

Universidad Nacional del Altiplano
Facultad de Ingeniería Estadística e Informática
Docente: Fred Torres Cruz
Estudiante 1 : Ruth Karina Apaza Solis
Estudiante 2 : Maita Mayta Edyydson Brayan

<https://github.com/R-Karina-A-Solis/lectura-de-datos.git>

Trabajo Encargado: Programa de lectura de base de tados

Buscar al cliente y mostrar todos sus datos, gasto de electricidad en 11 meses y en que mes gasto mas:

Busqueda Lineal

```
1 #include <iostream>
2 #include <fstream>
3 #include <sstream>
4 #include <string>
5 #include <chrono>
6 #include <limits>
7
8 using namespace std;
9
10 // Funci n para leer un archivo CSV con ';' como delimitador
11 void leerArchivoCSV(const string& nombreArchivo, string*& encabezados,
12     string**& datos, int& filas, int& columnas) {
13     ifstream archivo(nombreArchivo);
14
15     if (!archivo.is_open()) {
16         cerr << "No se pudo abrir el archivo: " << nombreArchivo << endl;
17         exit(1);
18     }
19
20     string linea;
21     filas = 0;
22     columnas = 0;
23
24     cout << "Leyendo el archivo: " << nombreArchivo << endl;
25
26     // Determinar el n mero de filas y columnas
27     while (getline(archivo, linea)) {
28         filas++;
29         if (filas == 1) { // Contar columnas en la primera l nea
30             stringstream ss(linea);
31             string celda;
32             while (getline(ss, celda, ';')) {
33                 columnas++;
34             }
35         }
36     }
```

```
35     }
36
37     archivo.clear();
38     archivo.seekg(0, ios::beg); // Volver al inicio del archivo
39
40     // Asignar memoria para encabezados y datos
41     encabezados = new string[columnas];
42     datos = new string*[filas - 1];
43     for (int i = 0; i < filas - 1; ++i) {
44         datos[i] = new string[columnas];
45     }
46
47     // Leer los datos del archivo
48     bool esPrimeraLinea = true;
49     int filaActual = 0;
50
51     while (getline(archivo, linea)) {
52         stringstream ss(linea);
53         string celda;
54         int columnaActual = 0;
55
56         while (getline(ss, celda, ';')) {
57             if (esPrimeraLinea) {
58                 encabezados[columnaActual] = celda;
59             } else {
60                 datos[filaActual][columnaActual] = celda;
61             }
62             columnaActual++;
63         }
64
65         if (!esPrimeraLinea) {
66             filaActual++;
67         } else {
68             esPrimeraLinea = false;
69         }
70     }
71
72     archivo.close();
73 }
74
75 // Funci n para buscar al cliente, mostrar todos los datos, sumar y
76 // encontrar el mayor
77 void buscarCliente(string** datos, string* encabezados, int filas, int
78 columnas, const string& clienteID) {
79     if (filas <= 0 || columnas <= 0) {
80         cout << "No hay datos disponibles.\n";
81         return;
82     }
83
84     auto inicio = chrono::high_resolution_clock::now();
85     bool encontrado = false;
86     int iteraciones = 0;
87
88     for (int i = 0; i < filas; ++i) {
```

```
87     iteraciones++;
88     if (datos[i][0] == clienteID) {
89         encontrado = true;
90         cout << "Cliente_encontrado:\n";
91         for (int j = 0; j < columnas; ++j) {
92             cout << encabezados[j] << ":\n" << datos[i][j] << endl;
93         }
94         double suma = 0, mayorValor = numeric_limits<double>::lowest();
95         ;
96         int columnaMayor = -1;
97         for (int j = 6; j < 17 && j < columnas; ++j) {
98             try {
99                 double valor = stod(datos[i][j]);
100                suma += valor;
101                if (valor > mayorValor) {
102                    mayorValor = valor;
103                    columnaMayor = j;
104                }
105            } catch (const invalid_argument&) {}
106        }
107        cout << "En_11_meses_gasto" << suma << "watts_de_energia" <<
108            endl;
109        if (columnaMayor != -1) {
110            cout << "En_el_mes_de_" << encabezados[columnaMayor]
111                << "gasto_la_mayor_cantidad_de_energia_electrica,_
112                que_es_de_" << mayorValor << "watts" << endl;
113        }
114        break;
115    }
116    if (!encontrado) {
117        cout << "Cliente_no_encontrado.\n";
118    }
119    auto fin = chrono::high_resolution_clock::now();
120    chrono::duration<double, milli> tiempo = fin - inicio;
121
122    cout << "Tiempo_de_búsqueda:" << tiempo.count() << "ms\n";
123 }
124
125 // Liberar memoria asignada din micamente
126 void liberarMemoria(string*& encabezados, string**& datos, int filas) {
127     delete[] encabezados;
128     for (int i = 0; i < filas; ++i) {
129         delete[] datos[i];
130     }
131     delete[] datos;
132 }
133
134 int main() {
135     string* encabezados = nullptr;
136     string** datos = nullptr;
137     int filas = 0, columnas = 0;
```

```

138     string nombreArchivo = "data_medidor.csv";
139     leerArchivoCSV(nombreArchivo, encabezados, datos, filas, columnas);
140     while (true) {
141         string clienteID;
142         cout << "Ingrese el numero de cliente (o escriba 'salir' para
            terminar): ";
143         getline(cin, clienteID);
144         if (clienteID == "salir") {
145             break;
146         }
147         buscarCliente(datos, encabezados, filas - 1, columnas, clienteID);
148     }
149     liberarMemoria(encabezados, datos, filas - 1);
150     return 0;
151 }

```

```

.49 void liber
.50 delete
.51 for (i
.52 de
.53 }
.54 delete
.55 }
.56
.57 int main()
.58 string
.59 string
.60 int fi
.61 string
.62
.63 // Lee
.64 leerAr
.65 cout <
.66
.67 while
.68 st
.69 co
.70 ge
.71
.72 if
.73
.74 }
.75
.76 En 11 meses gasto 22246 watts de energia
.77 // En el mes de NOV12 gasto la mayor cantidad de energia electrica, que es de 2079 wa
.78 bu
.79 tts
.80 Tiempo de busqueda: 9.4756 ms
.81 // Numero de iteraciones: 176
.82 libera
.83 Ingrese el numero de cliente (o escriba 'salir' para terminar): |

