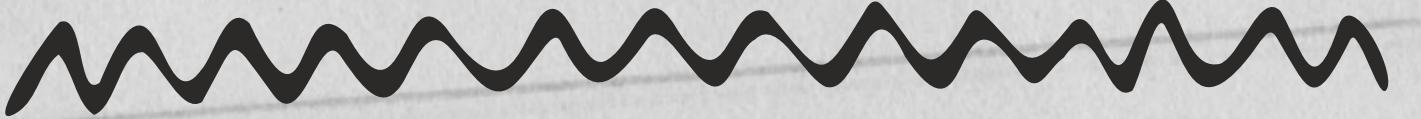
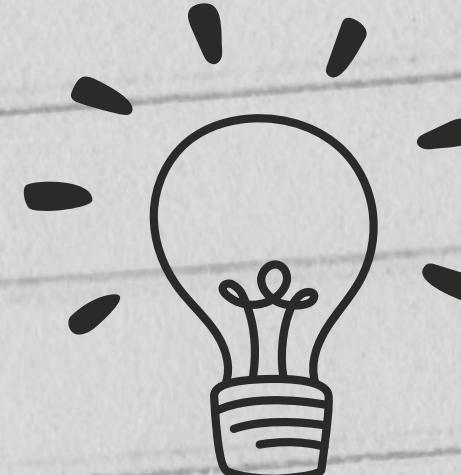


EVIDENCIA **ACTIVIDAD 1.3**



INTEGRANTES:
ANDRÉS EMILIANO DE LA GARZA ROSALES A01384096
RODRIGO DE JESÚS MELÉNDEZ MOLINA A00831646
FIDEL MORALES BRIONES A01198630





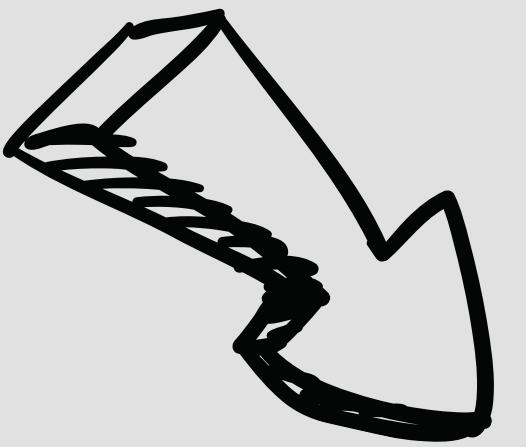
PROBLEMA INICIAL

Se recibió un archivo .txt que contiene tres columnas de datos; fecha, IP, e información. El problema es que esta información no está ordenada, y se pidió ordenar los datos por orden descendente de IP utilizando algoritmos de ordenamiento, para luego pasar los datos ordenados a un nuevo archivo y poder buscar y encontrar un rango de IP's con algoritmos de búsqueda.

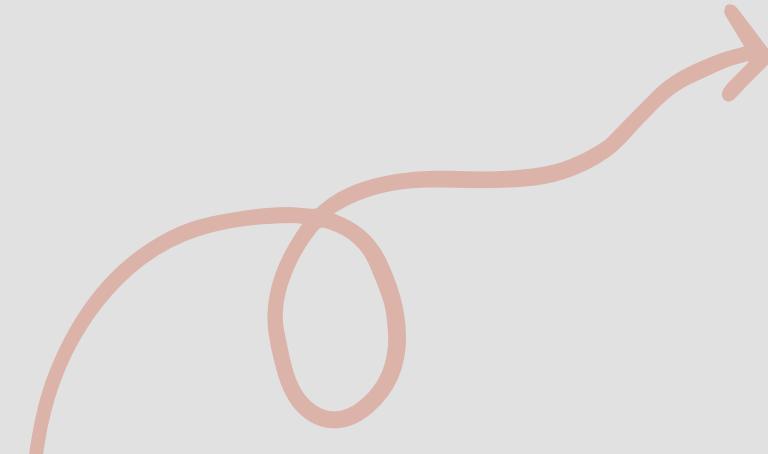
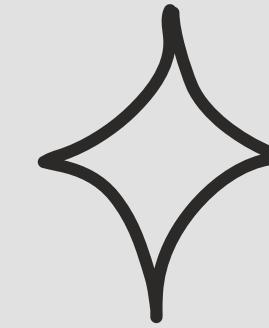
```
bitacoraelb.txt
```

