

UNIVERSIDAD TECNOLÓGICA DE SANTIAGO
SISTEMA CORPORATIVO
FACULTAD DE INGENIERÍA Y ARQUITECTURA
CARRERA DE SISTEMAS COMPUTACIONALES



ALGORITMOS PARALELOS

GRUPO 3

Presentado a:

Iván Mendoza

Presentado por:

Sugeiri Torres 1-16-0736

Dinnibel Azcona 1-16-0788

Claudia Báez 2-16-1116

Santiago de los Caballeros,
República Dominicana

INDICE

PAG.

1. Introducción.....	Error! Bookmark not defined.
2. Descripción del Proyecto.....	Error! Bookmark not defined.
3. Objetivos.....	Error! Bookmark not defined.
3. 1. Objetivo General.....	Error! Bookmark not defined.
3. 2. Objetivos Específicos.....	Error! Bookmark not defined.
4. Definición de Algoritmos Paralelos.....	Error! Bookmark not defined.
5. Etapas de los Algoritmos paralelos.....	Error! Bookmark not defined.
5. 1. Partición.....	Error! Bookmark not defined.
5. 2. Comunicación.....	Error! Bookmark not defined.
5. 3. Agrupamiento.....	Error! Bookmark not defined.
5. 4. Asignación.....	Error! Bookmark not defined.
6. Técnicas Algorítmicas Paralelas.....	Error! Bookmark not defined.
6. 1. Divide y Conquistarás.....	Error! Bookmark not defined.
6. 2. Aleatorización.....	Error! Bookmark not defined.
6. 3. Técnicas de Puntero Paralelo.....	Error! Bookmark not defined.
7. Modelos de Algoritmos Paralelos.....	Error! Bookmark not defined.
7. 1. Paralelismo de datos.....	Error! Bookmark not defined.
7. 2. Grafo de tareas.....	Error! Bookmark not defined.
7. 3. Conjunto de trabajadores (work pool).....	Error! Bookmark not defined.
7. 4. Maestro-esclavo (master-slave).....	Error! Bookmark not defined.
7. 5. Productor-consumidor (pipeline producer-consumer).....	Error! Bookmark not defined.
8. Algoritmos de Búsquedas y Ordenamiento (Adjuntar Pseudocódigo y código de cada uno).....	4
8. 1. Búsqueda Secuencial.....	4
8. 2. Búsqueda Binaria.....	5
8. 3. Algoritmo de Ordenamiento de la Burbuja.....	7

8. 4. Quick Sort.....	7
8. 5. Método de Inserción.....	9
9. Programa desarrollado.....	Error! Bookmark not defined.
9. 1. Explicación de su funcionamiento.....	Error! Bookmark not defined.
9. 2. Fotos de la aplicación.....	Error! Bookmark not defined.
9. 3. Link de GitHub y Ejecutable de la aplicación.....	Error! Bookmark not defined.
9. 4. Resultados (Tiempo en terminar los ordenamientos y búsqueda de cada algoritmo).....	Error! Bookmark not defined.
10. ¿Qué tanta memoria se consumió este proceso?.....	Error! Bookmark not defined.
11. ¿Cuál fue el algoritmo que realizo la búsqueda y el ordenamiento más rápido? Explique.....	Error! Bookmark not defined.
12. Conclusión.....	Error! Bookmark not defined.
13. Bibliografías.....	Error! Bookmark not defined.

1. Algoritmos de Búsquedas y Ordenamiento

1.1. Búsqueda Secuencial

Consiste en tomar un dato clave que identifica al elemento que se busca y hacer un recorrido a través de todo el arreglo comparando el dato de referencia con el dato de cada posición.

```
int[] ii_arreglo = new int[15] { 9, 15, 1, 3, 98, 23, 76, 7, 29, 67, 4, 45, 87, 34, 72 };

int Tamaño = ii_arreglo.Length;
int Posicion = 0;

while (Posicion < Tamaño)
{
    if (ii_arreglo[Posicion] == num)
    {
        encontrado_bs = "Si";
        stopwatch_bs.Stop();
        arreglo_bs = ii_arreglo;
        return ms;
    }
    else
    {
        Posicion++;
    }
}

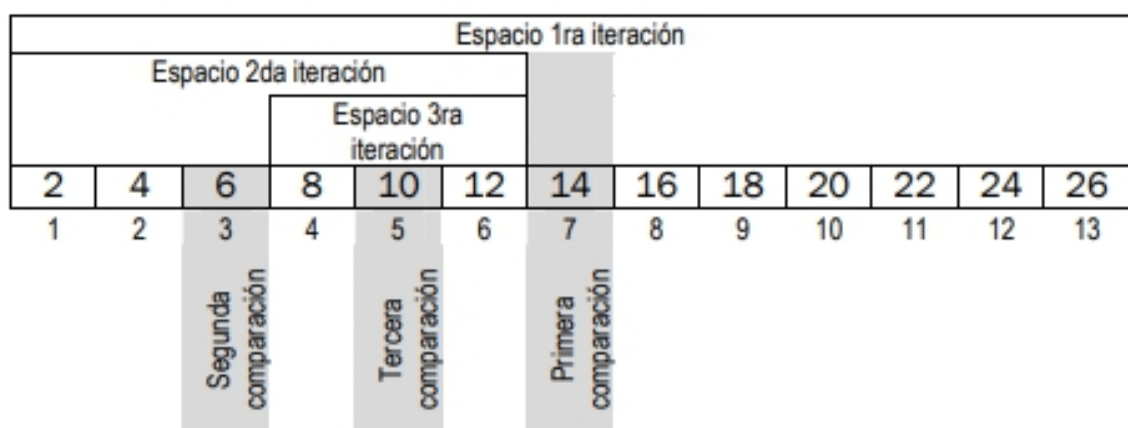
encontrado_bb = "No";
```

1. 2. **Búsqueda Binaria**

es más eficiente que la búsqueda secuencial pero sólo se puede aplicar sobre vectores o listas de datos ordenados.

