



Parallel and concurrent programming

# Estudio de Rendimiento: Quicksort en Paralelo y Secuencial

DOCENTE: JOSE ALFREDO HERRERA QUISPE

ALUMNO: FABRICIO MARIANO CRESPO MIRANDA



# ÍNDICE

- \*Acerca del algoritmo**
- \*Código(C++ )**
- \*Ejecución y Resultados**
- \*Conclusiones**
- \*Referencias**





## **ACERCA DEL ALGORITMO**



El algoritmo QuickSort está basado en el paradigma de "divide y vencerás". Este paradigma consiste en resolver un problema dividiéndolo en subproblemas más pequeños

El algoritmo de ordenamiento QuickSort es uno de los algoritmos más conocidos y utilizados para ordenar elementos en una lista o arreglo. Fue desarrollado por el científico británico Tony Hoare en 1960 mientras trabajaba en la computadora Moscow State University

### **Quick Sort de forma general:**

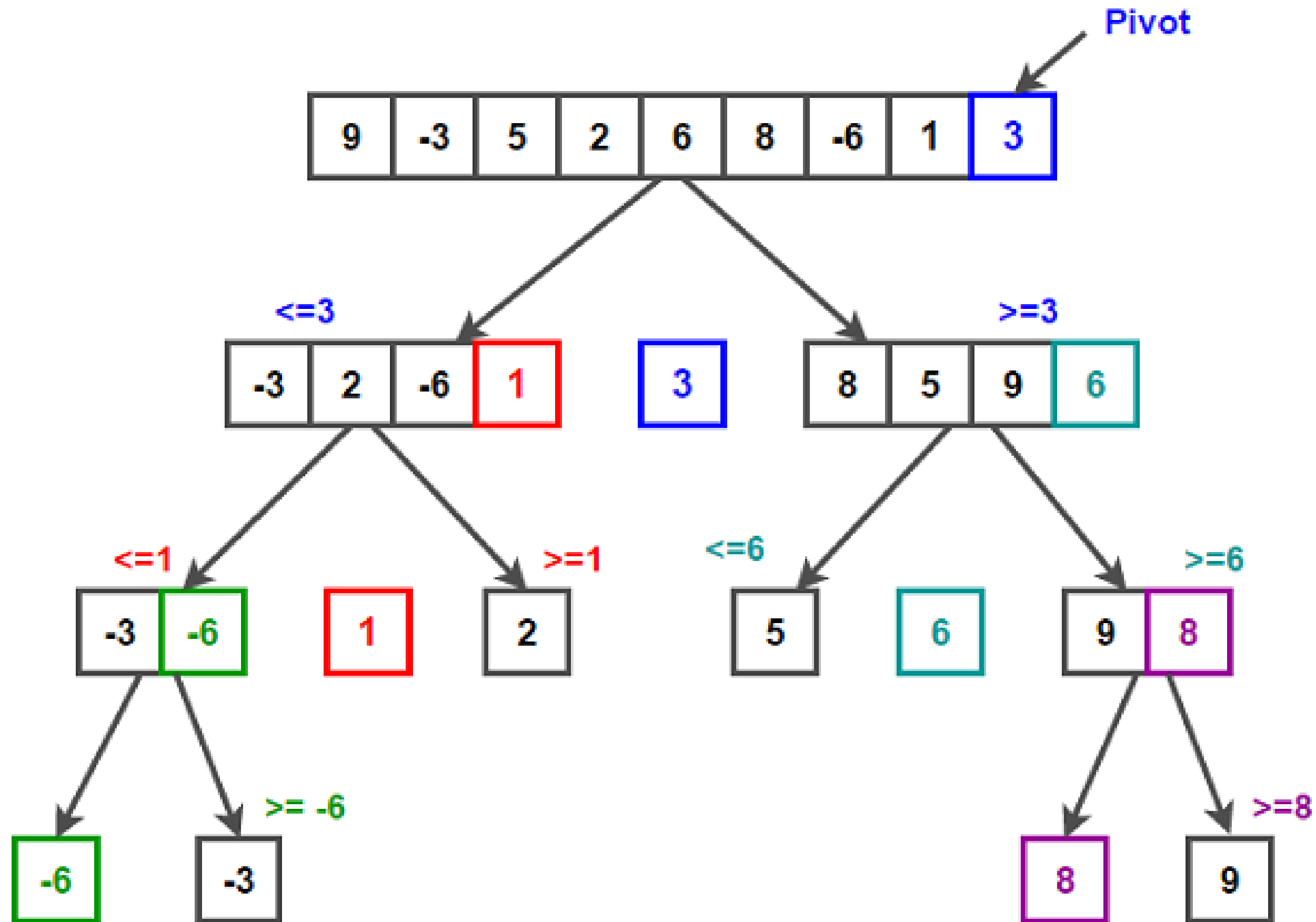
**Selección del elemento  
pivote**

**Mayores a la derecha y  
menores a la izquierda**

**Ordenar cada subarreglo  
recursivamente**



## ACERCA DEL ALGORITMO





# Código(C++)

## ➤ Función para la división del arreglo y encontrar el pivote

```
int partition(vector<string>& arr, int low, int high) {  
    string pivot = arr[high]; // Seleccionamos el último elemento como pivote  
    int i = low - 1; // Índice del elemento más pequeño  
  
    for (int j = low; j <= high - 1; j++) {  
        // Si el elemento actual es menor o igual al pivote  
        if (arr[j] <= pivot) {  
            i++; // Incrementamos el índice del elemento más pequeño  
            swap(arr, i, j);  
        }  
    }  
    swap(arr, i + 1, high);  
    return i + 1;  
}
```