The screenshot shows a dark-themed text editor window titled "bitacoraelb.txt". The menu bar includes "File", "Edit", "View", and a settings gear icon. The main area displays a log file with the following entries:

Date	Time	IP	Message
2020	Jan 20	15:50:25	IP: 201.59.175.55 INFO: User clicked on product
2020	Jan 20	15:50:26	IP: 201.59.175.55 INFO: User clicked on product
2020	Jan 20	15:50:27	IP: 201.59.175.55 INFO: User clicked on product
2022	Nov 23	01:17:25	IP: 119.153.136.208 INFO: User completed a purchase
2023	Jan 23	10:24:57	IP: 206.117.145.162 INFO: User completed a purchase
2022	Jul 07	01:20:42	IP: 126.173.123.25 INFO: User accessed homepage
2022	Feb 11	23:23:00	IP: 208.242.62.218 INFO: User completed a purchase



MÉTODOS **UTILIZADOS**



fileToVector()

Este método lee el archivo y almacena la información en una estructura **message**, que cuenta con las siguientes variables:

- int year, day
- string month, time, info
- long long ip

Para guardarla en cada variable de la estructura se utiliza `getline()` y un ciclo `while` que termina en la última fila del archivo .txt.

Después de que una fila es leída y almacenada dentro de un **message**, esta se almacena en un vector de **messages**.

Entradas: vector con n **messages**, nombre del archivo

Salidas: un vector con n **messages** con todos los datos del archivo

Complejidad en tiempo: O(n)

Complejidad en espacio: O(n)

```
void fileToVector(vector<message>& v, string fileName) {  
    fstream bitacora;  
    message temp;  
    string line, ip1, ip2, ip3, ip4;  
  
    bitacora.open(fileName, ios::in);  
  
    while (getline(bitacora, line, '\n')) {  
        temp.year = stoi(line);  
        getline(bitacora, temp.month, '\n');  
        getline(bitacora, line, '\n');  
        temp.day = stoi(line);  
  
        getline(bitacora, temp.time, '\n');  
  
        getline(bitacora, line, ':' );  
        getline(bitacora, ip1, ':' );  
        getline(bitacora, ip2, ':' );  
        getline(bitacora, ip3, ':' );  
        getline(bitacora, ip4, '\n');  
        temp.ip = ipToLong(ip1, ip2, ip3, ip4);  
  
        getline(bitacora, line, ':' );  
        getline(bitacora, line);  
        line.erase(0, 1);  
        temp.info = line;  
  
        v.push_back(temp);  
    }  
}
```



ipToLong()

```
long long ipToLong(string ip1, string ip2, string ip3, string i
string current, ip = "";
vector<string> ips = { ip1, ip2, ip3, ip4 };

if (ips[0][0] == ' ')
    ips[0].erase(0, 1);

for (int i = 0; i < 4; i++)
{
    current = ips[i];

    if (current.length() == 1) {
        ip += "00" + current;
    }
    else if (current.length() == 2) {
        ip += "0" + current;
    }
    else {
        ip += current;
    }
}

return stoll(ip);
}
```



Este método se encarga de convertir las IP's del archivo de string a long long. Primero se crea un vector de strings con cada uno de los cuatro valores de una IP. Se remueve el espacio en blanco que existe en el primer elemento. Y después se agregan los ceros necesarios a los números que sean de uno o dos dígitos para juntar los cuatro valores de la IP. Finalmente se convierte de string a long long.

Entradas: cuatro strings con los valores de una IP

Salidas: una IP en tipo de variable long long

Complejidad en tiempo: O(1)

Complejidad en espacio: O(1)

merge() mergeSortByIp()

Es el algoritmo tradicional con recursividad de mergeSort(), pero en esta ocasión los elementos del vector se van a ordenar de manera descendente. El criterio de cuál elemento es menor o mayor, será dependiendo del campo **ip** de la estructura **message**, esta es la que definira el nuevo orden del vector.

Características merge():

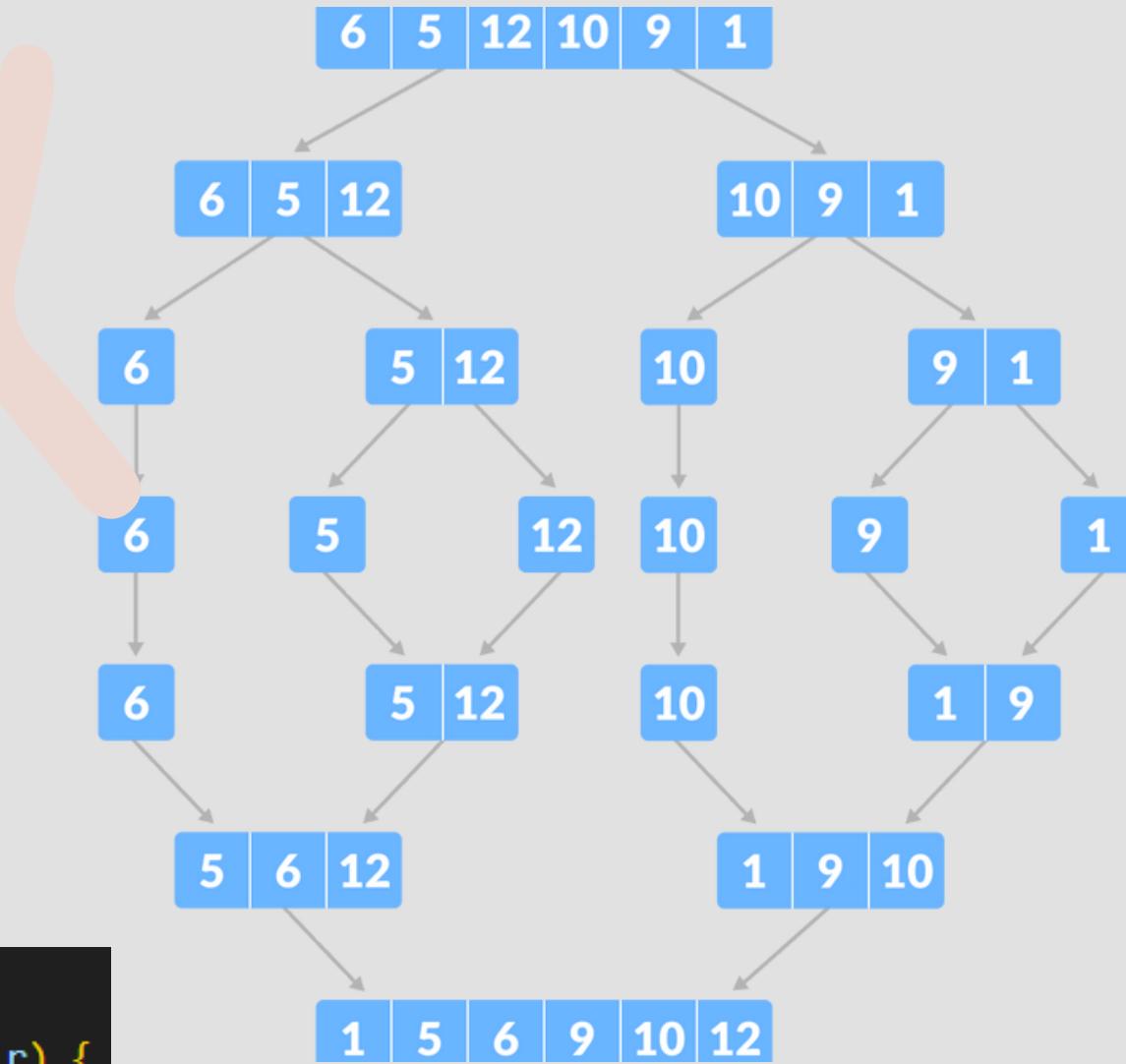
Entradas: un vector con n message, el primer índice de la sección a ordenar, el último índice de la sección a ordenar

Salidas: se modifica el vector dado directamente, no se regresa ningún valor

Complejidad en tiempo: $O(n)$

Complejidad en espacio: $O(n)$

```
void mergeSortByIp(vector<message>& v, int l, int r) {  
    if (l >= r)  
        return;  
  
    int mid = l + (r - l) / 2;  
    mergeSortByIp(v, l, mid);  
    mergeSortByIp(v, mid + 1, r);  
    merge(v, l, r);  
  
    return;  
}
```



