



UNIVERSIDAD NACIONAL ABIERTA
ÁREA DE INGENIERÍA
CARRERA: INGENIERÍA DE SISTEMAS

TRABAJO PRÁCTICO

Asignatura: **Computación II**

Código: **324**

Fecha de Entrega al Estudiante: **Al inicio del lapso académico 2022-2.**

Fecha de Devolución Informe por parte del Estudiante: **22/10/2022**

Nombre del Estudiante: **David Alfonso Olivo Rodríguez**

Cédula de Identidad: **29.959.060**

Correo Electrónico del Estudiante: davidoar15@gmail.com

Teléfono Celular: **0414-0529486**

Centro Local: **Centro Local Aragua**

Carrera: **236**

Lapso Académico: **2022 – 2**

Número de Originales: **1**

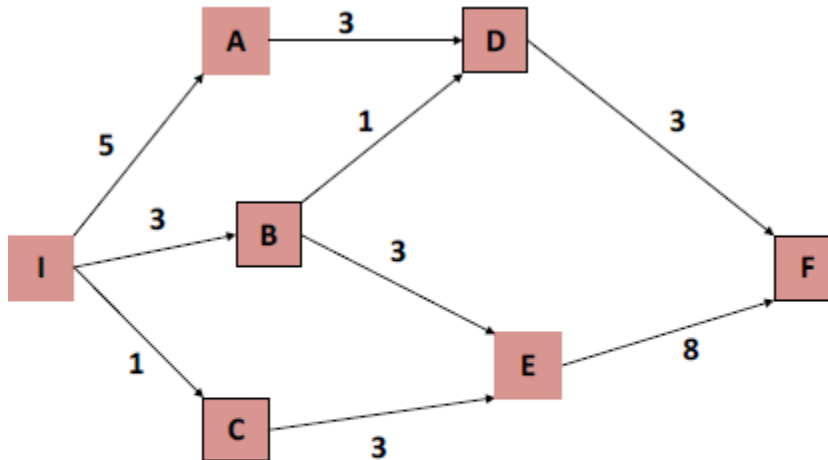
Resultados de Corrección

OBJ N°		5	6	7
0: NL	1: L			

Cuerpo del Trabajo

Objetivo 5:

1- El Grafo que se presenta a continuación representa las conexiones de gas existentes entre algunos de los diferentes almacenes de la empresa SUPRAGAS. Sobre cada arco, se indica la capacidad máxima de transporte de GAS (en m³/horas) de las tuberías que conectan los almacenes.



Fuente (Suministrador) = I
Sumidero (Colector) = F

Se Pide:

Utilizando el Algoritmo de Ford – Fulkerson

- ✓ Elabore un programa en C++ que calcule la máxima cantidad de gas que puede almacenarse en el Colector F.
- ✓ Determinar qué flujo debe circular por cada una de las tuberías para obtener el flujo máximo.

Respuesta:

- Código del Programa en C++:

```
#include<iostream>
```

```
#include<limits.h>
```

```
#include<string.h>
```

```

#include<queue>
#include<stdlib.h>
using namespace std;

// Se definen el Número de Vértices (Almacenes) del grafo
#define V 7

bool bfs(int grafoRes[V][V], int i, int f, int cont[ ]){
    bool visitado[V];
    memset(visitado, 0, sizeof(visitado));

    queue <int> q;
    q.push(i);
    visitado[i] = true;
    cont[i] = -1;

    while(!q.empty()){
        int u = q.front();
        q.pop();

        for(int v=0; v<V; v++){
            if(visitado[v] == false && grafoRes[u][v] > 0){
                q.push(v);
                cont[v] = u;
                visitado[v] = true;
            }
        }
    }

    return(visitado[f] == true);
}