# ➤ Quick Sort - SECUENCIAL

```
// Función principal de Quick Sort
void quickSort(vector<string>& arr, int low, int high) {
    if (low < high) {
        // Encuentra el índice del pivote
        int pi = partition(arr, low, high);

        // Ordena los elementos antes y después del pivote
        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
    }
}
```



# Quick Sort - PARALELO

```
// Quick Sort paralelo con profundidad máxima
void parallelQuickSort(vector<string>& arr, int low, int high, int depth) {
    if (low < high) {
        if (depth <= 0) {
            quickSort(arr, low, high); // Cambiar a QuickSort secuencial si se alcanza la profundidad máxima
            return;
        }
        int pi = partition(arr, low, high);
        thread left([&]() {
            parallelQuickSort(arr, low, pi - 1, depth - 1);
        });
        thread right([&]() {
            parallelQuickSort(arr, pi + 1, high, depth - 1);
        });
        left.join();
        right.join();
    }
}
```





# Ejecución y Resultados

● Ingrese la cantidad de palabras (frutas) a generar: 100000

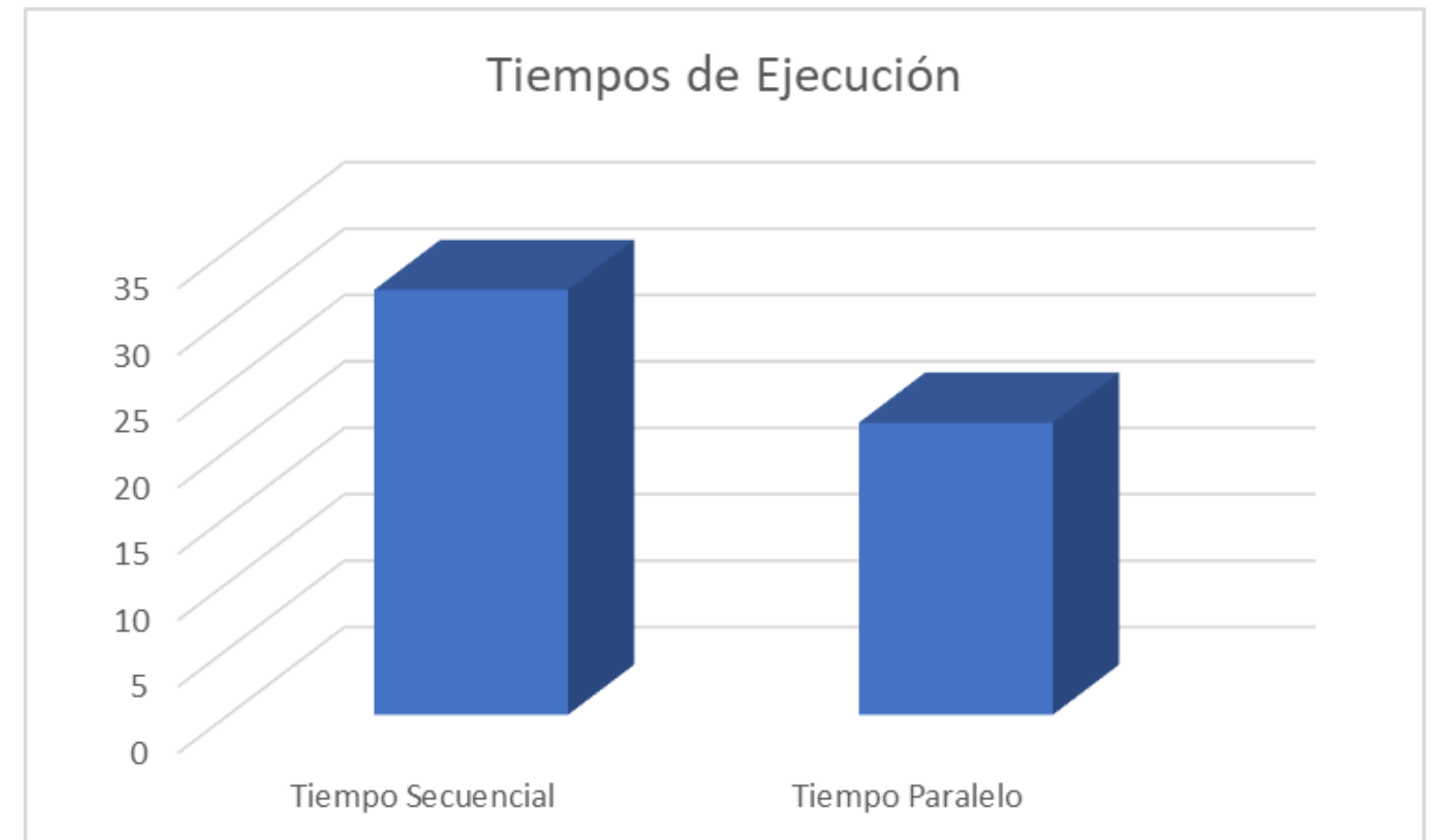
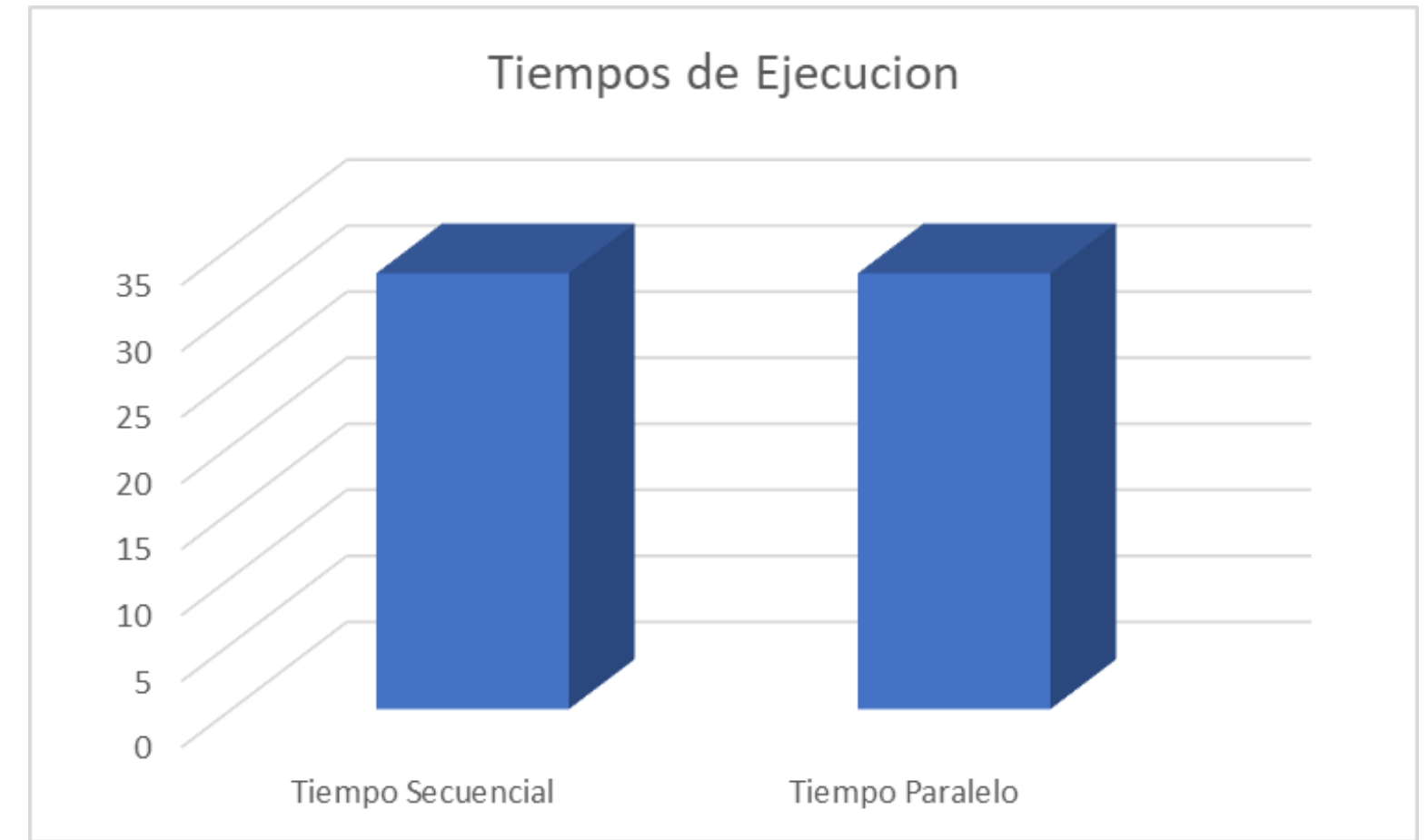
Tiempo de ejecución secuencial: 33 segundos

Tiempo de ejecución paralelo con profundidad máxima 1: 33 segundos

SPEEDUP: 1

EFICIENCIA: 12.5%

≥ 100,000



● PS C:\Users\Lenovo\Desktop\output> & .\ 'P\_2.exe'

Ingrese la cantidad de palabras (frutas) a generar: 100000

Tiempo de ejecución secuencial: 32 segundos

Tiempo de ejecución paralelo con profundidad máxima 5: 22 segundos

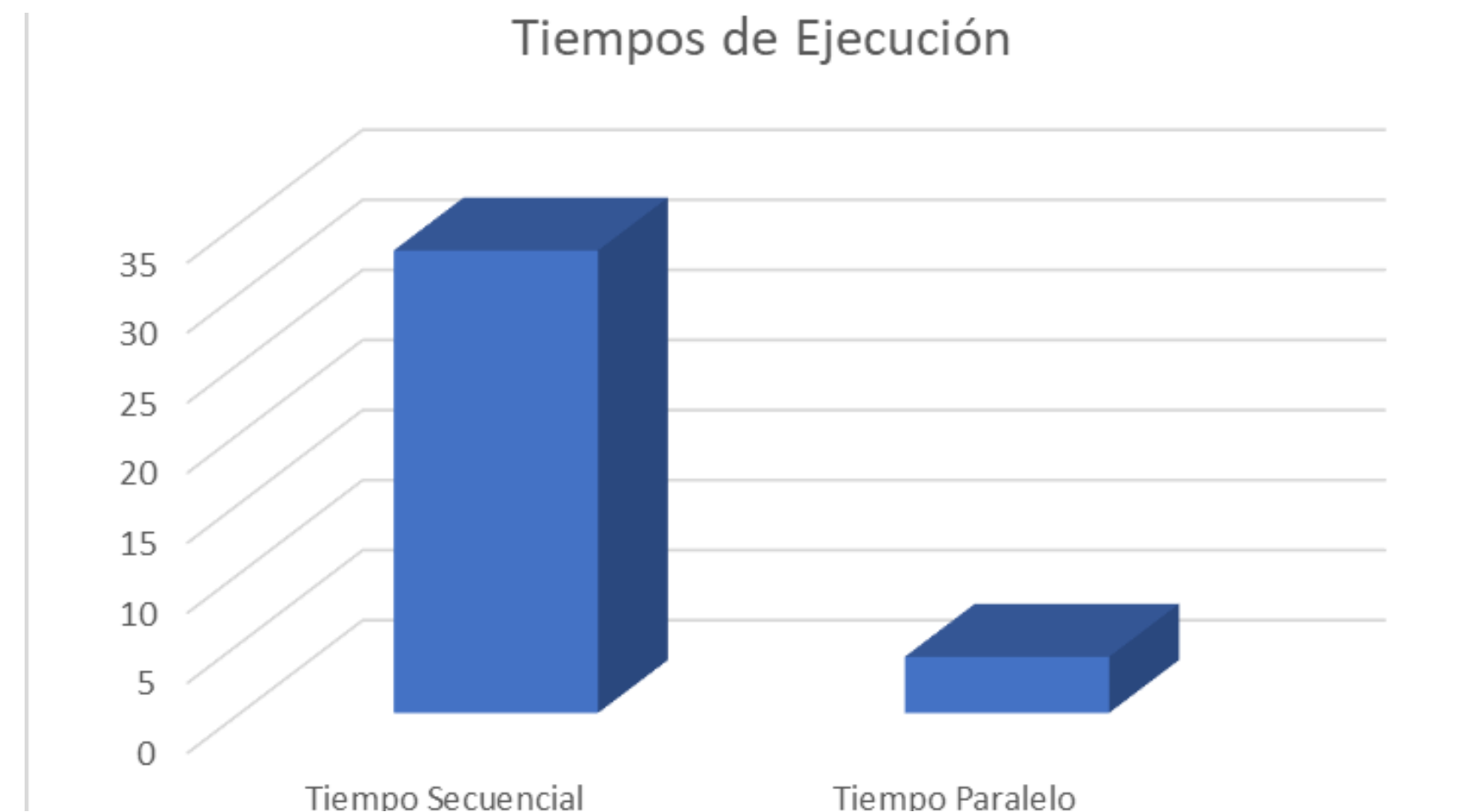
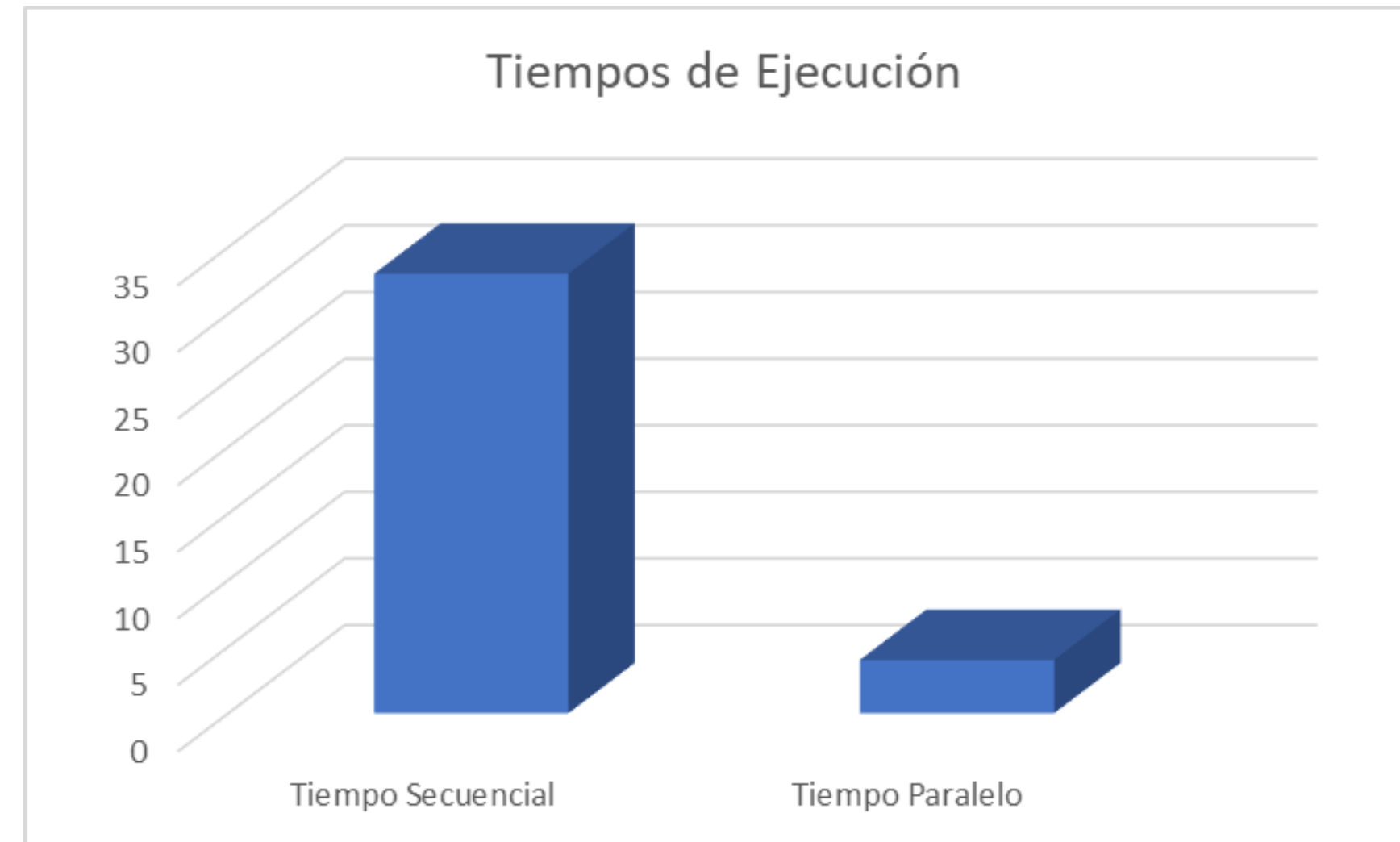
SPEEDUP: 1

EFICIENCIA: 12.5%

```
● PS C:\Users\Lenovo\Desktop\output> & .\'P_2.exe'  
  Ingrese la cantidad de palabras (frutas) a generar: 100000  
  
  Tiempo de ejecución secuencial: 33 segundos  
  
  Tiempo de ejecución paralelo con profundidad máxima 10: 4 segundos  
  
  SPEEDUP: 7  
  EFICIENCIA: 87.5%
```

