

Algoritmos de Búsqueda y Ordenamiento

Descripción act1.2

En esta actividad se utilizan 3 tipos de algoritmos de ordenamiento (Bubble Sort, Ordenamiento por Intercambio y Merge Sort). También se utilizan 2 tipos de algoritmos de búsqueda (Binary Search y Búsqueda Secuencial) en vectores ordenados por los algoritmos de ordenamiento.

Complejidad de algoritmos de ordenamiento:

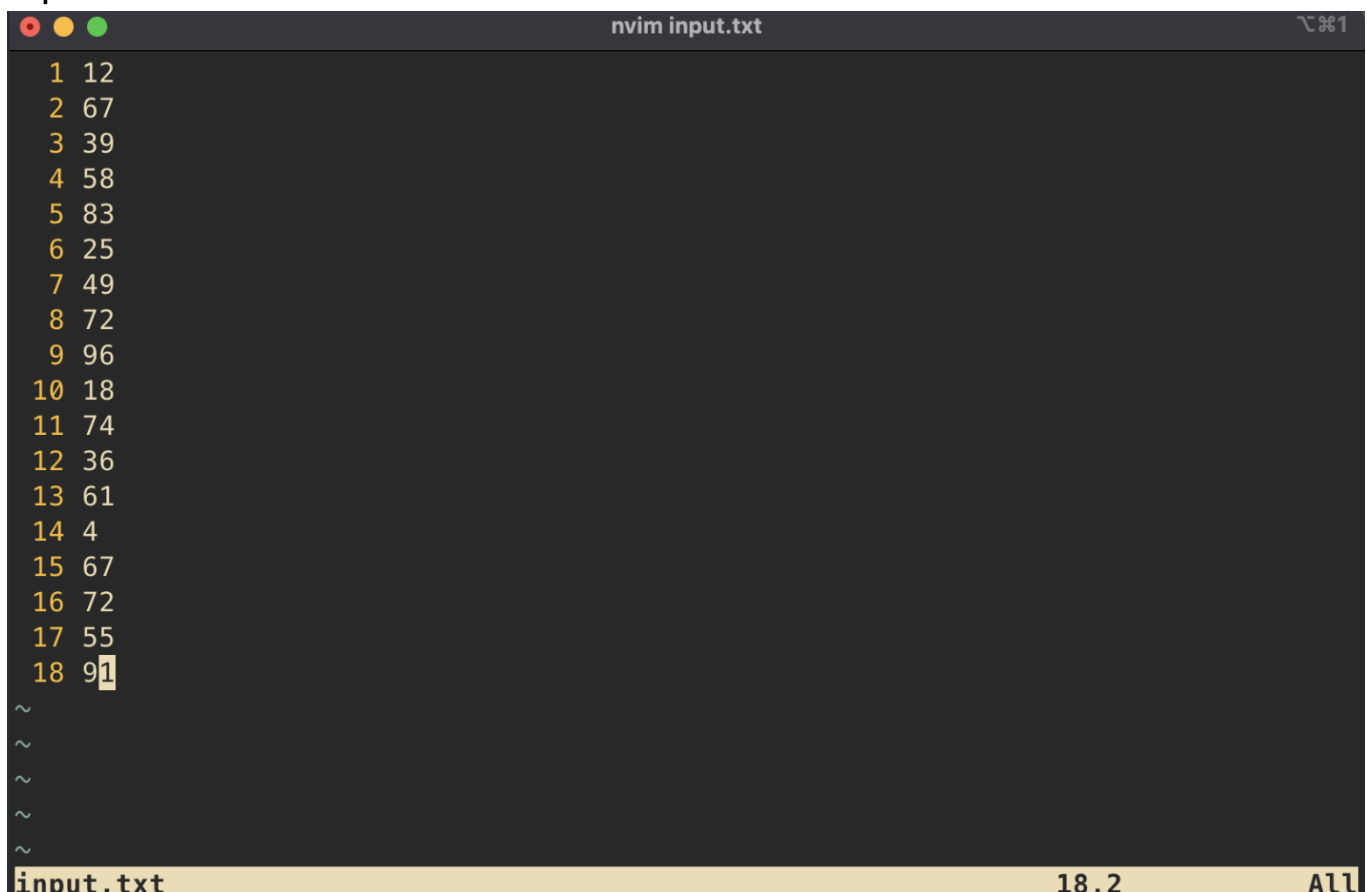
- *Bubble Sort*: Este algoritmo cuenta con una complejidad de $O(n^2)$ ya que cuenta con dos fors anidados. El primer bucle recorre todo el vector mientras que el segundo for se ejecuta con respecto a la cantidad de elementos n del primer ciclo. Entonces tenemos que ambas iteraciones dependen del tamaño n del vector, en el peor de los casos se harían n^2 comparaciones.
- *Ordenamiento de intercambio*: Cuenta con una complejidad de $O(n^2)$. Ya que como bubble sort, este tiene dos ciclos for anidados, lo que causa n^2 comparaciones. De igual manera el primer bucle recorre todo el vector mientras que el segundo se ejecuta con relación a la cantidad de elementos.
- *Merge sort*: En este algoritmo tenemos una mejora ya que su complejidad temporal es de $O(n \log(n))$, en el cual n representa el número de elementos en el vector a ordenar. La recursión en el algoritmo se divide en $\log(n)$ niveles y en cada nivel se realizan n comparaciones en la fusión de mitades por lo tanto por cada $\log(n)$ como formas de divisiones existe una n para comparar esa fusión de mitades.