Características mergeSortByIp():

Entradas: un vector con n message, el primer índice de la sección a ordenar, el último índice de la sección a ordenar

Salidas: se modifica el vector dado directamente, no se regresa ningún valor

Complejidad en tiempo: $O(n \cdot \log(n))$

Complejidad en espacio: $O(n)$

binarySearch NextSmallest()

```
int binarySearchNextSmallest(vector<message> v, long long target)
{
    int start = 0, end = v.size()-1;
    int ans = end+1;

    while (start <= end)
    {
        int mid = (start + end) / 2;

        if (v[mid].ip >= target)
            start = mid + 1;
        else
        {
            ans = mid;
            end = mid - 1;
        }
    }

    return ans;
}
```

Este método es una versión modificada de la búsqueda binaria. Va a buscar el siguiente elemento menor al que se le indique en la entrada. Como el vector en donde se está realizando la búsqueda está en orden descendente, los signos están volteados.

Entradas: un vector con n **messages** y el número objetivo

Salidas: el índice del número objetivo

Complejidad en tiempo: $O(\log(n))$

Complejidad en espacio: $O(1)$

printFormattedInfo()



```
void printFormattedMessage(message m) {  
    string ipString;  
  
    cout << m.year << " " << m.month << " ";  
    if (m.day < 10){  
        cout << '0';  
    }  
    cout << m.day << " " << m.time << " | IP: ";  
    ipString = to_string(m.ip);  
    while (ipString.length() != 12){  
        ipString = "0" + ipString;  
    }  
    cout << stoi(ipString.substr(0, 3)) << '.' << stoi(ipString.substr(3, 3))  
    cout << " | INFO: " << m.info << endl;  
}
```

Date	IP	Message
2021 Jul 12 16:59:19	IP: 255.226.15.150	INFO: User accessed homepage
2023 Jul 23 05:00:34	IP: 255.225.114.172	INFO: User accessed homepage
2021 Oct 16 16:53:28	IP: 255.224.214.64	INFO: User searched for item
2023 May 26 04:25:09	IP: 255.224.136.206	INFO: User accessed homepage

Se utiliza para imprimir en la terminal cada campo de un **message**. Se imprime con el mismo formato del archivo .txt original para tener consistencia en la legibilidad de los datos. Todos los datos se imprimen en su tipo de variable original, menos el campo **ip**, que se convierte a string y después a int.

Entradas: un **message**
Salidas: se imprimen los datos en la terminal
Complejidad en tiempo: O(1)
Complejidad en espacio: O(1)

toTxtFile()

```
void toTxtFile(vector<message> v, string fileName) {
    fstream bitacora;
    string ipString;

    bitacora.open(fileName, ios::out);

    for (int i = 0; i < v.size(); i++)
    {
        bitacora << v[i].year << " " << v[i].month << " ";
        if (v[i].day < 10)
            bitacora << '0';
        bitacora << v[i].day << " " << v[i].time << " | IP: ";

        ipString = to_string(v[i].ip);
        while (ipString.length() != 12)
            ipString = "0" + ipString;
        bitacora << stoi(ipString.substr(0, 3)) << '.' << stoi(ipString.substr(3, 3)) << '.' <<
        bitacora << " | INFO: " << v[i].info << endl;
    }

    bitacora.close();
}
```

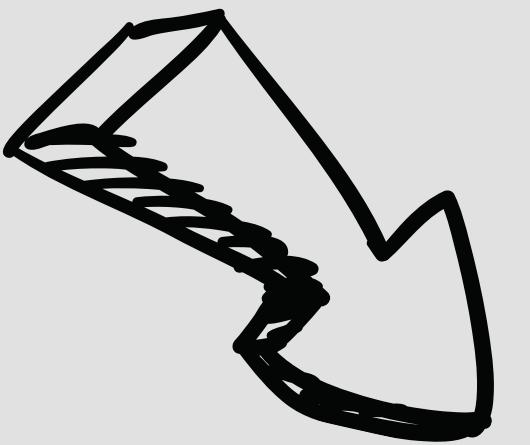
Sirve para mandar los datos ya ordenados dentro de un vector que contiene **messages** a un archivo .txt. Se logra con un ciclo for y se va imprimiendo la información en cada fila de manera ordenada.