≥ 100,000

```
● PS C:\Users\Lenovo\Desktop\output> & .\'P_2.exe'  
  Ingrese la cantidad de palabras (frutas) a generar: 100000  
  
  Tiempo de ejecución secuencial: 33 segundos  
  
  Tiempo de ejecución paralelo con profundidad máxima 16: 4 segundos  
  
  SPEEDUP: 7  
  EFICIENCIA: 87.5%
```



● Ingrese la cantidad de palabras (frutas) a generar: 200000

Tiempo de ejecución secuencial: 128 segundos

● Tiempo de ejecución paralelo con profundidad máxima 1: 121 segundos

SPEEDUP: 1

EFICIENCIA: 12.5%

➤ 200,000

● PS C:\Users\Lenovo\Desktop\output> & .\'P\_2.exe'

Ingrese la cantidad de palabras (frutas) a generar: 200000

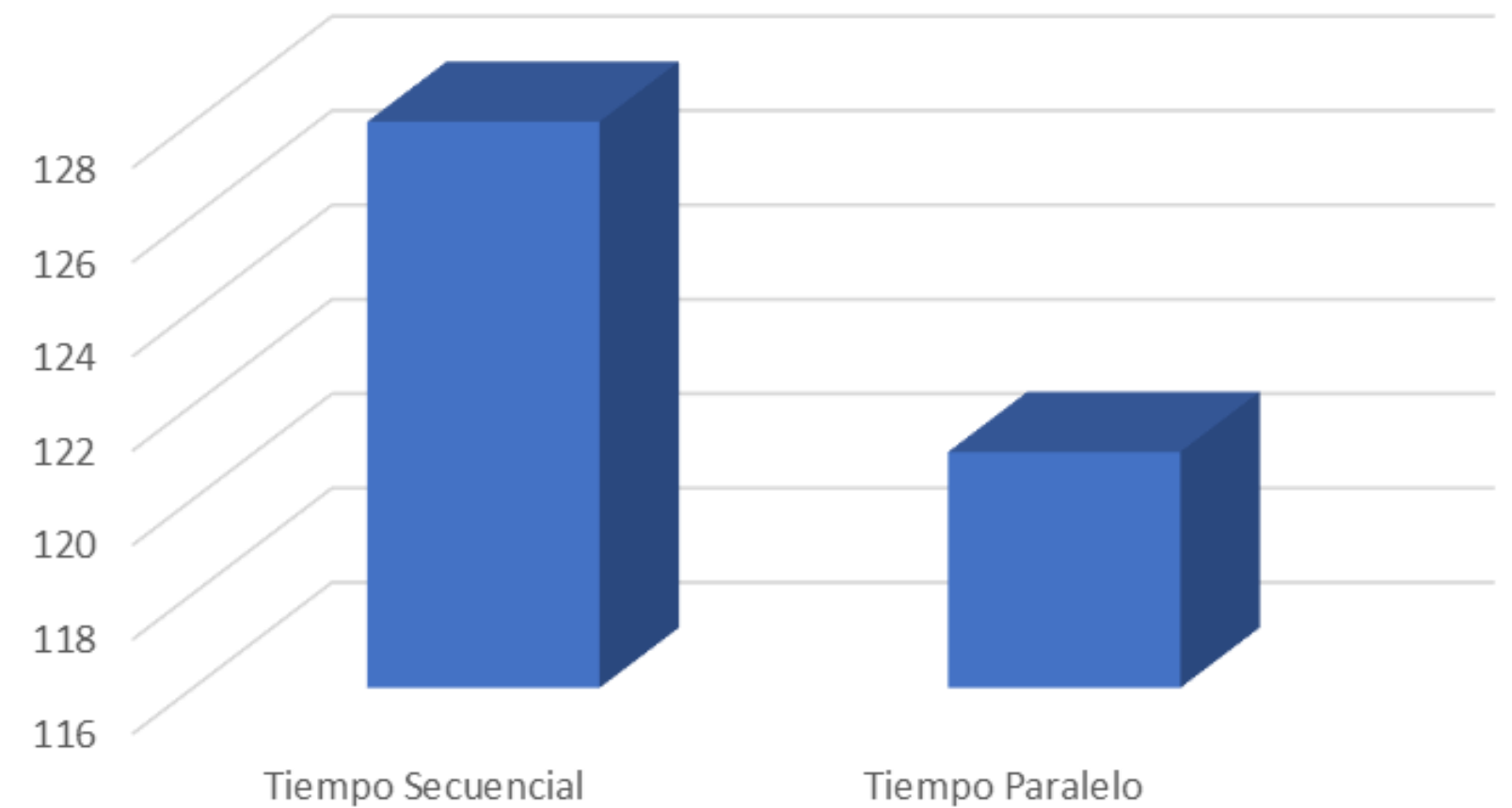
Tiempo de ejecución secuencial: 131 segundos

