Universidad de Guadalajara

Centro Universitario de Ciencias Exactas e Ingenierías

Alumnos:

Ávila Sanchez Carlos Humberto

Ruiz Hernández Oscar Alejandro

Sandoval Huerta Carlos Arturo

Análisis de algoritmos - D01

Entregable: Creación de Algoritmo y evaluación de tiempo

Introducción

El algoritmo de ordenamiento por selección es un algoritmo de ordenamiento simple y eficiente. En nuestro código se basa en lo siguiente:

1. Se incluyen las bibliotecas necesarias.

2. `llenarArreglo` es una función que llena un arreglo con números aleatorios. Utiliza la función `rand()` para generar números aleatorios.

3. `selectionSort` es la implementación del algoritmo de ordenamiento por selección.

4. En la función `main`, el programa realiza lo siguiente:

   - Itera a través de diferentes tamaños de arreglos (desde 5 hasta 1000 en incrementos de 50).

   - Para cada tamaño, crea un arreglo de ese tamaño y lo llena con números aleatorios.

   - Mide el tiempo que tarda el algoritmo `selectionSort` en ordenar el arreglo lleno.

   - Imprime el arreglo ordenado y el tiempo que tomó la ordenación.

   - Espera a que el usuario presione Enter antes de continuar con el siguiente tamaño de arreglo.

El código está diseñado para analizar cómo el tiempo de ejecución del algoritmo de ordenamiento por selección varía según el tamaño del arreglo.

Objetivo

Hacer un algoritmo referente de fuerza bruta, en el lenguaje Python o C, C + +.

Se debe de evaluar el tiempo de ejecución de dicho programa con una entrada de valores de N = 5, incrementando de 50 en 50 (N = N +50) hasta N =1000.

Se deben de obtener los resultados del tiempo y debe realizar una gráfica donde muestra los valores de N con respecto al tiempo (t).

El objetivo particular es analizar un problema de algoritmo de fuerza bruta con el fin de ver la eficiencia y tiempo que muestra este.

El objetivo general es analizar y comparar diversos algoritmos de nuestra clase para así aclarar y comprender las propiedades que buscan y necesitan los algoritmos.

Desarrollo

El algoritmo se hizo tomando en cuenta que se tenía que presentar un ejemplo de fuerza bruta, entonces partiendo de eso las dos formas más fáciles serían un Bubble Sort, Insertion Sort o Selection Sort, así que realizamos el de Selection.

Primero, para que el arreglo estuviera lleno en cada iteración, aumentando de 50 en 50, utilizamos la librería <chrono> de c++, esto más la función srand() y rand() nos daban estos números aleatorios cada que se llamara a la función.

Después la función de selectionSort recibe el arreglo y lo acomoda, el tiempo de ejecución se toma antes de llamar a esta función y se termina después de que este termine de acomodar los elementos.

