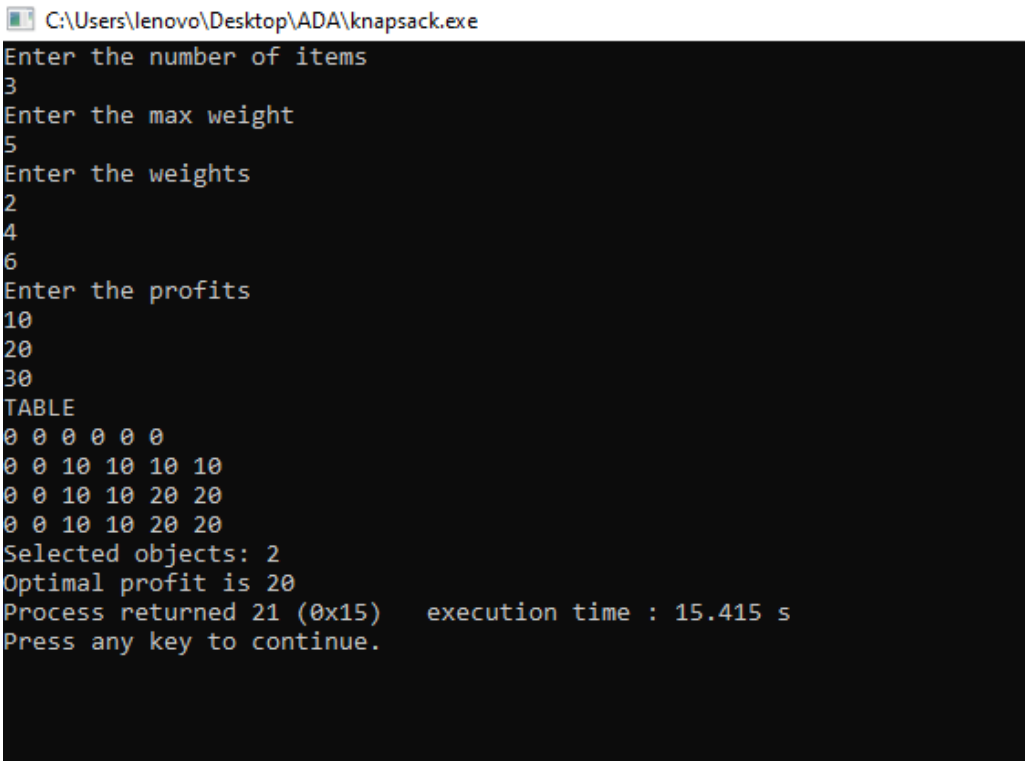
```
Enter size of array
5
Enter data elements: 88 -5 65 -10 0 25 18
Unsorted Array
88      -5      65      -10     0
Sorted array in ascending order:
-10     -5      0       65      88
Process returned 0 (0x0)   execution time : 22.359 s
Press any key to continue.
```

Implement 0/1 Knapsack problem using dynamic programming

```
#include <stdio.h>

#include <stdbool.h>


int p[15],w[15],maxW;

void main(){

   int n,i,j,maxP;

   printf("Enter the number of items\n");

   scanf("%d",&n);

   printf("Enter the max weight\n");

   scanf("%d",&maxW);

   printf("Enter the weights\n");

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

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

   printf("Enter the profits\n");

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

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

   maxP=knapsack(n);

   printf("Optimal profit is %d ",maxP);
```

```
}



int knapsack(int n) {
   int v[n+1][maxW+1],i,j;

   for (int i = 0; i <= n; i++) {
      for (int j = 0; j <= maxW; j++) {
         if (i == 0 || j == 0)
            v[i][j] = 0;
         else if (w[i-1] <= j)
            v[i][j] = max(p[i - 1] + v[i - 1][j - w[i - 1]], v[i - 1][j]);
         else
            v[i][j] = v[i - 1][j];
      }
   }

   int selected[n];
   i = n; j = maxW;
   int count = 0;

   while (i > 0 && j > 0) {
      if (v[i][j] != v[i - 1][j]) {
         selected[count++] = i;
         j -= w[i - 1];
         i--;
      } else {
         i--;
```

```
        }
    }

    printf("TABLE \n");
    for (int i = 0; i <= n; i++) {
        for (int j = 0; j <= maxW; j++) {
            printf("%d ",v[i][j]);
        }
        printf("\n");
    }

    printf("Selected objects: ");
    for (int j = count - 1; j >= 0; j--)
        printf("%d ", selected[j]);
    printf("\n");

    return v[n][maxW];
}



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


C:\Users\lenovo\Desktop\ADA\knapsack.exe
Enter the number of items
3
Enter the max weight
5
Enter the weights
2
4
6
Enter the profits
10
20
30
TABLE
0 0 0 0 0 0
0 0 10 10 10 10
0 0 10 10 20 20
0 0 10 10 20 20
Selected objects: 2
Optimal profit is 20
Process returned 21 (0x15)   execution time : 15.415 s
Press any key to continue.

Sort a given set of N integers using quick sort technique and compute its time taken.