Tiempo de ejecución paralelo con profundidad máxima 5: 30 segundos

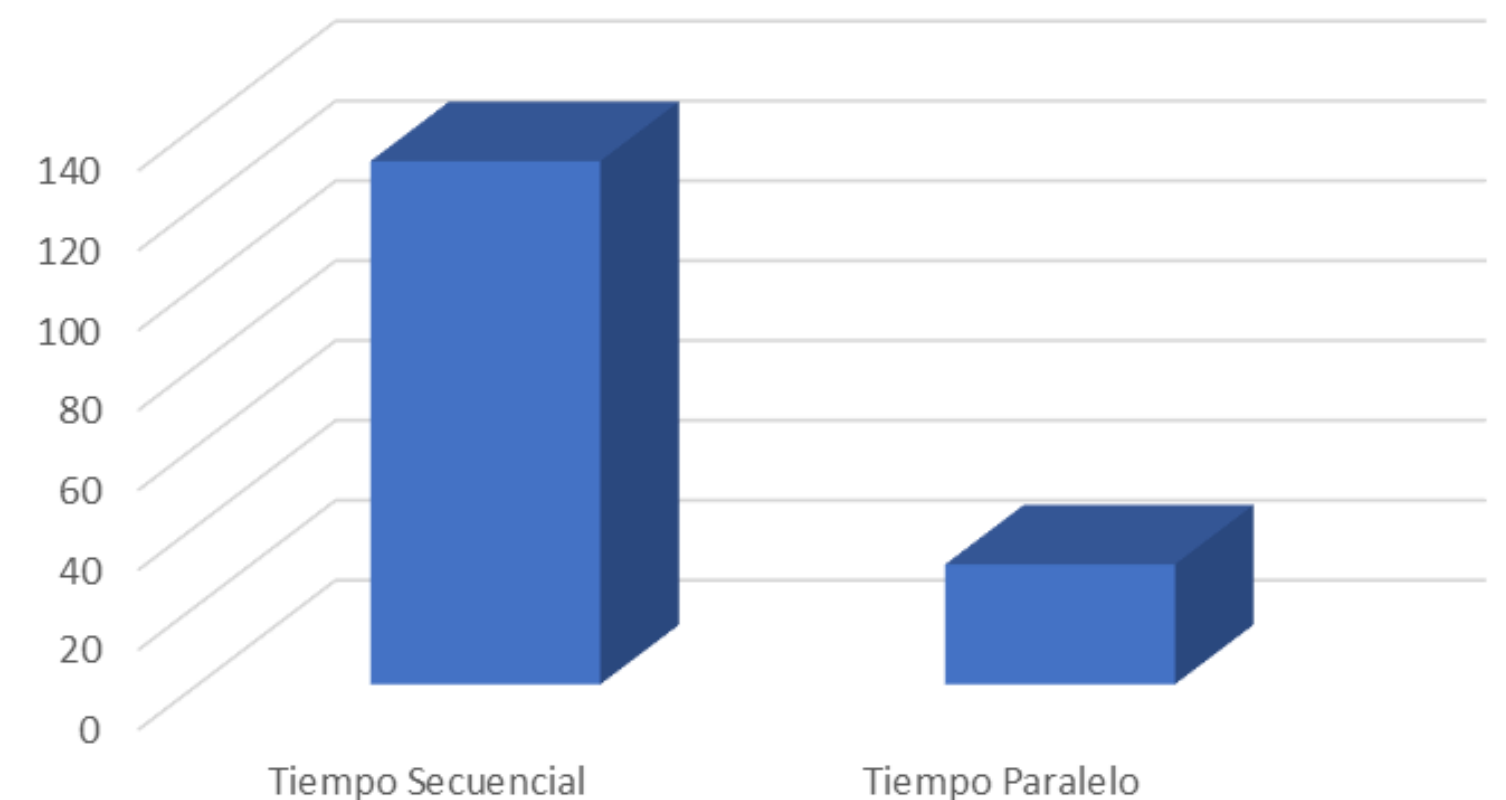
SPEEDUP: 4

EFICIENCIA: 50%

Tiempos de ejecución



Tiempos de Ejecución



```
PS C:\Users\Lenovo\Desktop\output> & .\'P_2.exe'  
Ingrese la cantidad de palabras (frutas) a generar: 200000
```

```
Tiempo de ejecución secuencial: 130 segundos
```

```
Tiempo de ejecución paralelo con profundidad máxima 10: 17 segundos
```

```
SPEEDUP: 7
```

```
EFICIENCIA: 87.5%
```

≥ 200,000

```
PS C:\Users\Lenovo\Desktop\output> & .\'P_2.exe'  
Ingrese la cantidad de palabras (frutas) a generar: 200000
```

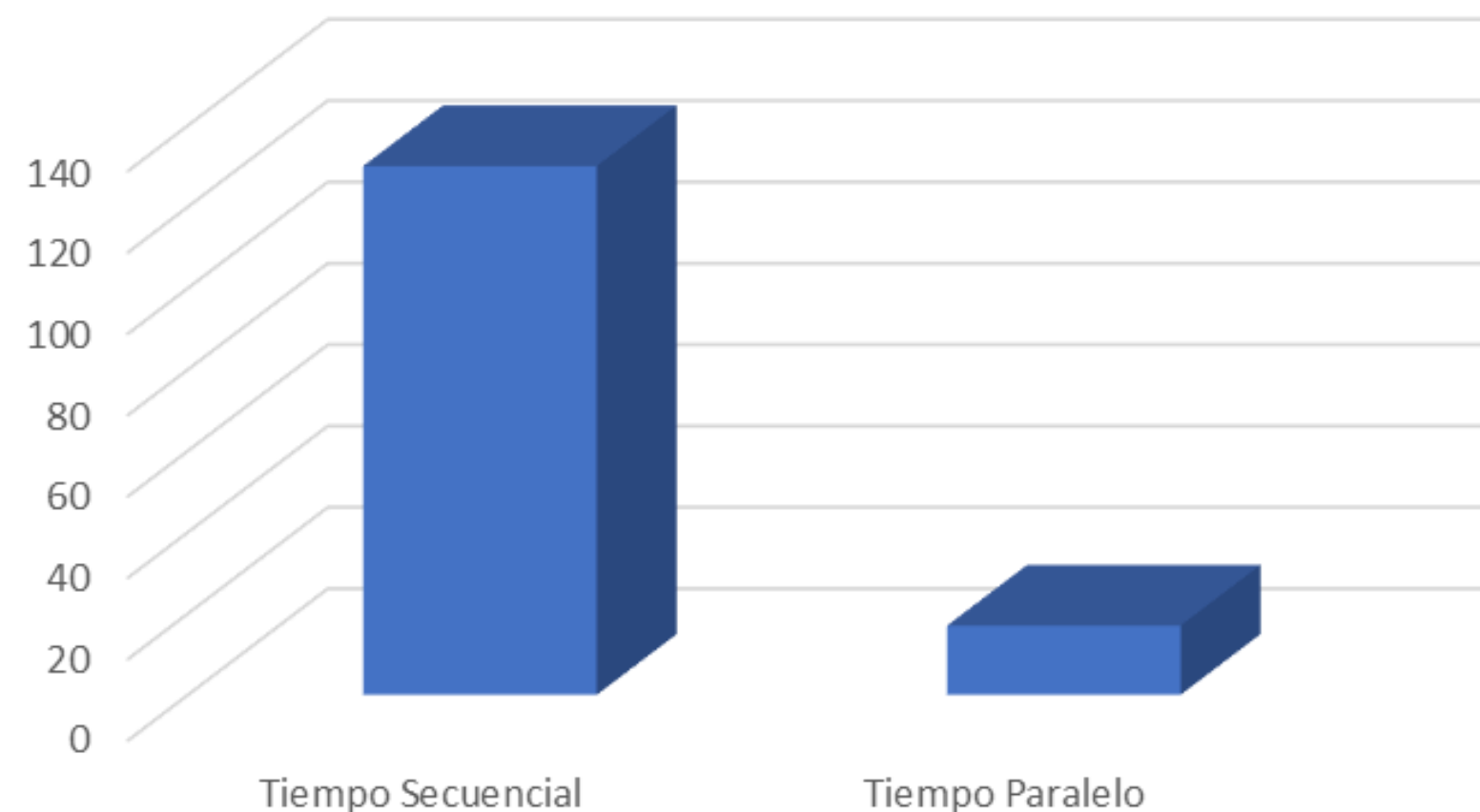
```
Tiempo de ejecución secuencial: 134 segundos
```

```
Tiempo de ejecución paralelo con profundidad máxima 16: 17 segundos
```

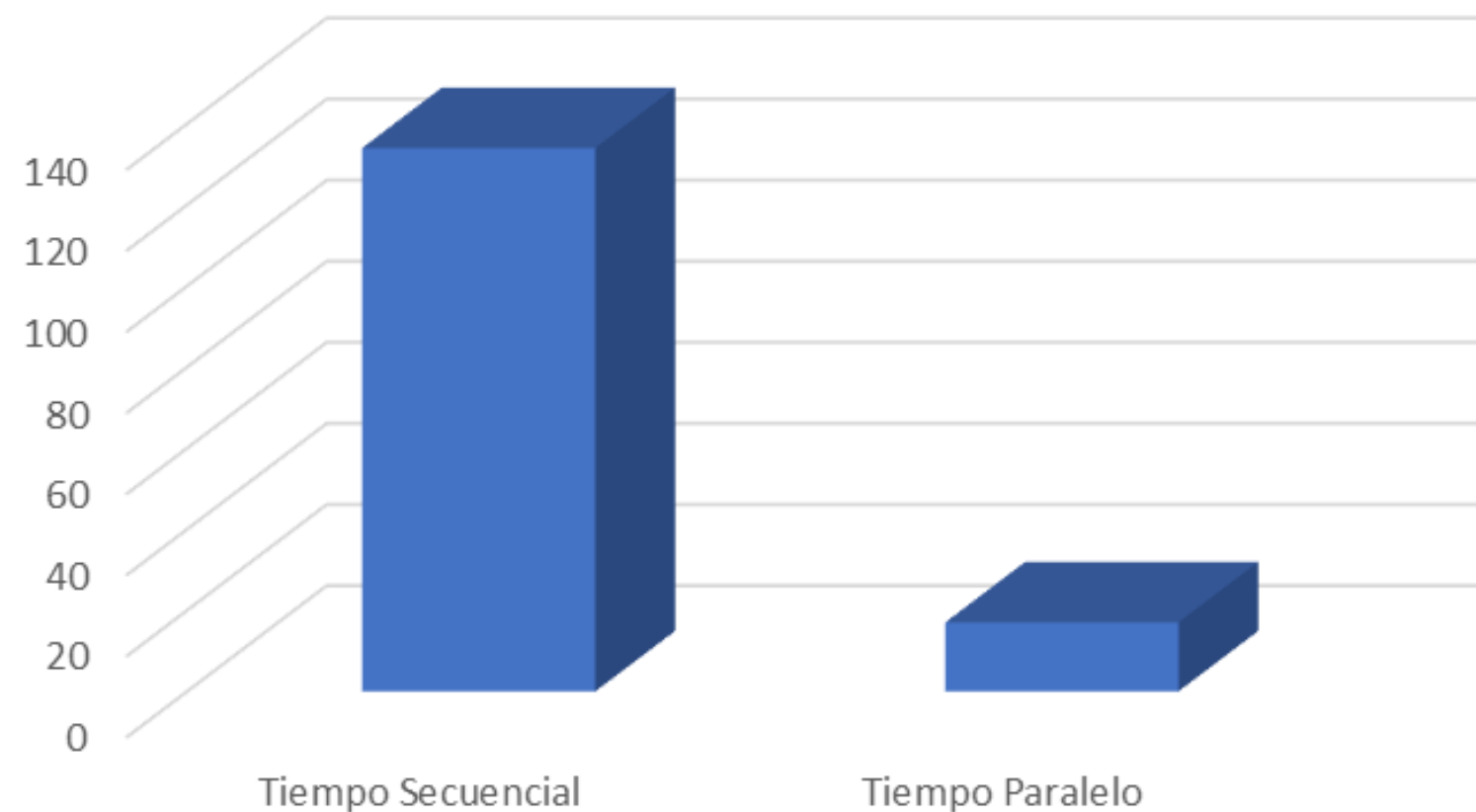
```
SPEEDUP: 7
```

```
EFICIENCIA: 87.5%
```