```

Figura 1: Busqueda lineal

Busqueda Binomial

```
1 #include <iostream>
2 #include <fstream>
3 #include <sstream>
4 #include <string>
5 #include <chrono>
6 #include <limits>
7 #include <algorithm>
8
9 using namespace std;
10
11 // Funci n para leer un archivo CSV con ';' como delimitador
12 void leerArchivoCSV(const string& nombreArchivo, string*& encabezados,
13 string**& datos, int& filas, int& columnas) {
14     ifstream archivo(nombreArchivo);
15
16     if (!archivo.is_open()) {
17         cerr << "No se pudo abrir el archivo: " << nombreArchivo << endl;
18         exit(1);
19     }
20
21     string linea;
22     filas = 0;
23     columnas = 0;
24
25     // Determinar el n mero de filas y columnas
26     while (getline(archivo, linea)) {
27         filas++;
28         if (filas == 1) { // Contar columnas en la primera l nea
29             stringstream ss(linea);
30             string celda;
31             while (getline(ss, celda, ';')) {
32                 columnas++;
33             }
34         }
35     }
36
37     archivo.clear();
38     archivo.seekg(0, ios::beg); // Volver al inicio del archivo
39
40     // Asignar memoria para encabezados y datos
41     encabezados = new string[columnas];
42     datos = new string*[filas - 1];
43     for (int i = 0; i < filas - 1; ++i) {
44         datos[i] = new string[columnas];
45     }
46
47     // Leer los datos del archivo
48     bool esPrimeraLinea = true;
49     int filaActual = 0;
50
51     while (getline(archivo, linea)) {
```

```
52     string celda;
53     int columnaActual = 0;
54
55     while (getline(ss, celda, ',')) {
56         if (esPrimeraLinea) {
57             encabezados[columnaActual] = celda;
58         } else {
59             datos[filaActual][columnaActual] = celda;
60         }
61         columnaActual++;
62     }
63
64     if (!esPrimeraLinea) {
65         filaActual++;
66     } else {
67         esPrimeraLinea = false;
68     }
69 }
70
71 archivo.close();
72 }
73
74 // Funci n para realizar una b squeda binaria con conteo de iteraciones
75 int busquedaBinaria(string** datos, int filas, const string& clienteID,
76 int& iteraciones) {
77     int izquierda = 0;
78     int derecha = filas - 1;
79     iteraciones = 0;
80
81     while (izquierda <= derecha) {
82         iteraciones++;
83         int medio = izquierda + (derecha - izquierda) / 2;
84         if (datos[medio][0] == clienteID) {
85             return medio; // Cliente encontrado
86         }
87         if (datos[medio][0] < clienteID) {
88             izquierda = medio + 1; // Buscar en la mitad derecha
89         } else {
90             derecha = medio - 1; // Buscar en la mitad izquierda
91         }
92     }
93     return -1; // Cliente no encontrado
94 }
95
96 // Funci n para ordenar los datos por el ID del cliente con conteo de
97 // iteraciones
98 void ordenarDatosPorID(string** datos, int filas, int& iteraciones) {
99     iteraciones = 0;
100     for (int i = 0; i < filas - 1; ++i) {
101         for (int j = i + 1; j < filas; ++j) {
102             iteraciones++;
103             if (datos[i][0] > datos[j][0]) {
104                 // Intercambiar filas
105                 swap(datos[i], datos[j]);
106             }
107         }
108     }
109 }
```

```

104     }
105 }
106 }
107 }
108
109 // Funci n para buscar al cliente, mostrar todos los datos, sumar y
    encontrar el mayor
110 void buscarCliente(string** datos, string* encabezados, int filas, int
    columnas, const string& clienteID) {
111     auto inicio = chrono::high_resolution_clock::now();
112     int iteracionesBusqueda;
113
114     // Buscar cliente usando b squeda binaria
115     int indiceCliente = busquedaBinaria(datos, filas, clienteID,
        iteracionesBusqueda);
116     if (indiceCliente != -1) {
117         cout << "Cliente encontrado:\n";
118
119         // Mostrar todos los datos de la fila
120         for (int j = 0; j < columnas; ++j) {
121             cout << encabezados[j] << ": " << datos[indiceCliente][j] <<
                endl;
122         }
123
124         // Sumar los valores de las columnas 7 a 17
125         double suma = 0, mayorValor = numeric_limits<double>::lowest();
126         int columnaMayor = -1;
127
128         for (int j = 6; j < 17 && j < columnas; ++j) { // Columnas 7 (
            ndice 6) a 17
129             try {
130                 double valor = stod(datos[indiceCliente][j]);
131                 suma += valor;
132                 if (valor > mayorValor) {
133                     mayorValor = valor;
134                     columnaMayor = j;
135                 }
136             } catch (const invalid_argument&) {
137                 cerr << "Error al convertir el valor: " << datos[
                    indiceCliente][j] << ".\n";
138             }
139         }
140
141         cout << "En 11 meses gasto: " << suma << " watts de energia" <<
            endl;
142         if (columnaMayor != -1) {
143             cout << "En el mes de " << encabezados[columnaMayor]
144                 << "gasto la mayor cantidad de energia electrica, que es
                    de " << mayorValor << " watss" << endl;
145         } else {
146             cout << "No se encontraron valores validos en las columnas 7
                a 17.\n";
147         }
148     } else {

```

```
149     cout << "Cliente_no_encontrado.\n";
150 }
151
152 auto fin = chrono::high_resolution_clock::now();
153 chrono::duration<double, milli> tiempo = fin - inicio;
154
155 cout << "Tiempo_de_búsqueda:\n" << tiempo.count() << "\nms\n";
156 cout << "Numero_de_iteraciones:\n" << iteracionesBusqueda << endl;
157 }
158
159 // Liberar memoria asignada dinamicamente
160 void liberarMemoria(string*& encabezados, string**& datos, int filas) {
161     delete[] encabezados;
162     for (int i = 0; i < filas; ++i) {
163         delete[] datos[i];
164     }
165     delete[] datos;
166 }
167
168 int main() {
169     string* encabezados = nullptr;          // Arreglo dinámico para los
170         encabezados
171     string** datos = nullptr;               // Arreglo dinámico para los
172         datos
173     int filas = 0, columnas = 0;            // Número de filas y columnas
174     string nombreArchivo = "data_medidor.csv"; // Nombre del archivo CSV
175
176     // Leer el archivo automáticamente al iniciar
177     leerArchivoCSV(nombreArchivo, encabezados, datos, filas, columnas);
178     cout << "Archivo_leido_correctamente.\n";
179
180     // Ordenar los datos por el identificador del cliente (suponiendo que
181     // el ID está en la primera columna)
182     int iteracionesOrdenamiento;
183     ordenarDatosPorID(datos, filas - 1, iteracionesOrdenamiento);
184     cout << "Datos_ordenados_correctamente_en\n" << iteracionesOrdenamiento
185         << "\niteraciones.\n";
186
187     while (true) {
188         string clienteID;
189         cout << "Ingrese_el_numero_de_cliente(o_escriba'salir'para_terminar):\n";
190         getline(cin, clienteID);
191
192         if (clienteID == "salir") {
193             cout << "Saliendo_del_programa...\n";
194             break;
195         }
196
197         // Buscar cliente y procesar los datos
198         buscarCliente(datos, encabezados, filas - 1, columnas, clienteID);
199     }
200
201     // Liberar memoria
```



```
198     liberarMemoria(encabezados, datos, filas - 1);
199
200     return 0;
201 }
```

```
159 // liberar memoria asignada dinamicamente
160 void liberarMemoria(string*& encabezados, string**& datos, int filas) {
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
```