En la búsqueda binaria no se hace un recorrido de principio a fin, sino que se delimita progresivamente el espacio de búsqueda hasta llegar al elemento buscado. La primera comparación se hace con el elemento de la mitad del arreglo, si aquel no es el dato buscado, se decide si buscar en la mitad inferior o en la mitad superior según la clave sea menor o mayor del elemento de la mitad. Se toma como espacio de búsqueda la mitad del vector que corresponda y se procede de igual forma, se compara con el elemento del centro, si ese no es el que se busca, se toma un nuevo espacio de búsqueda correspondiente a la mitad inferior o superior del espacio anterior, se compara nuevamente con el elemento del centro, y así sucesivamente hasta que se encuentre el elemento o el espacio de búsqueda se haya reducido un elemento.

Figura 114. Diagrama de flujo del algoritmo de búsqueda lineal



```
int[] ii_arreglo = new int[15] { 9, 15, 1, 3, 98, 23, 76, 7, 29, 67, 4, 45, 87, 34, 72 };
```

```
    int t;
    for (int a = 1; a < ii_arreglo.Length; a++)
        for (int b = ii_arreglo.Length - 1; b >= a; b--)
        {
            if (ii_arreglo[b - 1] > ii_arreglo[b])
            {
                t = ii_arreglo[b - 1];
                ii_arreglo[b - 1] = ii_arreglo[b];
                ii_arreglo[b] = t;
            }
        }
```

```
int l = 0, h = 14;
int m = 0;
bool found = false;
while (l <= h && found == false)
{
    m = (l + h) / 2;
    if (ii_arreglo[m] == num)
        found = true;
    if (ii_arreglo[m] > num)
        h = m - 1;
    else
        l = m + 1;
}
if (found == false)
    encontrado_bb = "No";
else
    encontrado_bb = "Si";
```

1. 3. **Algoritmo de Ordenamiento de la Burbuja**

La idea consiste en realizar varios recorridos secuenciales en el arreglo intercambiando los elementos adyacentes que estén desordenados. en la primera iteración se lleva el máximo a su posición definitiva al final del arreglo. En la segunda iteración se lleva el segundo máximo a su posición definitiva. Y así continúa hasta ordenar todo el arreglo.

```
int[] ii_arreglo = new int[15] { 9, 15, 1, 3, 98, 23, 76, 7, 29, 67, 4, 45, 87, 34, 72 };

int t;
for (int a = 1; a < ii_arreglo.Length; a++)
    for (int b = ii_arreglo.Length - 1; b >= a; b--)
    {
        if (ii_arreglo[b - 1] > ii_arreglo[b])
        {
            t = ii_arreglo[b - 1];
            ii_arreglo[b - 1] = ii_arreglo[b];
            ii_arreglo[b] = t;
        }
    }
```

1. 4. **Quick Sort**

El algoritmo trabaja de la siguiente forma:

Elegir un elemento del conjunto de elementos a ordenar, al que llamaremos pivote. Re-situar los demás elementos de la lista a cada lado del pivote, de manera que a un lado queden todos los menores que él, y al otro los mayores. Los elementos iguales al pivote pueden ser colocados tanto a su derecha como a su izquierda, dependiendo de la implementación deseada. En este momento, el pivote ocupa exactamente el lugar que le corresponderá en la lista ordenada.

La lista queda separada en dos sub listas, una formada por los elementos a la izquierda del pivote, y otra por los elementos a su derecha.

Repetir este proceso de forma recursiva para cada sub lista mientras éstas contengan más de un elemento. Una vez terminado este proceso todos los elementos estarán ordenados.

```
private int[] QuickSort(int[] ii_arreglo, int primero, int ultimo)
{
    int i, j, central;
    double pivote;
    central = (primero + ultimo) / 2;
    pivote = ii_arreglo[central];
    i = primero;
    j = ultimo;
    do
    {
        while (ii_arreglo[i] < pivote) i++;
        while (ii_arreglo[j] > pivote) j--;
        if (i <= j)
        {
            int temp;
            temp = ii_arreglo[i];
            ii_arreglo[i] = ii_arreglo[j];
            ii_arreglo[j] = temp;
            i++;
            j--;
        }
    } while (i <= j);

    if (primero < j)
    {
        ii_arreglo= QuickSort(ii_arreglo, primero, j);
    }
    if (i < ultimo)
    {
        ii_arreglo =QuickSort(ii_arreglo, i, ultimo);
    }
    return ii_arreglo;
}
```


1. 5. Método de Inserción

Es un algoritmo de fácil aplicación que permite el ordenamiento de una lista. Su funcionamiento consiste en el recorrido por la lista seleccionando en cada iteración un valor como clave y compararlo con el resto insertándolo en el lugar correspondiente.

```
int[] ii_arreglo = new int[15] { 9, 15, 1, 3, 98, 23, 76, 7, 29, 67, 4, 45, 87, 34, 72 };

    int auxili;
    int j;
    for (int i = 0; i < ii_arreglo.Length; i++)
    {
        auxili = ii_arreglo[i];
        j = i - 1;
        while (j >= 0 && ii_arreglo[j] > auxili)
        {
            ii_arreglo[j + 1] = ii_arreglo[j];
            j--;
        }
        ii_arreglo[j + 1] = auxili;
    }
```