

Tarea de Medición de Tiempo en C++

Lipa Quispe Alex Dannis

November 22, 2024

Descripción del Código

La principal estructura del código se basa en abrir un archivo ‘.txt’ para luego almacenar sus datos en un arreglo.

Una vez guardados estos valores, el código encargado de ordenar el arreglo es **QuickSort**, lo que permite buscar los números de manera más rápida y eficiente.

Posteriormente, el programa solicita al usuario la cantidad de números que desea buscar. Una vez ingresados estos valores, el programa identifica en qué posición se encuentra cada número y el tiempo que demoró en encontrarlo. Finalmente, compara los tiempos de búsqueda, mostrando en pantalla los números con los tiempos máximos y mínimos.

Código del Programa

```

1  #include <iostream>
2  #include <fstream>
3  #include <vector>
4  #include <cmath>
5  #include <chrono>
6  #include <iomanip>
7  #include <limits>
8  using namespace std;
9
10 // Jump Search
11 int jumpSearch(const vector<float>& arr, float x, int n) {
12     int paso = sqrt(n), prev = 0;
13
14     while (arr[min(paso, n) - 1] < x) {
15         prev = paso;
16         paso += sqrt(n);
17         if (prev >= n) return -1;
18     }
19
20     while (arr[prev] < x) {
21         prev++;
22         if (prev == min(paso, n)) return -1;
23     }
24
25     return arr[prev] == x ? prev : -1;
26 }
27
28 // QuickSort

```

Figure 1: Función que se encarga de buscar los valores deseados.

```

28 // QuickSort
29 void quickSort(vector<float>& arr, int primero, int ultimo) {
30     if (primero >= ultimo) return;
31     int i = primero, j = ultimo;
32     float pivote = arr[(primero + ultimo) / 2];
33     while (i <= j) {
34         while (arr[i] < pivote) i++;
35         while (arr[j] > pivote) j--;
36         if (i <= j) swap(arr[i], arr[j]), i++, j--;
37     }
38     quickSort(arr, primero, j);
39     quickSort(arr, i, ultimo);
40 }
41
42 int main() {
43     // Abrir el archivo
44     ifstream archivo("C://doc//numeros_aleatorios.txt");
45     if (!archivo) {
46         cout << "Error al abrir el archivo." << endl;
47         return 1;
48     }
49
50     vector<float> arr;
51     for (string linea; getline(archivo, linea); arr.push_back(stof(linea)));
52     archivo.close();
53
54     // Ordenar el vector
55     quickSort(arr, 0, arr.size() - 1);

```

Figure 2: Función que ordena los datos del archivo utilizando QuickSort.

```

54 // Ordenar el vector
55 quickSort(arr, 0, arr.size() - 1);
56
57 // Solicitar la cantidad de números a buscar
58 int n;
59 cout << "Cuantos numeros deseas buscar? ";
60 cin >> n;
61
62 vector<float> numeros_a_buscar(n);
63 cout << "Introduce " << n << " numeros a buscar:\n";
64 for (int i = 0; i < n; ++i) {
65     cin >> numeros_a_buscar[i];
66 }
67
68 // Inicializar variables para tiempos mínimo y máximo
69 double tiempo_minimo = numeric_limits<double>::max();
70 double tiempo_maximo = numeric_limits<double>::lowest();
71 float num_minimo = 0, num_maximo = 0;
72
73 int repeticiones = 1000; // Repetir cada búsqueda para calcular tiempos promedio
74
75 for (int i = 0; i < n; ++i) {
76     float x = numeros_a_buscar[i];
77     double total_tiempo = 0;
78
79     for (int j = 0; j < repeticiones; ++j) {
80         auto start_time = chrono::high_resolution_clock::now();
81         jumpSearch(arr, x, arr.size());
82         auto end_time = chrono::high_resolution_clock::now();

```

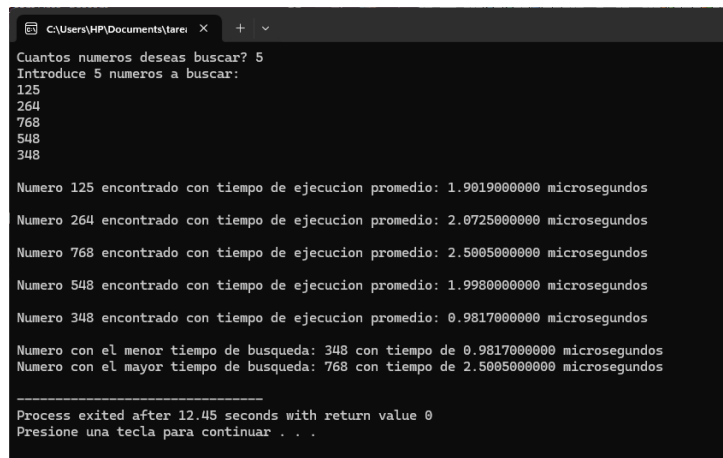
Figure 3: Estructura principal del programa llamando las funciones necesarias.

```

82     auto end_time = chrono::high_resolution_clock::now();
83     chrono::duration<double, std::micro> execution_time_micro = end_time - start_time;
84     total_tiempo += execution_time_micro.count();
85 }
86 // Calcular tiempo promedio
87 double tiempo_promedio = total_tiempo / repeticiones;
88
89 // Actualizar tiempos mínimos y máximos
90 if (tiempo_promedio < tiempo_minimo) {
91     tiempo_minimo = tiempo_promedio;
92     num_minimo = x;
93 }
94 if (tiempo_promedio > tiempo_maximo) {
95     tiempo_maximo = tiempo_promedio;
96     num_maximo = x;
97 }
98 // Mostrar resultados para cada número
99 cout << "\nNumero " << fixed << setprecision(0) << x << " encontrado con tiempo de ejecucion promedio: "
100 << fixed << setprecision(10)
101 << tiempo_promedio << " microsegundos" << endl;
102 }
103 // Mostrar resultados finales
104 cout << "\nNumero con el menor tiempo de busqueda: " << fixed << setprecision(0) << num_minimo
105 << " con tiempo de " << fixed << setprecision(10) << tiempo_minimo << " microsegundos" << endl;
106
107 cout << "Numero con el mayor tiempo de busqueda: " << fixed << setprecision(0) << num_maximo
108 << " con tiempo de " << fixed << setprecision(10) << tiempo_maximo << " microsegundos" << endl;
109 return 0;
110 }

```

Figure 4: Parte final del código donde calcula y verifica el mayor y menor tiempo en hacer la búsqueda



```
C:\Users\HP\Documents\tare...
Cuantos numeros deseas buscar? 5
Introduce 5 numeros a buscar:
125
264
768
548
348

Numero 125 encontrado con tiempo de ejecucion promedio: 1.9019000000 microsegundos
Numero 264 encontrado con tiempo de ejecucion promedio: 2.0725000000 microsegundos
Numero 768 encontrado con tiempo de ejecucion promedio: 2.5005000000 microsegundos
Numero 548 encontrado con tiempo de ejecucion promedio: 1.9980000000 microsegundos
Numero 348 encontrado con tiempo de ejecucion promedio: 0.9817000000 microsegundos

Numero con el menor tiempo de busqueda: 348 con tiempo de 0.9817000000 microsegundos
Numero con el mayor tiempo de busqueda: 768 con tiempo de 2.5005000000 microsegundos

-----
Process exited after 12.45 seconds with return value 0
Presione una tecla para continuar . . .
```

Figure 5: El progrma se compilo correctamente

Una vez de haber escrito de manera ordenado y estructurada el código se procede a compilar el código: