Universidad EAFIT

Estructura de datos y algoritmos I

Documentación Parcial 1 – Momento 2

Ismael García Ceballos

Carlos Alberto Álvarez Henao

Agosto 18 de 2025

Introducción:

Para este proyecto se requirió de:

- Instalación y configuración de GIT: https://git-scm.com/downloads
- Instalación del compilador g++: https://www.msys2.org/
- Clonación del repositorio de GITHUB mediante GIT CMD a través de la ruta y la URL: https://github.com/IsmaelGC24/Parcial1-EDA1.git

Punto 1: Inversión de arreglos

Enfoque utilizado:

Para resolver el problema, se implementaron dos funciones:

- **invertirArreglo**: Recibe un arreglo (que en este caso es un vector) y genera uno nuevo con los elementos en orden inverso. Se recorre el arreglo original de atrás hacia adelante y se copian los valores en el nuevo vector.
- **mostrarArreglo**: Recorre el arreglo y muestra sus elementos en consola, separados por -->.

El enfoque sigue la lógica propuesta en el pseudocódigo de la guía, pero adaptado a C++ usando la librería "vector".

Dificultades encontradas:

- Inicialmente, la función mostrarArreglo asumía que el vector siempre tenía al menos un elemento, lo que causaba un error si se agregaba un caso donde el arreglo fuera un vector vacío, lo cual se podría solucionar con un condicional if (arr.empty()) que verifique el tamaño del arreglo, además de añadir el caso en el main con un mensaje especial si ocurre.
- También se cuidó el formato de salida para que fuera legible en los diferentes casos de prueba.

Estrategia de pruebas:

Se plantearon los siguientes casos de prueba para comprobar el funcionamiento del algoritmo:

1. **Caso normal:** Arreglo con varios elementos distintos.
   Este fue el único que se proponía en el ejercicio, por lo cual es el único contenido en el código fuente, sin embargo, otros casos fueron probados en una versión más extensa del código.
   Se pedía un arreglo de al menos 5 elementos, por lo que se creó uno de 10 que incluye números positivos, negativos, repetidos y el 0.

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\Ismael\OneDrive\Documentos\Repositories\Parcial1-EDA1\Punto1> g++ -o main.exe ./main.cpp
PS C:\Users\Ismael\OneDrive\Documentos\Repositories\Parcial1-EDA1\Punto1> ./main.exe
Arreglo original:
3 --> 8 --> 15 --> 42 --> 7 --> -23 --> 11 --> 99 --> 15 --> 0
Arreglo invertido:
0 --> 15 --> 99 --> 11 --> -23 --> 7 --> 42 --> 15 --> 8 --> 3
PS C:\Users\Ismael\OneDrive\Documentos\Repositories\Parcial1-EDA1\Punto1>
```

2. **Caso unitario:** arreglo con un solo elemento.
3. **Caso repetidos:** arreglo donde todos los elementos son iguales.

Para estos dos casos las modificaciones respecto al código original fueron:

```cpp
    // Caso 2: Arreglo con un solo elemento
    vector<int> uno = {42};
    cout << "\nCaso 2: Arreglo con un solo elemento" << endl;
    cout << "Arreglo original:" << endl;
    mostrarArreglo(uno);
    vector<int> invertidoUno = invertirArreglo(uno);
    cout << "Arreglo invertido:" << endl;
    mostrarArreglo(invertidoUno);

    // Caso 3: Arreglo con elementos repetidos
    vector<int> repetidos = {7, 7, 7, 7};
    cout << "\nCaso 3: Arreglo con elementos repetidos" << endl;
    cout << "Arreglo original:" << endl;
    mostrarArreglo(repetidos);
    vector<int> invertidoRepetidos = invertirArreglo(repetidos);
    cout << "Arreglo invertido:" << endl;
    mostrarArreglo(invertidoRepetidos);

    return 0;
}
```

Mostrando la siguiente salida en consola:

```
PS C:\Users\Ismael\OneDrive\Documentos\Repositories\Parcial1-EDA1\Punto1> g++ -o main.exe ./main.cpp
PS C:\Users\Ismael\OneDrive\Documentos\Repositories\Parcial1-EDA1\Punto1> ./main.exe
Caso 1: Arreglo normal
Arreglo original:
3 --> 8 --> 15 --> 42 --> 7 --> -23 --> 11 --> 99 --> 15 --> 0
Arreglo invertido:
0 --> 15 --> 99 --> 11 --> -23 --> 7 --> 42 --> 15 --> 8 --> 3

Caso 2: Arreglo con un solo elemento
Arreglo original:
42
Arreglo invertido:
42

Caso 3: Arreglo con elementos repetidos
Arreglo original:
7 --> 7 --> 7 --> 7
Arreglo invertido:
7 --> 7 --> 7 --> 7
PS C:\Users\Ismael\OneDrive\Documentos\Repositories\Parcial1-EDA1\Punto1>
```

Resultados obtenidos:

- En todos los casos, el programa mostró correctamente el arreglo original e invertido.
- Con un solo elemento, el arreglo invertido es idéntico.
- Con valores repetidos, el orden se mantiene correcto.
- En el caso vacío (si se incluye), el programa imprime un mensaje indicando que está vacío.

Punto 2: Inserción ordenada en una lista enlazada

Enfoque utilizado:

```c
C/C++

#include <stdio.h>
#include <stdlib.h>

struct Node {
    int data;
    struct Node* next;
};

void insertAtHead(struct Node** head_ref, int new_data) {
    struct Node* new_node = (struct Node*) malloc(sizeof(struct Node));
    new_node->data = new_data;
    new_node->next = (*head_ref);
    (*head_ref) = new_node;
}

int main() {
    struct Node* head = NULL;
    insertAtHead(&head, 1);
    printf("Data in head node is %d\n", head->data);
    return 0;
}
```

Tomado de:
https://github.com/carlosalvarezh/EstructuraDatosAlgoritmos1/blob/main/S04S05_EDLineales_ListasEnlazadas.ipynb

Se tomó como base la anterior estructura de creación de un nodo, y se modifico considerablemente para adaptarlo al ejercicio pedido.

Se definió una estructura Nodo con dos campos: un entero dato y un puntero siguiente. Y se crearon las siguientes funciones:

- **insertarOrdenado**: Recibe la cabeza de la lista y un nuevo valor, y crea un nodo nuevo insertándolo en la posición correcta para mantener el orden ascendente. Se contemplan los tres casos:
    1. Insertar al inicio.
    2. Insertar en medio.
    3. Insertar al final.
- **mostrarLista**: Recorre la lista imprimiendo cada valor seguido de -->, finalizando en NULL.
- **liberarLista**: Recorre la lista y libera la memoria de todos los nodos para evitar fugas.

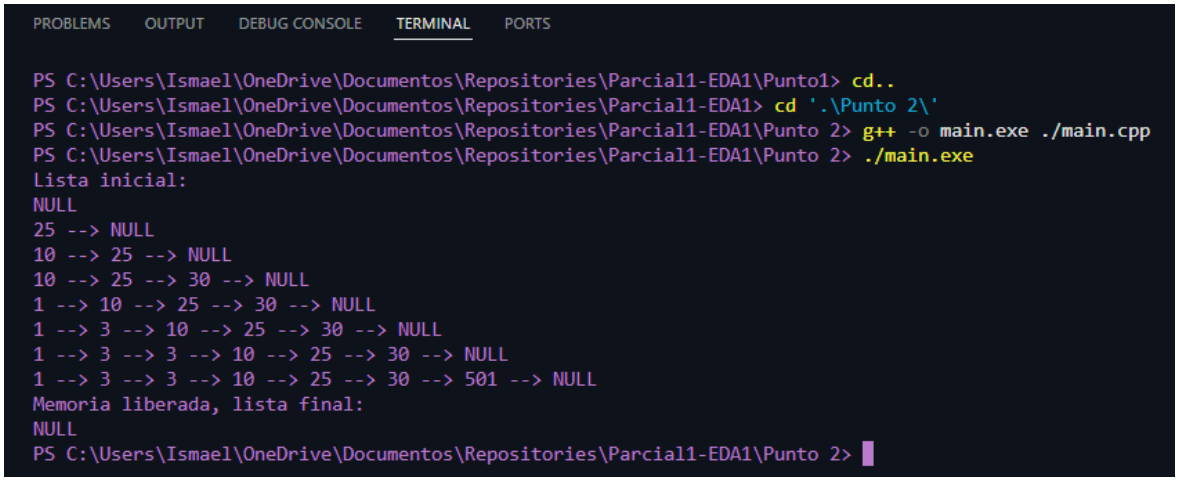
Dificultades encontradas:

- La principal dificultad fue garantizar que la inserción funcionara correctamente en los tres casos (inicio, medio, final).
- Otro aspecto fue considerar la inserción de valores repetidos, que también debía insertarse respetando el orden.
- Finalmente, se añadió una función para liberar memoria, ya que cada nodo se crea dinámicamente con new, y según lo visto en clase es buena práctica (por no decir necesario) utilizar delete para estructuras generadas con new o malloc, para así evitar memory leaks.

Estrategias de prueba:

Se probaron los siguientes escenarios:

1. **Lista vacía:** inserción del primer nodo.
2. **Inserción al inicio:** un valor menor que todos los existentes.
3. **Inserción en medio:** un valor intermedio.
4. **Inserción al final:** un valor mayor que todos los existentes.
5. **Inserción de valores repetidos:** insertar un número ya existente en la lista.



Resultados obtenidos:

- La lista se mantuvo siempre ordenada después de cada inserción.
- La inserción de valores repetidos funcionó correctamente.
- La función mostrarLista presentó la lista en el formato esperado (dato --> ... --> NULL).
- La función liberarLista liberó toda la memoria al final, dejando la lista vacía y evitando fugas de memoria.