```

/* Función del Algoritmo Ford – Fulkerson: Retorna el flujo máximo desde i (inicio) hasta f (final) en el grafo dado */

```
int fordFulkerson(int grafo[V][V], int i, int f){
    int u, v;
    int grafoRes[V][V];

    for(u=0; u<V; u++){
        for(v=0; v<V; v++){
            grafoRes[u][v] = grafo[u][v];
        }
    }

    int cont[V];
    int flujoMax = 0;
    while (bfs(grafoRes, i, f, cont)){
        int rutaFlujo = INT_MAX;

        for(v=f; v!=i; v=cont[v]){
            u = cont[v];
            rutaFlujo = min(rutaFlujo, grafoRes[u][v]);
        }

        for(v=f; v!=i; v=cont[v]){
            u = cont[v];
            grafoRes[u][v] -= rutaFlujo;
            grafoRes[v][u] += rutaFlujo;
        }

        flujoMax += rutaFlujo;
    }
    return flujoMax;
}
```

```

int main(){
    // Se insertan los datos correspondientes del grafo
    int grafo[V][V] = {
        {0, 5, 3, 1, 0, 0, 0}, // Almacén I para Almacenes: A, B y C. (INICIO)
        {0, 0, 0, 0, 3, 0, 0}, // Almacén A para Almacén D.
        {0, 0, 0, 0, 1, 3, 0}, // Almacén B para Almacenes: D y E.
        {0, 0, 0, 0, 0, 3, 0}, // Almacén C para Almacén E.
        {0, 0, 0, 0, 0, 0, 3}, // Almacén D para Almacén F.
        {0, 0, 0, 0, 0, 0, 8}, // Almacén E para Almacén F
        {0, 0, 0, 0, 0, 0, 0} // Almacén F. (FINAL)
    };

    /* Como si fuera una matriz, cada Columna representa la capacidad de transporte que tiene
    cada vértice a otro; y cada Fila representa al vértice o almacén correspondiente. Ejemplo:

    {0, 5, 3, 1, 0, 0, 0} es el almacén I (inicio), que tiene tuberías de: capacidad 5 m³/horas hacia
    almacén A, capacidad 3 m³/horas hacia almacén B y capacidad 1 m³/horas hacia almacén
    C. Obviamente, tiene capacidad 0 para sí mismo y para los almacenes D, E y F, ya que no
    existen tuberías entre ellos.

    {0, 0, 0, 0, 3, 0, 0} A: tubería de capacidad 3 m³/horas hacia D.

    {0, 0, 0, 0, 1, 3, 0} B: tuberías de: capacidad 1 m³/horas hacia D y capacidad 3 m³/horas
    hacia E.

    {0, 0, 0, 0, 0, 3, 0} C: tubería de capacidad 3 m³/horas hacia E.

    {0, 0, 0, 0, 0, 0, 3} D: tubería de capacidad 3 m³/horas hacia F (final).

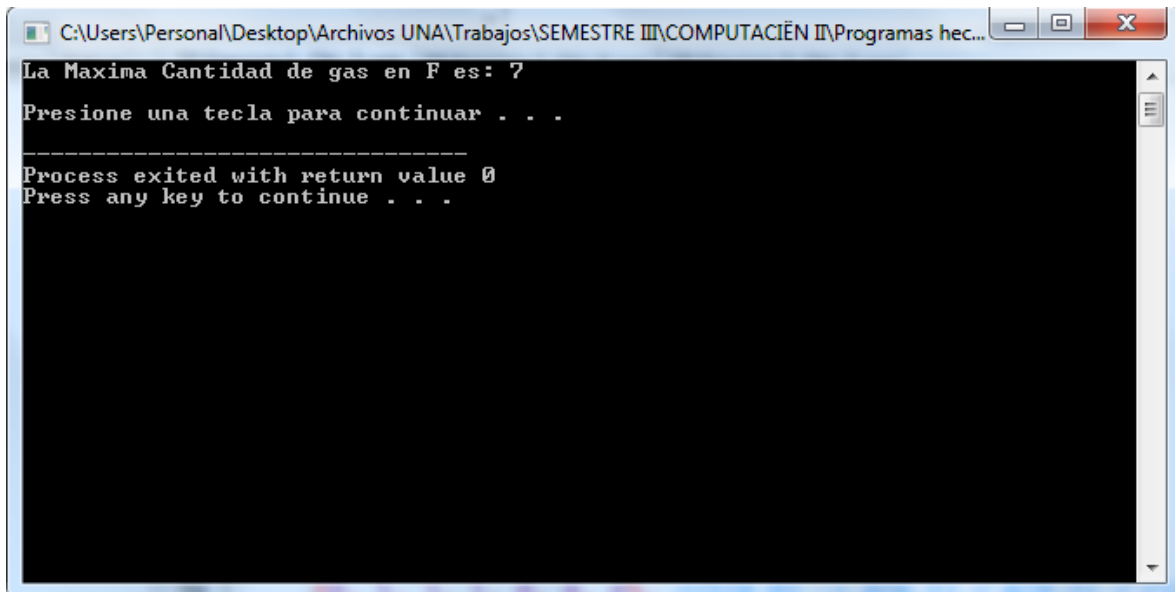
    {0, 0, 0, 0, 0, 0, 8} E: tubería de capacidad 8 m³/horas hacia F (final). */

    cout<<"La Maxima Cantidad de gas en F es: "<<fordFulkerson(grafo, 0, 6)<<"\n";

    cout<<"\n";
    system("pause");
    return 0;
}

```

- Prueba de Funcionamiento:



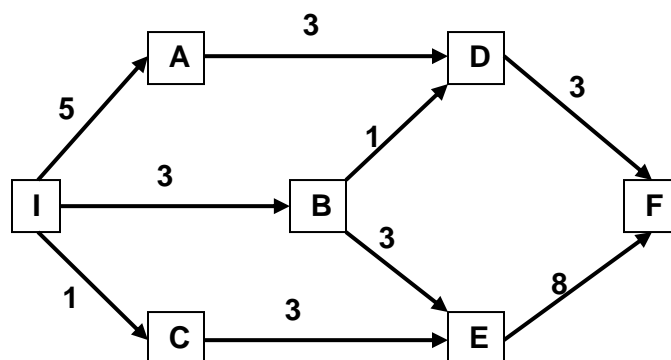
```
C:\Users\Personal\Desktop\Archivos UNA\Trabajos\SEMESTRE III\COMPUTACIÓN II\Programas hec...
La Maxima Cantidad de gas en F es: 7
Presione una tecla para continuar . . .
-----
Process exited with return value 0
Press any key to continue . . .
```

- **Determinación del Flujo en cada Tubería:**

El Algoritmo o Método Ford – Fulkerson, de forma práctica, funciona de la siguiente manera:

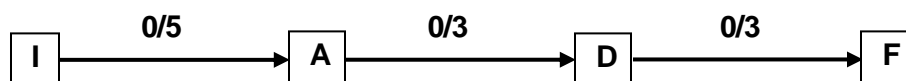
Para buscar el Flujo Máximo desde un punto inicial a un punto final, dentro de un mapa de caminos (como lo es el grafo presentado), se deben seleccionar las posibles rutas y en cada una seleccionar el flujo mínimo que puede pasar por la ruta seleccionada. En este caso, el análisis es el siguiente:

- Grafo:

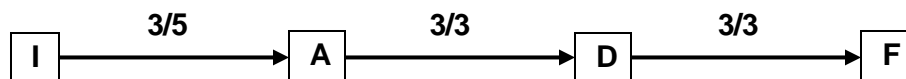


Entonces, las rutas seleccionadas son las siguientes:

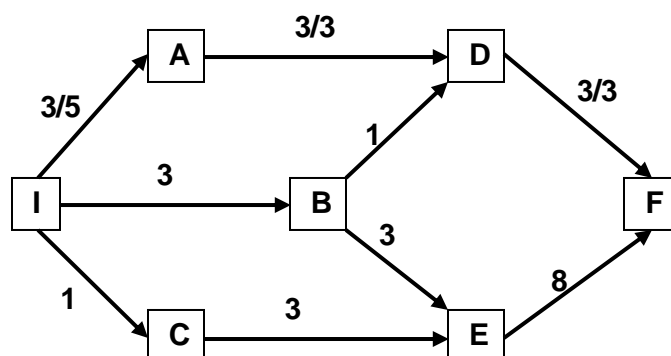
Ruta 1:



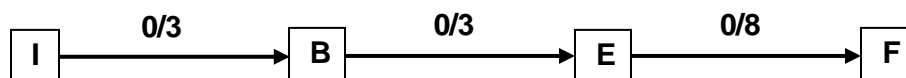
El flujo mínimo en esta ruta es 3. Ahora la Ruta 1 queda así:



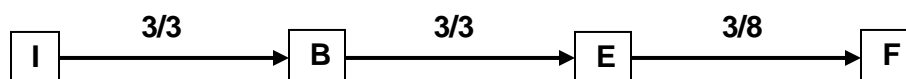
Desde I hasta A aún hay capacidad, pero las tuberías desde A hasta D y desde D hasta F ya están saturadas, por lo que ya no es posible pasar por ellas. El grafo ahora queda así:



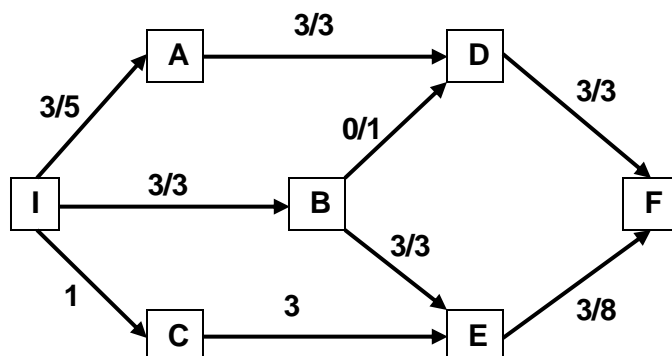
Ruta 2:



Nuevamente, el flujo mínimo es 3. La Ruta 2 queda así:

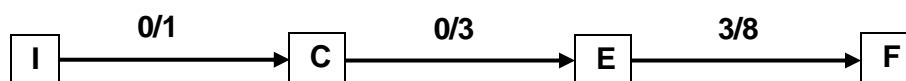


Desde E hasta F aún hay capacidad, pero las tuberías desde I hasta B y desde B hasta E ya están saturadas, por lo que ya no es posible pasar por ellas. El grafo ahora queda así:

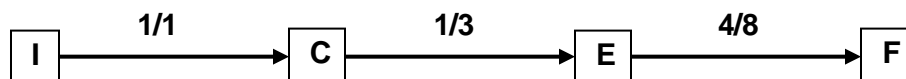


Como se puede observar, no hay flujo en la tubería que conecta B y D, ya que la tubería desde D hasta F ya está saturada y ocurriría una sobresaturación.

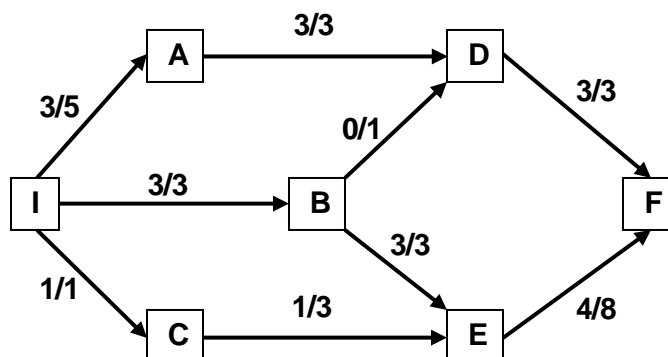
Ruta 3:



El flujo mínimo es 1. La Ruta 3 queda así:



El grafo ahora queda así:



Con este grafo, se puede saber qué flujo debe circular por cada una de las tuberías para obtener el flujo máximo:

- Desde I hasta A debe pasar un flujo de **3**.
- Desde A hasta D debe pasar un flujo de **3**.
- Desde D hasta F debe pasar un flujo de **3**.
- Desde I hasta B debe pasar un flujo de **3**.
- Desde B hasta D no hay flujo (flujo de **0**).
- Desde B hasta E debe pasar un flujo de **3**.
- Desde I hasta C debe pasar un flujo de **1**.
- Desde C hasta E debe pasar un flujo de **1**.
- Desde E hasta F debe pasar un flujo de **4**.

Además, el Flujo Máximo es la suma de los flujos mínimos en cada ruta seleccionada:

$$\text{Flujo Máximo} = 3 + 3 + 1 = 7$$

Siendo este valor la cantidad máxima de gas que se puede almacenar en el almacén F.

Objetivo 6:

2- Utilizando el método de ordenamiento por quicksort y dado un vector entero de 30 elementos, los cuales son introducidos de forma aleatoria, ordenar dichos números enteros contenidos en él array.

Respuesta:

- **Código del Programa en C++:**

```
#include<iostream>
```

```
#include<stdlib.h>
```

```
using namespace std;
```

// Prototipos de Funciones:

```
void quicksort(float [ ], int, int);
```

```
void escribir(float [ ], int);
```

```
int main(){
```

```
int n;
```

```
cout<<"Inserte el numero de elementos que tendra el array/arreglo: "; cin>>n;
```

```
float numeros[n];
```

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

```
cout<<i+1<<". Inserte un numero cualquiera: "; cin>>numeros[i];
```

}

```
cout<<"\n";
```

```
cout<<"Array Ingresado: "<<endl;
```

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

```
cout<<"|"<<numeros[i]<<"| ";
```