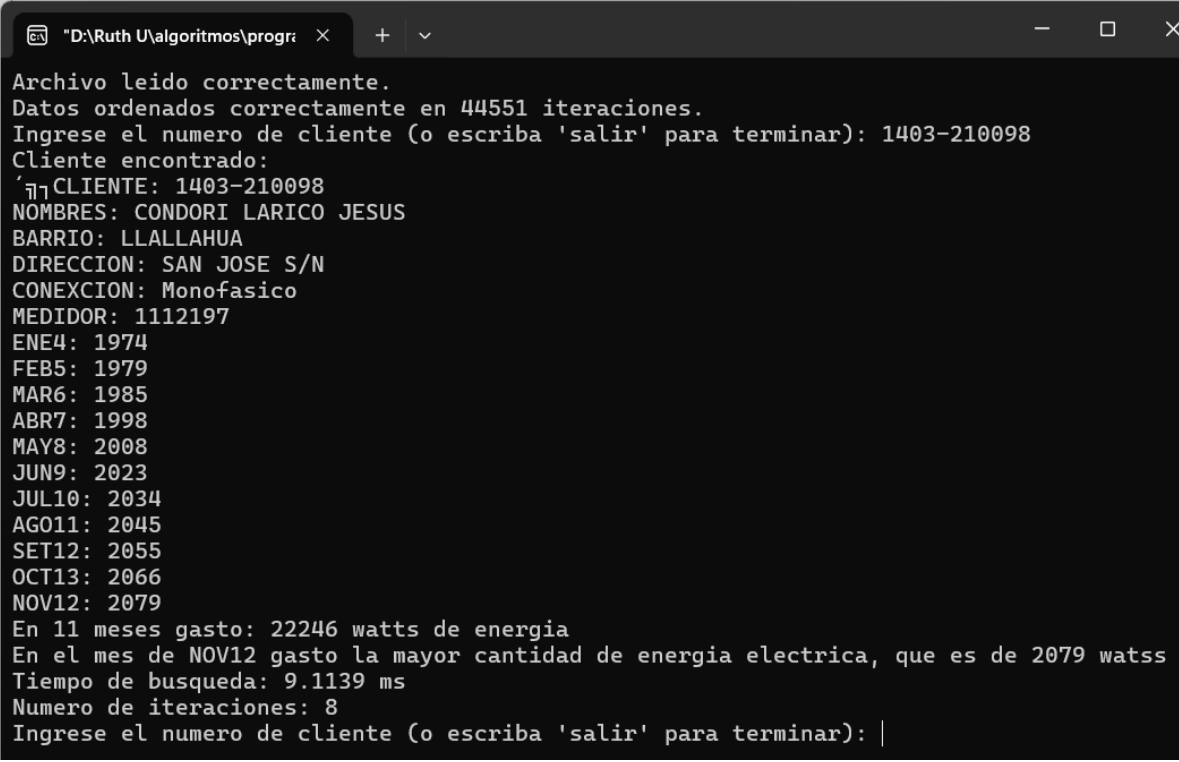


Figura 2: Busqueda binomial

Busqueda por indice

```
1 #include <iostream>
2 #include <fstream>
3 #include <sstream>
4 #include <string>
5 #include <chrono>
6 #include <limits>
7 #include <unordered_map>
8
9 using namespace std;
10
11 // Funci n para leer un archivo CSV con ';' como delimitador y construir
   un ndice
12 void leerArchivoCSV(const string& nombreArchivo, string*& encabezados,
   string**& datos, int& filas, int& columnas, unordered_map<string, int>&
   indice) {
13     ifstream archivo(nombreArchivo);
14
15     if (!archivo.is_open()) {
16         cerr << "No se pudo abrir el archivo: " << nombreArchivo << endl;
17         exit(1);
18     }
19
20     string linea;
21     filas = 0;
22     columnas = 0;
23
24     // Determinar el n mero de filas y columnas
25     while (getline(archivo, linea)) {
26         filas++;
27         if (filas == 1) { // Contar columnas en la primera l nea
28             stringstream ss(linea);
29             string celda;
30             while (getline(ss, celda, ';')) {
31                 columnas++;
32             }
33         }
34     }
35
36     archivo.clear();
37     archivo.seekg(0, ios::beg); // Volver al inicio del archivo
38
39     // Asignar memoria para encabezados y datos
40     encabezados = new string[columnas];
41     datos = new string*[filas - 1];
42     for (int i = 0; i < filas - 1; ++i) {
43         datos[i] = new string[columnas];
44     }
45
46     // Leer los datos del archivo y construir el ndice
47     bool esPrimeraLinea = true;
48     int filaActual = 0;
49 }
```

```
50 while (getline(archivo, linea)) {
51     stringstream ss(linea);
52     string celda;
53     int columnaActual = 0;
54
55     while (getline(ss, celda, ',')) {
56         if (esPrimeraLinea) {
57             encabezados[columnaActual] = celda;
58         } else {
59             datos[filaActual][columnaActual] = celda;
60         }
61         columnaActual++;
62     }
63
64     if (!esPrimeraLinea) {
65         // Agregar el identificador nico al ndice
66         indice[datos[filaActual][0]] = filaActual; // Supone que el
67             identificador nico est en la primera columna
68         filaActual++;
69     }
70     esPrimeraLinea = false;
71 }
72
73 archivo.close();
74
75 // Funci n para buscar al cliente simulando iteraciones
76 void buscarClienteSimulado(string** datos, string* encabezados, int
77     columnas, int filas, const string& clienteID) {
78     auto inicio = chrono::high_resolution_clock::now();
79
80     int iteracionesBusqueda = 0; // Contador de iteraciones de b squeda
81     int filaEncontrada = -1;
82
83     // Simular b squeda lineal en los datos
84     for (int i = 0; i < filas; ++i) {
85         iteracionesBusqueda++;
86         if (datos[i][0] == clienteID) { // Supone que el clienteID est
87             en la primera columna
88             filaEncontrada = i;
89             break;
90         }
91     }
92
93     if (filaEncontrada != -1) {
94         cout << "Cliente_encontrado:\n";
95
96         // Mostrar todos los datos de la fila
97         for (int j = 0; j < columnas; ++j) {
98             cout << encabezados[j] << ":\n" << datos[filaEncontrada][j] <<
99                 endl;
100         }
101
102         // Sumar los valores de las columnas 7 a 17
```

```

100     double suma = 0, mayorValor = numeric_limits<double>::lowest();
101     int columnaMayor = -1;
102
103     for (int j = 6; j < 17 && j < columnas; ++j) {
104         try {
105             double valor = stod(datos[filaEncontrada][j]);
106             suma += valor;