Entradas: un vector con n **messages** y el nombre del archivo nuevo

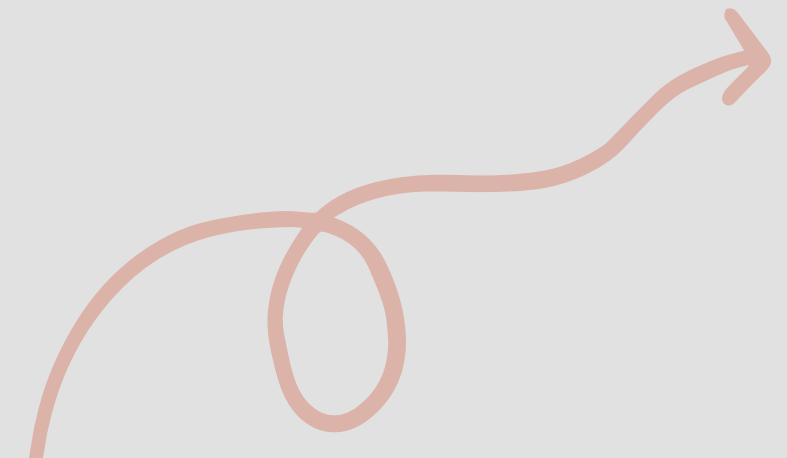
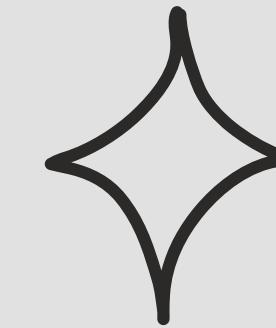
Salidas: un archivo .txt con los datos del vector de **messages**

Complejidad en tiempo: O(n)

Complejidad en espacio: O(n)



FUNCIÓN **MAIN ()**





FUNCIÓN MAIN ()

1. Definición de variables
2. fileToVector()
3. mergeSortByIp()
4. Pedir IP's por buscar
5. Cambiarlas a tipo de variable long long
6. binarySearchNextSmallest()
7. printFormattedInfo()
8. toTxtFile()

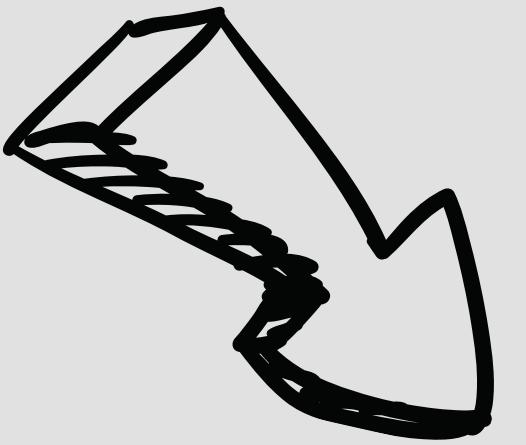
A screenshot of a terminal window showing the contents of a file named "bitacoraelbSorted.txt". The terminal has a dark theme with light-colored text. The file contains log entries with timestamp, IP address, and log level.

```
bitacoraelbSorted.txt
```

File	Edit	View	More Options
			- X +

```
2022 Aug 26 14:41:49 | IP: 255.247.113.32 | INFO: User clicked on product
2020 Apr 29 22:28:22 | IP: 255.244.65.132 | INFO: User completed a purchase
2022 Nov 09 13:59:31 | IP: 255.240.163.11 | INFO: User completed a purchase
2020 Jan 01 02:47:53 | IP: 255.239.61.134 | INFO: User clicked on product
2022 Apr 21 23:01:10 | IP: 255.233.15.32 | INFO: User accessed homepage
```

```
2022 Aug 26 14:41:49 | IP: 255.247.113.32 | INFO: User clicked on product
2020 Apr 29 22:28:22 | IP: 255.244.65.132 | INFO: User completed a purchase
2022 Nov 09 13:59:31 | IP: 255.240.163.11 | INFO: User completed a purchase
2020 Jan 01 02:47:53 | IP: 255.239.61.134 | INFO: User clicked on product
2022 Apr 21 23:01:10 | IP: 255.233.15.32 | INFO: User accessed homepage
```



REFLEXIONES **PERSONALES**