}

```

cout<<endl;
quicksort(numeros, 0, n-1);
escribir(numeros, n);

cout<<"\n";
system("pause");
return 0;
}

// Función Ordenamiento QuickSort:
void quicksort(float nros[ ], int first, int last){
    int mitad, i, j;
    float pivot;

    mitad = (first + last)/2;
    pivot = nros[mitad];
    i = first;
    j = last;

    do{
        while(nros[i] < pivot){
            i++;
        }
        while(nros[j] > pivot){
            j--;
        }

        if(i <= j){
            float aux;
            aux = nros[i];
            nros[i] = nros[j];
            nros[j] = aux;

```

```

        i++; j--;
    }
} while(i <= j);

if(first < j){
    quicksort(nros, first, j);
}
if(i < last){
    quicksort(nros, i, last);
}
}

void escribir(float nros[ ], int x){

    cout<<"\nArray Resultante (Ordenado por Metodo QuickSort): "<<endl;
    for(int i = 0; i < x; i++){
        cout<<"| "<<nros[i]<<"| ";
    }
    cout<<endl;
}

```

- Prueba de Funcionamiento:

```
C:\Users\Personal\Desktop\Archivos UNA\Trabajos\SEMESTRE III\COMPUTACI3N II\Programas hec...
Inserte el numero de elementos que tendra el array/arreglo: 30
1. Inserte un numero cualquiera: 2.3
2. Inserte un numero cualquiera: -4.7
3. Inserte un numero cualquiera: 19
4. Inserte un numero cualquiera: 1.41
5. Inserte un numero cualquiera: 26
6. Inserte un numero cualquiera: 3
7. Inserte un numero cualquiera: -1
8. Inserte un numero cualquiera: 0
9. Inserte un numero cualquiera: 79
10. Inserte un numero cualquiera: 2
11. Inserte un numero cualquiera: 2
12. Inserte un numero cualquiera: 47
13. Inserte un numero cualquiera: 12
14. Inserte un numero cualquiera: -19
15. Inserte un numero cualquiera: 9
16. Inserte un numero cualquiera: 34
17. Inserte un numero cualquiera: -20
18. Inserte un numero cualquiera: 67
19. Inserte un numero cualquiera: 112
20. Inserte un numero cualquiera: -112
21. Inserte un numero cualquiera: 75
22. Inserte un numero cualquiera: -4.95
23. Inserte un numero cualquiera: 17
24. Inserte un numero cualquiera: 90
```

```
C:\Users\Personal\Desktop\Archivos UNA\Trabajos\SEMESTRE III\COMPUTACI3N II\Programas hec...
20. Inserte un numero cualquiera: -112
21. Inserte un numero cualquiera: 75
22. Inserte un numero cualquiera: -4.95
23. Inserte un numero cualquiera: 17
24. Inserte un numero cualquiera: 90
25. Inserte un numero cualquiera: 31
26. Inserte un numero cualquiera: 55
27. Inserte un numero cualquiera: 35
28. Inserte un numero cualquiera: 67
29. Inserte un numero cualquiera: 100
30. Inserte un numero cualquiera: 99

Array Ingresado:
12.3! -4.7! 19! 1.41! 26! 3! -1! 0! 79! 2! 2! 47! 12! -19! 9! 34!
-20! 67! 112! -112! 75! -4.95! 17! 90! 31! 55! 35! 67! 100! 99!

Array Resultante (Ordenado por Metodo QuickSort):
-112! -20! -19! -4.95! -4.7! -1! 0! 1.41! 2! 2! 2.3! 3! 9! 12! 17
! 19! 26! 31! 34! 35! 47! 55! 67! 67! 75! 79! 90! 99! 100! 112!

Presione una tecla para continuar . . .

Process exited with return value 0
Press any key to continue . . .
```

Objetivo 7:

3- Realice un Programa en C++ que implemente la búsqueda binaria de un elemento determinado dentro de un array (ordenado en formato descendente) de N elementos.