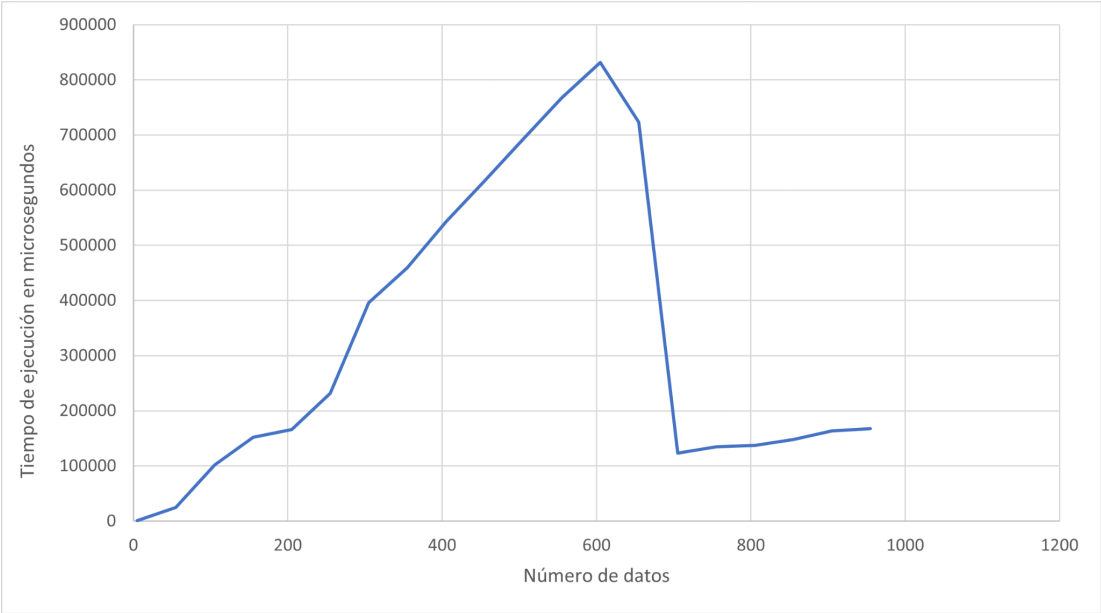
Para el usuario se utiliza dando solamente enters, para que el programa aumente en 50 los datos a ordenar, y al final de esto se da el tiempo en milisegundos.

```
#include <iostream>
#include <time.h>
#include <math.h>
#include <chrono>

using namespace std;

void llenarArreglo(int tamanioArreglo,int arreglo[]){
    srand((unsigned)time(0));
    int aleatorio;
    for(int i=0;i<tamanioArreglo;i++){
        aleatorio=rand();
        arreglo[i]=aleatorio;
    }
}

void selectionSort(int tamanioArreglo,int arreglo[]){
    int i,j,menor;
    for(i=0;i<tamanioArreglo-1;++i){
    menor=i;
        for(j=i+1;j<tamanioArreglo;++j){
            if (arreglo[j]<arreglo[menor]){
                menor=j;
```

```
            }
        }
        int aux=arreglo[i];
        arreglo[i]=arreglo[menor];
        arreglo[menor]=aux;
    }
}

int main()
{
    int tamanioArreglo;
    for(int i=5;i<=1000;i+=50){
        tamanioArreglo=i;
        int arreglo[tamanioArreglo];
        llenarArreglo(tamanioArreglo,arreglo);
        auto startTime = std::chrono::high_resolution_clock::now();
        selectionSort(tamanioArreglo,arreglo);
        for(int i=0;i<tamanioArreglo;i++){
        cout << endl << endl << arreglo[i] << " ";
        }
        auto endTime = std::chrono::high_resolution_clock::now();
        auto duracion =
std::chrono::duration_cast<std::chrono::microseconds>
        (endTime - startTime).count();
        std::cout << "Duracion: " << duracion << "\n";
        cout << "Presiona enter" << endl;
        cin.get();
    }
    return 0;
}
```

Gráfica de eficiencia

T	N
994	5
24413	55
102243	105
151836	155
165792	205
231850	255
396385	305
459674	355
543301	405
617079	455
692400	505
767195	555
831299	605
723215	655
123577	705
134416	755
137599	805
147592	855
163235	905
167795	955



Conclusiones

Ávila Carlos:

Con este trabajo aprendimos a medir la eficiencia de un algoritmo y a cómo representarla de una manera visual, esto con la finalidad de comparar nuestra solución con otras y ver en qué situaciones un algoritmo es mejor que otro y por qué. Como conclusión, pienso que es una buena actividad para comenzar, muy útil no solo para la carrera, sino para el entorno laboral.

Ruiz Oscar:

Esta tarea me mostró que para el desarrollo y buena implementación de un algoritmo se necesita de una característica base, “el tiempo”, ya que con este se demuestra como es tan eficiente un algoritmo y a

cuando estuvimos en desarrollo de la graficación de este problema se puede analizar el fenómeno y comportamiento del tiempo que tarda en el acomodo de los datos.

Sandoval Carlos:

Esta práctica nos mostró cómo la eficiencia de los algoritmos afecta el tiempo de ejecución y en consecuencia, el gasto de energía, en programas pequeños esto casi no se nota, pero en programas grandes, esto ya toma una gran importancia; es por eso que se hacen gráficas para analizar estos comportamientos.

Referencias

Alfred V. Aho, John E. Hopcroft, Jefrey D. Ullman con la colaboración de Guillermo Levine Gutiérrez; versión en español de Américo Vargas y Jorge Lozano PUBLICACIÓN México, DF : Addison-Wesley Iberoamericana: Sistemas Técnicos de Edición, 1988