107             if (valor > mayorValor) {
108                 mayorValor = valor;
109                 columnaMayor = j;
110             }
111         } catch (const invalid_argument&) {
112             cerr << "Error al convertir el valor: " << datos[
113                 filaEncontrada][j] << ".\n";
114         }
115
116         cout << "En 11 meses gasto " << suma << " watts de energia " <<
117             endl;
118         if (columnaMayor != -1) {
119             cout << "En el mes de " << encabezados[columnaMayor]
120                 << " gasto la mayor cantidad de energia electrica, que es
121                 de " << mayorValor << " watts " << endl;
122         } else {
123             cout << "No se encontraron valores validos en las columnas 7 a
124                 17.\n";
125         }
126     } else {
127         cout << "Cliente no encontrado.\n";
128     }
129
130     cout << "Numero de iteraciones en la busqueda: " <<
131         iteracionesBusqueda << endl;
132
133     auto fin = chrono::high_resolution_clock::now();
134     chrono::duration<double, milli> tiempo = fin - inicio;
135
136     cout << "Tiempo de busqueda: " << tiempo.count() << " ms\n";
137 }
138
139 // Liberar memoria asignada dinamicamente
140 void liberarMemoria(string*& encabezados, string**& datos, int filas) {
141     delete[] encabezados;
142     for (int i = 0; i < filas; ++i) {
143         delete[] datos[i];
144     }
145     delete[] datos;
146 }
147
148 int main() {
149     string* encabezados = nullptr; // Arreglo dinámico para los
150         encabezados
151     string** datos = nullptr; // Arreglo dinámico para los
152         datos

```

```
147     int filas = 0, columnas = 0;           // Número de filas y columnas
148     unordered_map<string, int> indice; // índice para búsqueda rápida
149     string nombreArchivo = "data_medidor.csv"; // Nombre del archivo CSV
150
151     // Leer el archivo automáticamente al iniciar
152     leerArchivoCSV(nombreArchivo, encabezados, datos, filas, columnas,
153                   indice);
154     cout << "Archivo leído correctamente.\n";
155
156     while (true) {
157         string clienteID;
158         cout << "Ingrese el número de cliente (o escriba 'salir' para
159               terminar):\n";
160         getline(cin, clienteID);
161
162         if (clienteID == "salir") {
163             cout << "Saliendo del programa...\n";
164             break;
165         }
166
167         // Buscar cliente simulando iteraciones
168         buscarClienteSimulado(datos, encabezados, columnas, filas - 1,
169                               clienteID);
170
171     }
172
173     // Liberar memoria asignada dinámicamente
174     liberarMemoria(encabezados, datos, filas - 1);
175
176     return 0;
177 }
```

```
132 cout << "Tiempo de busqueda: " << tiempo.count() << " ms\n";
133
134
135
136 Archivo leído correctamente.
137 Ingrese el numero de cliente (o escriba 'salir' para terminar): 1403-210098
138 Cliente encontrado:
139 CLIENTE: 1403-210098
140 NOMBRES: CONDORI LARICO JESUS
141 BARRIO: LLALLAHUA
142 DIRECCION: SAN JOSE S/N
143 CONEXCION: Monofasico
144 MEDIDOR: 1112197
145 ENE4: 1974
146 FEB5: 1979
147 MAR6: 1985
148 ABR7: 1998
149 MAY8: 2008
150 JUN9: 2023
151 JUL10: 2034
152 AGO11: 2045
153 SET12: 2055
154 OCT13: 2066
155 NOV12: 2079
156 En 11 meses gasto 22246 watts de energia
157 En el mes de NOV12 gasto la mayor cantidad de energia electrica, que es de 2079 watts
158 Numero de iteraciones en la busqueda: 176
159 Tiempo de busqueda: 14.696 ms
160 Ingrese el numero de cliente (o escriba 'salir' para terminar): |
161
162
```

Figura 3: Busqueda por indice

Busqueda por hash

```
1 #include <iostream>
2 #include <fstream>
3 #include <sstream>
4 #include <string>
5 #include <chrono>
6 #include <limits>
7 #include <unordered_map>\vspace{5mm}
8
9 \begin{figure} [H]
10     \centering
11     \includegraphics[width=1.0\linewidth]{indice.png}
12     \caption{}
13 \end{figure}
14
15 using namespace std;
16
17 // Funci n para leer un archivo CSV con ';' como delimitador
18 void leerArchivoCSV(const string& nombreArchivo, string*& encabezados,
19     string**& datos, int& filas, int& columnas, unordered_map<string, int>&
20     hashClientes) {
21     ifstream archivo(nombreArchivo);
22
23     if (!archivo.is_open()) {
24         cerr << "No se pudo abrir el archivo: " << nombreArchivo << endl;
25         exit(1);
26     }
27
28     string linea;
29     filas = 0;
30     columnas = 0;
31
32     while (getline(archivo, linea)) {
33         filas++;
34         if (filas == 1) { // Contar columnas en la primera l nea
35             stringstream ss(linea);
36             string celda;
37             while (getline(ss, celda, ';')) {
38                 columnas++;
39             }
40         }
41
42         archivo.clear();
43         archivo.seekg(0, ios::beg); // Volver al inicio del archivo
44
45         encabezados = new string[columnas];
46         datos = new string*[filas - 1];
47         for (int i = 0; i < filas - 1; ++i) {
48             datos[i] = new string[columnas];
49         }
50
51         bool esPrimeraLinea = true;
```

```
51     int filaActual = 0;
52
53     while (getline(archivo, linea)) {
54         stringstream ss(linea);
55         string celda;
56         int columnaActual = 0;
57
58         while (getline(ss, celda, ',')) {
59             if (esPrimeraLinea) {
60                 encabezados[columnaActual] = celda;
61             } else {
62                 datos[filaActual][columnaActual] = celda;
63             }
64             columnaActual++;
65         }
66
67         if (!esPrimeraLinea) {
68             hashClientes[datos[filaActual][0]] = filaActual; // Supone que
69                 el clienteID est  en la primera columna
70             filaActual++;
71         } else {
72             esPrimeraLinea = false;
73         }
74     }
75     archivo.close();
76 }
77
78 // Funci n para buscar al cliente y procesar sus datos
79 void buscarCliente(const unordered_map<string, int>& hashClientes, string
80     ** datos, string* encabezados, int columnas, const string& clienteID) {
81     auto inicio = chrono::high_resolution_clock::now();
82     int iteraciones = 0;
83
84     // Buscar el cliente en la tabla hash
85     iteraciones++; // Incrementa una iteraci n al buscar en el hash
86     auto it = hashClientes.find(clienteID);
87
88     if (it != hashClientes.end()) {
89         int fila = it->second;
90         cout << "Cliente encontrado:\n";
91
92         // Mostrar todos los datos de la fila
93         for (int j = 0; j < columnas; ++j) {
94             cout << encabezados[j] << ": " << datos[fila][j] << endl;
95             iteraciones++; // Una iteraci n por columna mostrada
96         }
97
98         // Sumar valores de las columnas 7 a 17 y encontrar el mayor
99         double suma = 0, mayorValor = numeric_limits<double>::lowest();
100         int columnaMayor = -1;
101
102         for (int j = 6; j < 17 && j < columnas; ++j) {
103             iteraciones++; // Incrementa iteraciones por operaci n en
```



```

        cada columna
103     try {
104         double valor = stod(datos[filas][j]);
105         suma += valor;
106         if (valor > mayorValor) {
107             mayorValor = valor;
108             columnaMayor = j;
109         }
110     } catch (const invalid_argument&) {
111         cerr << "Error al convertir el valor: " << datos[filas][j]
112             << ".\n";
113     }
114 }
115 cout << "En 11 meses gasto: " << suma << " watts de energia." <<
    endl;
116 if (columnaMayor != -1) {
117     cout << "En el mes de " << encabezados[columnaMayor]
118         << "gasto la mayor cantidad de energia electrica: " <<
        mayorValor << " watts." << endl;
119 } else {
120     cout << "No se encontraron valores validos en las columnas 7 a
        17.\n";
121 }
122 } else {
123     cout << "Cliente no encontrado.\n";
124 }
125
126 auto fin = chrono::high_resolution_clock::now();
127 chrono::duration<double, milli> tiempo = fin - inicio;
128
129 cout << "Tiempo de b squeda: " << tiempo.count() << "ms\n";
130 cout << "Numero de iteraciones realizadas: " << iteraciones << endl;
131 }
132
133 // Liberar memoria asignada din micamente
134 void liberarMemoria(string*& encabezados, string**& datos, int filas) {
135     delete[] encabezados;
136     for (int i = 0; i < filas; ++i) {
137         delete[] datos[i];
138     }
139     delete[] datos;
140 }
141
142 int main() {
143     string* encabezados = nullptr;
144     string** datos = nullptr;
145     int filas = 0, columnas = 0;
146     string nombreArchivo = "data_medidor.csv";
147
148     unordered_map<string, int> hashClientes;
149
150     leerArchivoCSV(nombreArchivo, encabezados, datos, filas, columnas,
        hashClientes);

```

```
151     cout << "Archivo leido correctamente.\n";
152
153     while (true) {
154         string clienteID;
155         cout << "Ingrese el numero de cliente (o escriba 'salir' para
156             terminar):\n";
157         getline(cin, clienteID);
158
159         if (clienteID == "salir") {
160             cout << "Saliendo del programa...\n";
161             break;
162         }
163
164         buscarCliente(hashClientes, datos, encabezados, columnas,
165             clienteID);
166
167     }
168
169     liberarMemoria(encabezados, datos, filas - 1);
170
171     return 0;
172 }
```

```
120     auto fin = chrono::high_resolution_clock::now();
121
122
123
124     Archivo leido correctamente.
125     Ingrese el numero de cliente (o escriba 'salir' para terminar): 1403-210098
126     Cliente encontrado:
127     'CLIENTE: 1403-210098
128     NOMBRES: CONDORI LARICO JESUS
129     BARRIO: LLALLAHUA
130     DIRECCION: SAN JOSE S/N
131     CONEXCION: Monofasico
132     MEDIDOR: 1112197
133     ENE4: 1974
134     FEB5: 1979
135     MAR6: 1985
136     ABR7: 1998
137     MAY8: 2008
138     JUN9: 2023
139     JUL10: 2034
140     AGO11: 2045
141     SET12: 2055
142     OCT13: 2066
143     NOV12: 2079
144     En 11 meses gasto: 22246 watts de energia.
145     En el mes de NOV12 gasto la mayor cantidad de energia electrica: 2079 watts.
146     Tiempo de busqueda: 6.5314 ms
147     Numero de iteraciones realizadas: 29
148     Ingrese el numero de cliente (o escriba 'salir' para terminar): |
149
150
151
152
153
154
```