Respuesta:

- **Código del Programa en C++:**

```
#include<iostream>
```

```
#include<stdlib.h>
```

```
using namespace std;
```

```
// Prototipos de Funciones:
```

```
void quicksort(float [ ], int, int);
```

```
void escribir(float [ ], int);
```

```
void busBinaria(float [ ], int, int);
```

```
int main(){
```

```
    int n;
```

```
    cout<<"Inserte el numero de elementos que tendra el array/arreglo: "; cin>>n;
```

```
    float Elementos[n];
```

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

```
        cout<<i+1<<" Inserte un numero cualquiera: "; cin>>Elementos[i];
```

```
    }
```

```
    cout<<"\n";
```

```
    quicksort(Elementos, 0, n-1);
```

```
    busBinaria(Elementos, 0, n-1);
```

```
    escribir(Elementos, n);
```

```

cout<<"\n";
cout<<"Gracias. Feliz dia."<<endl;

cout<<"\n";
system("pause");
return 0;
}

// Función Ordenamiento QuickSort Inverso:
void quicksort(float elementos[ ], int first, int last){
    int mitad, i, j;
    float pivot;

    mitad = (first + last)/2;
    pivot = elementos[mitad];
    i = first;
    j = last;

    do{
        while(elementos[i] > pivot){
            i++;
        }
        while(elementos[j] < pivot){
            j--;
        }

        if(i <= j){
            float aux;
            aux = elementos[i];
            elementos[i] = elementos[j];
            elementos[j] = aux;
            i++; j--;
        }
    } while(i < j);
}

```

```

    }
}while(i <= j);

if(first < j){
    quicksort(elementos, first, j);
}
if(i < last){
    quicksort(elementos, i, last);
}
}

```

```

void escribir(float elementos[ ], int x){

    cout<<"\nArray Ordenado (de forma Descendente): "<<endl;
    cout<<"\n";
    for(int i = 0; i < x; i++){
        cout<<"| "<<elementos[i]<<"| ";
    }
    cout<<endl;
}

```

// Función de Búsqueda Binaria:

```

void busBinaria(float elementos[ ], int first, int last){
    int a = 0, z = 0, mitad, dato;
    char BOOL = 'F';
    a = first; z = last+1;

    cout<<"\nInserte el elemento que desea buscar: "; cin>>dato;

    while(a <= z){
        mitad = (a+z)/2;

```

```

        if(elementos[mitad] == dato){
            BOOL = 'V';
            break;
        }
        if(elementos[mitad] < dato){
            z = mitad-1;
            mitad = (a+z)/2;
        }
        if(elementos[mitad] > dato){
            a = mitad+1;
            mitad = (a+z)/2;
        }
    }

    if(BOOL == 'V'){
        cout<<"\nEl elemento "<<dato<<" ha sido encontrado en la posicion: "<<mitad+1<<endl;
    }
    else{
        cout<<"\n";
        cout<<"\nEl elemento ingresado NO ha sido encontrado"<<endl;
    }

    system("pause");
}

```


- Prueba de Funcionamiento:

```
C:\Users\Personal\Desktop\Archivos UNA\Trabajos\SEMESTRE III\COMPUTACI3N II\Programas hec...
Inserte el numero de elementos que tendra el array/arreglo: 10
1. Inserte un numero cualquiera: 5
2. Inserte un numero cualquiera: 3.2
3. Inserte un numero cualquiera: -6
4. Inserte un numero cualquiera: -1.2
5. Inserte un numero cualquiera: 55
6. Inserte un numero cualquiera: 100
7. Inserte un numero cualquiera: 4
8. Inserte un numero cualquiera: 0
9. Inserte un numero cualquiera: 19
10. Inserte un numero cualquiera: -5

Inserte el elemento que desea buscar: -6

El elemento -6 ha sido encontrado en la posicion: 10
Presione una tecla para continuar . . .

Array Ordenado (de forma Descendente):

100 55 19 5 4 3.2 0 -1.2 -5 -6

Gracias. Feliz dia.
Presione una tecla para continuar . . .
```

```
C:\Users\Personal\Desktop\Archivos UNA\Trabajos\SEMESTRE III\COMPUTACI3N II\Programas hec...
Inserte el numero de elementos que tendra el array/arreglo: 10
1. Inserte un numero cualquiera: -3
2. Inserte un numero cualquiera: 55
3. Inserte un numero cualquiera: 77
4. Inserte un numero cualquiera: -2
5. Inserte un numero cualquiera: 0
6. Inserte un numero cualquiera: 11
7. Inserte un numero cualquiera: 23
8. Inserte un numero cualquiera: -26.7
9. Inserte un numero cualquiera: 10
10. Inserte un numero cualquiera: 88

Inserte el elemento que desea buscar: 1

El elemento ingresado NO ha sido encontrado
Presione una tecla para continuar . . .

Array Ordenado (de forma Descendente):

88 77 55 23 11 10 0 -2 -3 -26.7

Gracias. Feliz dia.
```

FIN DEL TRABAJO PRÁCTICO