```
#include <stdio.h>
#include <time.h>
void quicksort (int number[25], int first, int last)
{
int i, j, pivot, temp;
if (first < last)
{
pivot = first;
i = first;
j = last;
while (i < j)
{
while (number[i] <= number[pivot] && i < last]
i++;
while (number [j] > number[pivot])
j--;
if (i < j)
{
temp = number[i];
number[i] = number[j];
number[j] = temp;
}
}
temp = number [pivot];
number [pivot] = number [j];
number [j] = temp;
quicksort (number, first, j-1);
quicksort (number, j+1, last);
}
}
int main()
{
```

```
int i, count, number[25];
clock_t start, end;
start = clock();
Printf("Enter no. of elements:");
scanf("%d", &count);
Printf("Enter %d elements:", count);
for(i=0; i<count; i++)
scanf("%d", &number[i]);
quicksort(number, 0, count-1);
end = clock();
printf("Order of sorted elements:");
for(i=0; i<count; i++)
Printf("%d", number[i]);
Printf("\n Time taken to sort is %f", difftime
(end, start) / clocks_per_sec);
return 0;
}
```

Output

Enter the number of elements to be sorted : 5
Enter 5 elements:
10
89
1
58
7
Order of corded elements : 1 7 10 58 89
time taken to sort is : 10.594000.
Process returned 0 (0x0) execution time : 10.594 secs.

Implement 0/1 knapsack problem using dynamic Programming

```
#include<stdio.h>
#include<conio.h>
void knapsack();
int max(int, int);
int i, j, n, m, p[10], w[10], v[10][10];
void main()
{
Printf("Enter no. of items\n");
Scanf("%d", &n);
Printf(" Enter the weight of each item: \n");
for(i=1; i<=n; i++)
{
Scanf("%d", &w[i]);
}
Printf(" Enter the profit of each item \n");
for(i=1; i<=n; i++)
{
Scanf("%d", &p[i]);
}
Printf("\n enter the knapsack's capacity:\t");
Scanf("%d", &m);
knapsack();
getch();
}
void knapsack()
{
int m[10];
for(i=0; i<=n; i++).
{
for(j=0; j<=m; j++)
{
if(i==0||j==0)
{
```

```
        v[i][j]=0;
    }
    else if (j-w[i]<0)
    {
        v[i][j]=v[i-1][j];
    }
    else
    {
    v[i][j]=max(v[i-1][j], v[i-1][j-w[i]]+p[i]);
    }
    }
    }
    Printf("The output is:\n");
    for(i=0; i<=n; i++)
    {
    for(j=0; j<=m; j++)
    {
    Printf("%d\t", v[i][j]);
    }
    Printf("\n\n");
    }
    Printf("\n The optimal solution is %d", v[n][m]);
    printf("\n Solution vector is:\n");
    for(i=n; i>=1; i--)
    {
    if(v[i][m]!=v[i-1][m])
    {
    x[i]=1;
    m=m-w[i];
    }
    else
    {
    x[i]=0
    }
```

```
}
for(i=1; i<=n; i++)
{
Printf("%d\t", x[i]);
}
}
int max (int x, int y)
{
if (x>y)
{
return x;
}
else
{
return y;
}
}
```

Output

```
Enter the no. of item : 3
Enter the weight of each item
2
4
6
enter profit of each item
10
20
30
enter Knap sack capacity : 5
The output is
0  0  0  0  0  0
0  0  10 10 10 10
0  0  10 10 20 20
0  0  10 10 20 20
The optimal solution is 20
The solution vector is 0 1 0.
```