Figura 4: Busqueda de hash

Busqueda por saltos

```
1 #include <iostream>
2 #include <fstream>
3 #include <sstream>
4 #include <string>
5 #include <chrono>
6 #include <limits>
7 #include <cmath>
8 #include <algorithm> // Para std::sort
9
10 using namespace std;
11
12 // Funci n para leer un archivo CSV con ';' como delimitador
13 void leerArchivoCSV(const string& nombreArchivo, string*& encabezados,
14     string**& datos, int& filas, int& columnas) {
15     ifstream archivo(nombreArchivo);
16
17     if (!archivo.is_open()) {
18         cerr << "No se pudo abrir el archivo: " << nombreArchivo << endl;
19         exit(1);
20     }
21
22     string linea;
23     filas = 0;
24     columnas = 0;
25
26     cout << "Leyendo el archivo: " << nombreArchivo << endl;
27
28     // Determinar el n mero de filas y columnas
29     while (getline(archivo, linea)) {
30         filas++;
31         if (filas == 1) { // Contar columnas en la primera l nea
32             stringstream ss(linea);
33             string celda;
34             while (getline(ss, celda, ';')) {
35                 columnas++;
36             }
37         }
38     }
39
40     archivo.clear();
41     archivo.seekg(0, ios::beg); // Volver al inicio del archivo
42
43     // Asignar memoria para encabezados y datos
44     encabezados = new string[columnas];
45     datos = new string*[filas - 1];
46     for (int i = 0; i < filas - 1; ++i) {
47         datos[i] = new string[columnas];
48     }
49
50     // Leer los datos del archivo
51     bool esPrimeraLinea = true;
52     int filaActual = 0;
```

```
52
53 while (getline(archivo, linea)) {
54     stringstream ss(linea);
55     string celda;
56     int columnaActual = 0;
57
58     while (getline(ss, celda, ',')) {
59         if (esPrimeraLinea) {
60             encabezados[columnaActual] = celda;
61         } else {
62             datos[filaActual][columnaActual] = celda;
63         }
64         columnaActual++;
65     }
66
67     if (!esPrimeraLinea) {
68         // Mostrar cada fila de datos le da
69         cout << "Fila_" << filaActual + 1 << ":\n";
70         for (int j = 0; j < columnas; ++j) {
71             cout << datos[filaActual][j] << (j < columnas - 1 ? "," : " " :
72                 "");
73         }
74         cout << endl;
75         filaActual++;
76     } else {
77         // Mostrar los encabezados le dos
78         cout << "Encabezados:\n";
79         for (int j = 0; j < columnas; ++j) {
80             cout << encabezados[j] << (j < columnas - 1 ? "," : " ");
81         }
82         cout << endl;
83         esPrimeraLinea = false;
84     }
85 }
86
87 archivo.close();
88
89 // Funci n para realizar la b squeda por salto (Jump Search)
90 void buscarClienteJumpSearch(string** datos, string* encabezados, int
91     filas, int columnas, const string& clienteID) {
92     if (filas <= 0 || columnas <= 0) {
93         cout << "No hay datos disponibles.\n";
94         return;
95     }
96
97     auto inicio = chrono::high_resolution_clock::now();
98     bool encontrado = false;
99     int iteraciones = 0;
100
101     // Ordenar los datos por el ID del cliente para realizar Jump Search
102     sort(datos, datos + filas, [](const string* a, const string* b) {
103         return a[0] < b[0]; // Suponiendo que el ID del cliente est en
104             la primera columna
105     });
106 }
```

```

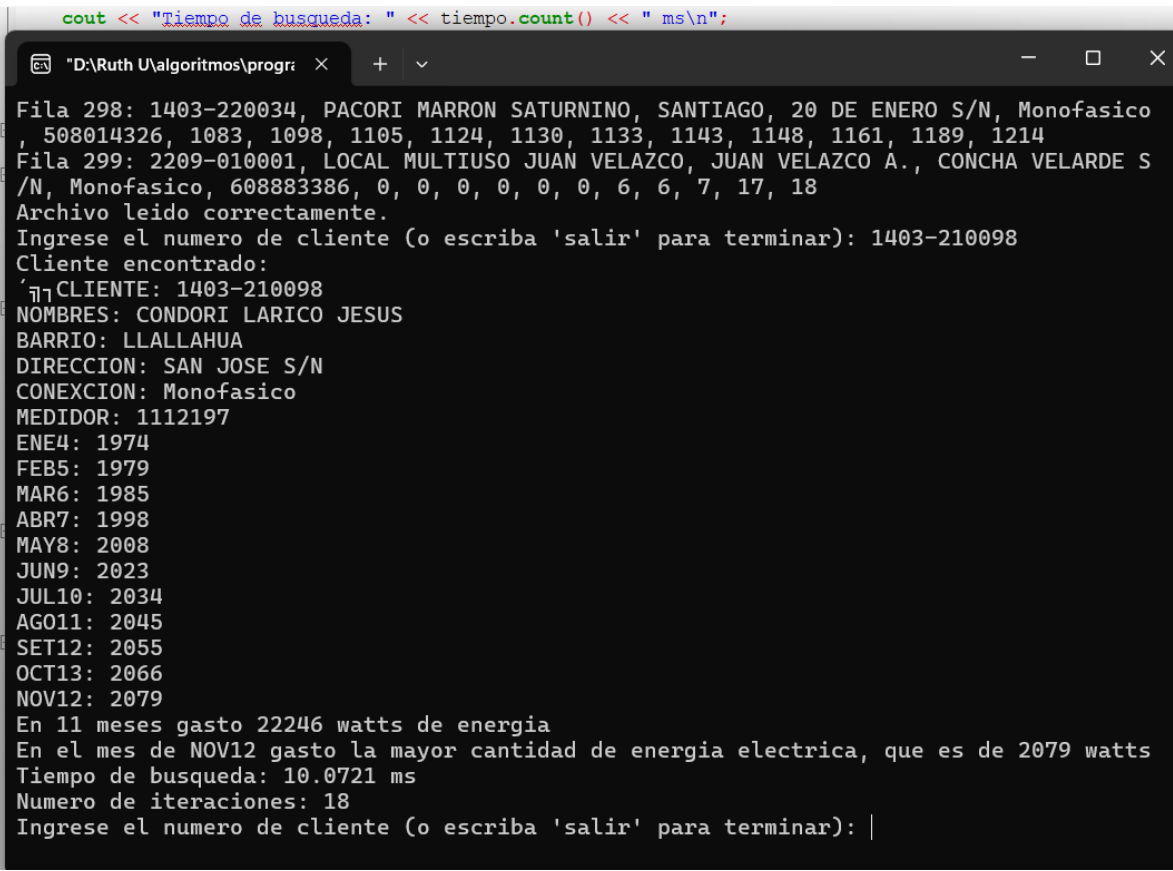
103     });
104
105     // Tamaño del salto
106     int salto = sqrt(filas);
107     int prev = 0;
108
109     // Buscar el bloque en el que podrá estar el cliente
110     while (prev < filas && datos[min(filas, prev + salto)][0] < clienteID)
111     {
112         prev += salto;
113         iteraciones++;
114     }
115
116     // Realizar búsqueda lineal en el bloque
117     for (int i = prev; i < min(prev + salto, filas); ++i) {
118         iteraciones++;
119         if (datos[i][0] == clienteID) { // Supone que el identificador
120             // está en la primera columna
121             encontrado = true;
122             cout << "Cliente encontrado:\n";
123
124             // Mostrar todos los datos de la fila
125             for (int j = 0; j < columnas; ++j) {
126                 cout << encabezados[j] << ": " << datos[i][j] << endl;
127             }
128
129             // Sumar los valores de las columnas 7 a 17
130             double suma = 0, mayorValor = numeric_limits<double>::lowest();
131             int columnaMayor = -1;
132
133             for (int j = 6; j < 17 && j < columnas; ++j) { // Columnas 7 (índice 6) a 17
134                 try {
135                     double valor = stod(datos[i][j]);
136                     suma += valor;
137                     if (valor > mayorValor) {
138                         mayorValor = valor;
139                         columnaMayor = j;
140                     }
141                 } catch (const invalid_argument&) {
142                     cerr << "Error al convertir el valor: " << datos[i][j]
143                         << ".\n";
144                 }
145             }
146
147             cout << "En 11 meses gasto " << suma << " watts de energía " <<
148                 endl;
149             if (columnaMayor != -1) {
150                 cout << "En el mes de " << encabezados[columnaMayor]
151                     << "gasto la mayor cantidad de energía eléctrica, "
152                     << "que es de " << mayorValor << " watts " << endl;
153             } else {
154                 cout << "No se encontraron valores válidos en las "