Complejidad de algoritmos de búsqueda:

- *Búsqueda binaria*: Este algoritmo tiene una complejidad temporal de $O(\log(n))$ donde n representa el número de elementos dentro del vector. Este, divide repetidamente el espacio de búsqueda por la mitad, cada vez reduciendo el número de elementos por iteración lo cual básicamente se traduce a $\log(n)$. Por cada iteración tendremos reducción por mitades lo cual hace que sea bastante eficiente para buscar elementos en vectores arreglados.
- *Búsqueda Secuencial*: En este algoritmo tenemos una complejidad de $O(n)$ ya que contamos con solo un ciclo for dependiente de la cantidad de elementos del vector n , este for itera sobre cada elemento del vector donde en el peor de los casos el número a buscar se encontrará al final del vector lo que resulta en una complejidad lineal $O(n)$.

Casos prueba

Input



The image shows a terminal window titled 'nvim input.txt'. It contains a list of 18 numbers, each preceded by a line number from 1 to 18. The numbers are: 12, 67, 39, 58, 83, 25, 49, 72, 96, 18, 74, 36, 61, 4, 67, 72, 55, and 91. The cursor is positioned at the end of the 18th line. The terminal has a dark background with light-colored text. The status bar at the bottom shows 'input.txt', '18.2', and 'ALL'.

```
1 12
2 67
3 39
4 58
5 83
6 25
7 49
8 72
9 96
10 18
11 74
12 36
13 61
14 4
15 67
16 72
17 55
18 91
~
~
~
~
~
```

input.txt 18.2 ALL

Output

nvim output.txt

```
1 66 63 32
2 7 8 4
3 8 9 2
4 -1 12 4
5 -1 12 4
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
```

Main.cpp

```
int comparacionesGlobales = 0;

int main(){
    // Primer lectura para agrarrar los numeros que se leeran
    std::ifstream inputFile("input.txt");
    int numElementosOrdenar; // Cantidad de elementos a ordenar
    inputFile >> numElementosOrdenar;
    //std::cout << "El número de elementos a ordenar es: " << numElementosOrdenar << std::endl;

    // Lectura de elemntos del vector de numeros a ordenar
    std::vector<int> numbers;

    for (int i = 0; i < numElementosOrdenar; i++){
        int number;
        if (inputFile >> number){
            numbers.push_back(number);
        }
    }
    // vector reasignado para bubble Sort

    vector<int> prueba = numbers;

    // Elementos a buscar
    int numElementosBuscar;
    inputFile >> numElementosBuscar;
    // Lectura de elemntos del vector a buscar
    std::vector<int> numbersToSearch;
    for (int i = 0; i < numElementosBuscar; i++) {
        int number;
        if (inputFile >> number){
            numbersToSearch.push_back(number);
        }
    }

    //vector ordenado con bubble
    std::vector<int> sortedNumbers = bubbleSortVector(numbers);

    // Apertura de archivo output.txt
    std::ofstream outputFile("output.txt");

    if (outputFile.is_open()) {
        // Comparaciones de ordenamiento
        // comparaciones de Intercambio, comparaciones de Bubble, comparaciones merge sort
        int contadores = 0;
        ordenaMerge(numbers, 0, numbers.size() - 1, contadores);
        outputFile << ordenamientoIntercambio(numbers) << " " << bubbleSort(prueba, prueba.size()) << " " << contadores << std::endl;
        for (int i = 0; i < numbersToSearch.size(); i++) {
            int elementToSearch = numbersToSearch[i];
            int indexes = busquedaSecuencial(sortedNumbers, elementToSearch);
            int comparacionesBinarias = busquedaBin(sortedNumbers, elementToSearch);

            // Indexes, comparaciones con buesqueda secuencial, comparaciones con busqueda binaria.
            outputFile << indexes << " " << comparacionesGlobales << " " << comparacionesBinarias << std::endl;

            comparacionesGlobales = 0;
        }
        std::cout << "Los resultados se han guardado en output.txt " << std::endl;
        outputFile.close();
    } else {
        std::cerr << "Unable to open output file." << std::endl;
    }
}
```