```

```
150         }
151
152         break;
153     }
154 }
155
156 if (!encontrado) {
157     cout << "Cliente_no_encontrado.\n";
158 }
159
160 auto fin = chrono::high_resolution_clock::now();
161 chrono::duration<double, milli> tiempo = fin - inicio;
162
163 cout << "Tiempo_de_búsqueda:_ " << tiempo.count() << "ms\n";
164 cout << "Numero_de_iteraciones:_ " << iteraciones << endl;
165 }
166
167 // Liberar memoria asignada dinamicamente
168 void liberarMemoria(string*& encabezados, string**& datos, int filas) {
169     delete[] encabezados;
170     for (int i = 0; i < filas; ++i) {
171         delete[] datos[i];
172     }
173     delete[] datos;
174 }
175
176 int main() {
177     string* encabezados = nullptr; // Arreglo dinámico para los
178                                     encabezados
179     string** datos = nullptr; // Arreglo dinámico para los
180                                 datos
181
182     int filas = 0, columnas = 0; // Número de filas y columnas
183     string nombreArchivo = "data_medidor.csv"; // Nombre del archivo CSV
184
185     // Leer el archivo automáticamente al iniciar
186     leerArchivoCSV(nombreArchivo, encabezados, datos, filas, columnas);
187     cout << "Archivo_leido_correctamente.\n";
188
189     while (true) {
190         string clienteID;
191         cout << "Ingrese_el_numero_de_cliente_(o_escriba_'salir'_para_terminar):_";
192         getline(cin, clienteID);
193
194         if (clienteID == "salir") {
195             cout << "Saliendo_del_programa...\n";
196             break;
197         }
198
199         // Buscar cliente usando Jump Search
200         buscarClienteJumpSearch(datos, encabezados, filas - 1, columnas,
201                                 clienteID);
202     }
```

```
199 // Liberar memoria asignada dinamicamente
200 liberarMemoria(encabezados, datos, filas - 1);
201
202 return 0;
203 }
204 }
```

```
163 cout << "Tiempo de busqueda: " << tiempo.count() << " ms\n";
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
```



The screenshot shows a Windows command prompt window titled "D:\Ruth U\algoritmos\progra...". The output of the program is as follows:

```
Fila 298: 1403-220034, PACORI MARRON SATURNINO, SANTIAGO, 20 DE ENERO S/N, Monofasico
, 508014326, 1083, 1098, 1105, 1124, 1130, 1133, 1143, 1148, 1161, 1189, 1214
Fila 299: 2209-010001, LOCAL MULTIUSO JUAN VELAZCO, JUAN VELAZCO A., CONCHA VELARDE S
/N, Monofasico, 608883386, 0, 0, 0, 0, 0, 0, 6, 6, 7, 17, 18
Archivo leído correctamente.
Ingrese el numero de cliente (o escriba 'salir' para terminar): 1403-210098
Cliente encontrado:
CLIENTE: 1403-210098
NOMBRES: CONDORI LARICO JESUS
BARRIO: LLALLAHUA
DIRECCION: SAN JOSE S/N
CONEXCION: Monofasico
MEDIDOR: 1112197
ENE4: 1974
FEB5: 1979
MAR6: 1985
ABR7: 1998
MAY8: 2008
JUN9: 2023
JUL10: 2034
AGO11: 2045
SET12: 2055
OCT13: 2066
NOV12: 2079
En 11 meses gasto 22246 watts de energia
En el mes de NOV12 gasto la mayor cantidad de energia electrica, que es de 2079 watts
Tiempo de busqueda: 10.0721 ms
Numero de iteraciones: 18
Ingrese el numero de cliente (o escriba 'salir' para terminar): |
```

Figura 5: Busqueda por saltos

Comparación (5 tipos de algoritmos de búsqueda)

En el programa hecho en c++ hicimos un código que lea el archivo csv y después que busque al cliente con el ID 1403-210098 y nos muestre en pantalla todos los datos que se encuentran en el archivo que son: Nombre completo, barrio, dirección, tipo de conexión, medidor y el gasto de energía que hizo en cada uno de los meses de enero a noviembre, seguidamente en el código implementamos que realice la suma total del gasto de energía eléctrica de los 11 meses y también que nos muestre en pantalla en cuál de los meses de enero a noviembre gastó más energía eléctrica.

Los algoritmos utilizados en esta búsqueda fueron: Lineal, binomial, índice, hash y saltos.

Tamaño de datos

El archivo CSV procesado en los cinco tipos de búsqueda tiene las siguientes características:

- En bytes : 40,417 bytes.
- En kilobytes (KB) : aproximadamente 40,42 KB.
- En megabytes (MB) : aproximadamente 0,0385 MB.
- Número de filas : 300 (incluido el encabezado).
- Número de columnas : 17

Tiempo de búsqueda

Nombre del algoritmo	Tiempo
Búsqueda lineal	9.4756 ms
Búsqueda binomial	9.1139 ms
Búsqueda por índice	14.696 ms
Búsqueda por hash	6.5314 ms
Búsqueda por saltos	10.0721 ms

Cuadro 1:

Número de saltos (Iteraciones)

Nombre del algoritmo	Iteraciones
Búsqueda lineal	176
Búsqueda binomial	8
Búsqueda por índice	176
Búsqueda por hash	29
Búsqueda por saltos	18

Cuadro 2:

Complejidad

1. Búsqueda lineal

La búsqueda lineal tiene una complejidad de $O(n)$, donde n es el número de elementos en la lista. En el peor de los casos, el algoritmo revisa todos los elementos uno por uno. En el mejor caso, si el elemento está al principio, la complejidad es $O(1)$ y peor caso es si el elemento está al final de la lista o no está presente, se tendrá que recorrer toda la lista, lo que lleva a una complejidad de $O(n)$.

Consiste en verificar cada elemento de una lista secuencialmente desde el inicio hasta el final, comparando cada uno con el valor buscado. Si se encuentra el valor, el algoritmo termina. Si no, continúa hasta el final de la lista.

2. Búsqueda binomial

El algoritmo de búsqueda binaria tiene una complejidad temporal de $O(\log n)$, donde n es el número de elementos en la lista. En el mejor caso, es $O(1)$ si el valor está en el medio, y en el peor caso, sigue dividiendo la lista hasta encontrar el valor o reducir el rango a cero.

La búsqueda binaria funciona dividiendo repetidamente el conjunto de datos ordenados a la mitad. En cada paso, compara el valor buscado con el valor en el punto medio y decide si continuar buscando en la mitad inferior o superior de la lista. Este proceso se repite hasta encontrar el valor o reducir el rango de búsqueda a cero.

3. Búsqueda por índice

El algoritmo de búsqueda por índice tiene una complejidad temporal de $O(1)$, ya que accede directamente al elemento mediante su índice en la lista o arreglo. En este tipo de búsqueda tanto el mejor caso como el peor caso tienen una complejidad de $O(1)$.

La búsqueda por índice consiste en acceder a un elemento específico de la lista utilizando su índice directamente, lo que permite obtener el valor en tiempo constante sin necesidad de recorrer la lista. Este tipo de búsqueda es posible en estructuras de datos como arreglos o listas indexadas.

4. Búsqueda por hash

El algoritmo de búsqueda por hash tiene una complejidad temporal promedio de $O(1)$, pero en el peor caso, puede ser $O(n)$, dependiendo de cómo se manejen las colisiones. El mejor caso $O(1)$ es cuando no hay colisiones, ya que el acceso a la posición en la tabla es directo y en el peor caso $O(n)$ es cuando muchas claves se asignan a la misma posición, lo que puede requerir recorrer todos los elementos en esa ubicación.

La búsqueda de hash utiliza una función hash para convertir el valor buscado en un índice en una tabla hash. A continuación, acceda a esa posición directamente. Si dos elementos tienen el mismo índice (colisión), se deben resolver mediante técnicas como encadenamiento o direccionamiento abierto.

5. Búsqueda por saltos

El algoritmo de búsqueda por saltos tiene una complejidad de $O(n)$, donde n es el número de elementos en la lista. Este algoritmo es útil para listas ordenadas y combina la búsqueda lineal y la búsqueda binaria. En el mejor caso $O(1)$ es cuando el elemento buscado se encuentra en el primer bloque o al principio del bloque donde se hace la búsqueda lineal y en el

peor caso $O(n)$ es cuando el elemento se encuentra al final de la lista o en el último bloque, lo que requiere recorrer los bloques y luego hacer una búsqueda lineal.

La búsqueda por saltos divide la lista en bloques de tamaño n . Primero, se salta por bloques hasta encontrar un bloque que contenga el elemento buscado. Luego, dentro de ese bloque, se realiza una búsqueda lineal para encontrar el elemento específico. Este enfoque reduce la cantidad de elementos que se tienen que recorrer en comparación con la búsqueda lineal, pero no tanto como la búsqueda binaria.

Conclusión

Al analizar los cinco algoritmos de búsqueda implementados sobre un archivo CSV con 300 registros, podemos concluir lo siguiente en cuanto a tiempo de búsqueda y número de iteraciones:

Tiempo de Búsqueda:

- Búsqueda por Hash es la más rápida, con 6.5314 ms.
- Búsqueda Binomial sigue de cerca con 9.1139 ms.
- Búsqueda Lineal tiene un tiempo de 9.4756 ms, ligeramente más lento que la binomial.
- Búsqueda por Saltos es un poco más lenta con 10.0721 ms.
- Búsqueda por Índice es la más lenta, con 14.696 ms.

Número de Iteraciones:

- Búsqueda Binomial tiene solo 8 iteraciones, lo que la hace la más eficiente en términos de número de pasos.
- Búsqueda por Saltos realiza 18 iteraciones, que es bastante eficiente.
- Búsqueda por Hash hace 29 iteraciones, lo cual es razonable para su tiempo de búsqueda.
- Búsqueda Lineal y Búsqueda por Índice realizan 176 iteraciones cada una, siendo las más ineficientes en términos de número de pasos.

Con esta comparación llegamos a la conclusión de que, para este tipo de tarea, donde la base de datos es relativamente pequeña (300 registros), la búsqueda por Hash es la mejor opción debido a su menor tiempo de ejecución y un número de iteraciones moderado. Sin embargo, en bases de datos mucho más grandes, la búsqueda Binomial es más adecuada debido a su capacidad de reducir el número de iteraciones de manera significativa con una complejidad de $O(\log n)$.

Referencias

- [1] Algoritmos de búsqueda. (s. f.). Recuperado de <http://artemisa.unicauca.edu.co/~nediaz/EDDI/cap02.htm>
- [2] Khan Academy. (s. f.). Recuperado de <https://es.khanacademy.org/computing/computer-science/algorithms/binary-search/a/binary-search>
- [3] Tejada, R. (2024, 17 mayo). Jump Search - Algoritmos de Búsqueda - Aviada's Insights Hub. *Aviada's Insights Hub*. Recuperado de <https://blog.aviada.mx/es/jump-search/>
- [4] ALGORITMIA ALGO+ - Algoritmos y estructuras de datos. (s. f.). Recuperado de <http://www.algoritmia.net/articles.php?id=32>
- [5] Ďurišinová, M. (2023, 29 mayo). 6 Tipos de Algoritmos de Búsqueda Que Debes Conocer - Luigi's Box. *Luigi's Box*. Recuperado de <https://www.luigisbox.es/blog/tipos-de-algoritmos-de-busqueda/>