Tiempos de Ejecución



Tiempos de Ejecución





```
PS C:\Users\Lenovo\Desktop\output> & .\P_2.exe
Ingrese la cantidad de palabras (frutas) a generar: 300000

Tiempo de ejecución secuencial: 287 segundos

Tiempo de ejecución paralelo con profundidad máxima 1: 213 segundos

SPEEDUP: 1
EFICIENCIA: 12.5%
```

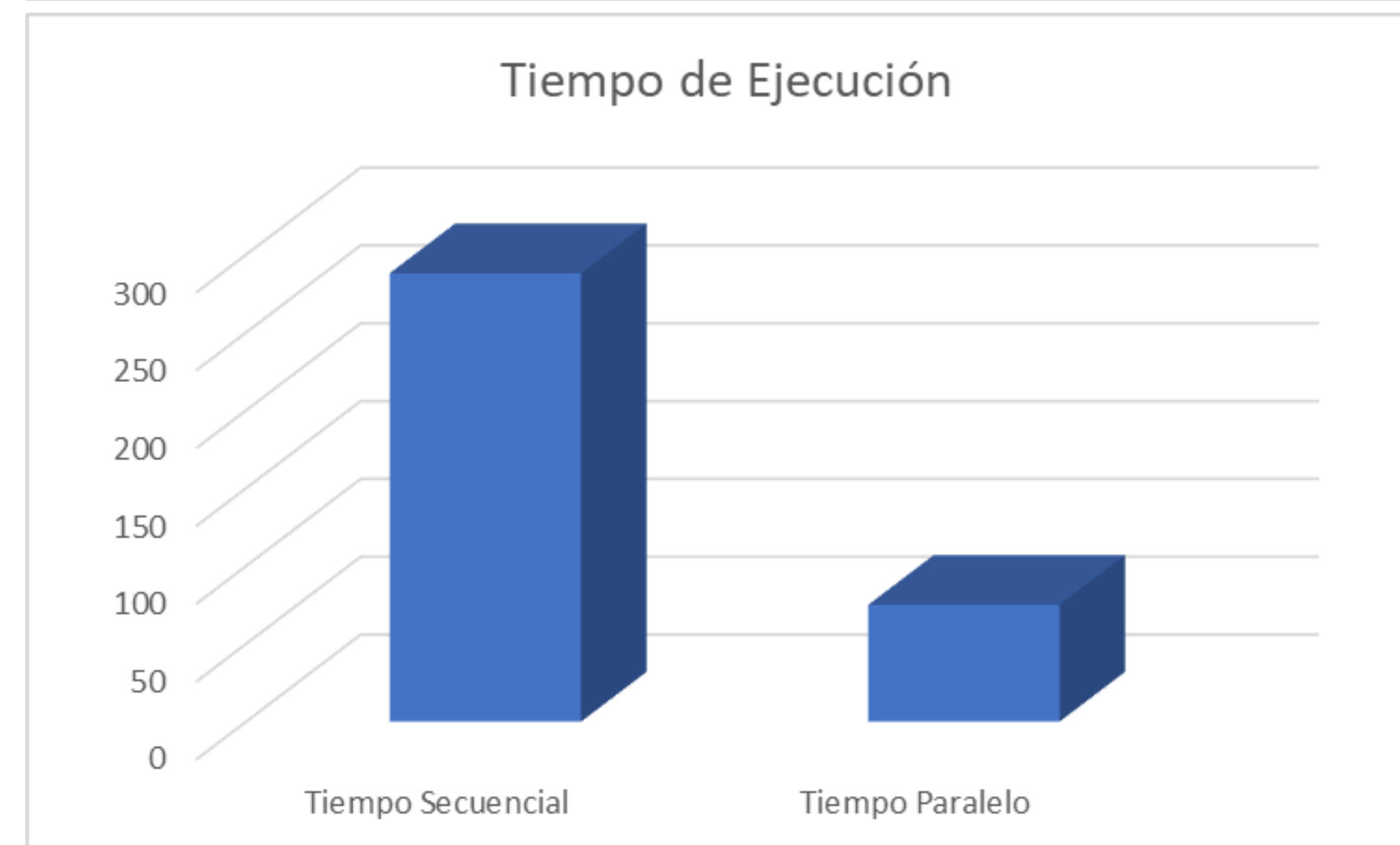
≥ 300,000

```
PS C:\Users\Lenovo\Desktop\output> & .\P_2.exe
Ingrese la cantidad de palabras (frutas) a generar: 300000

Tiempo de ejecución secuencial: 288 segundos

Tiempo de ejecución paralelo con profundidad máxima 5: 75 segundos

SPEEDUP: 3
EFICIENCIA: 37.5%
```



```
PS C:\Users\Lenovo\Desktop\output> & .\P_2.exe'  
Ingrese la cantidad de palabras (frutas) a generar:  
300000
```

```
Tiempo de ejecución secuencial: 292 segundos
```

```
Tiempo de ejecución paralelo con profundidad máxima 10: 39 segundos
```

```
SPEEDUP: 7
```

```
EFICIENCIA: 87.5%
```

➤ 300,000

```
PS C:\Users\Lenovo\Desktop\output> & .\P_2.exe'  
Ingrese la cantidad de palabras (frutas) a generar: 300000
```

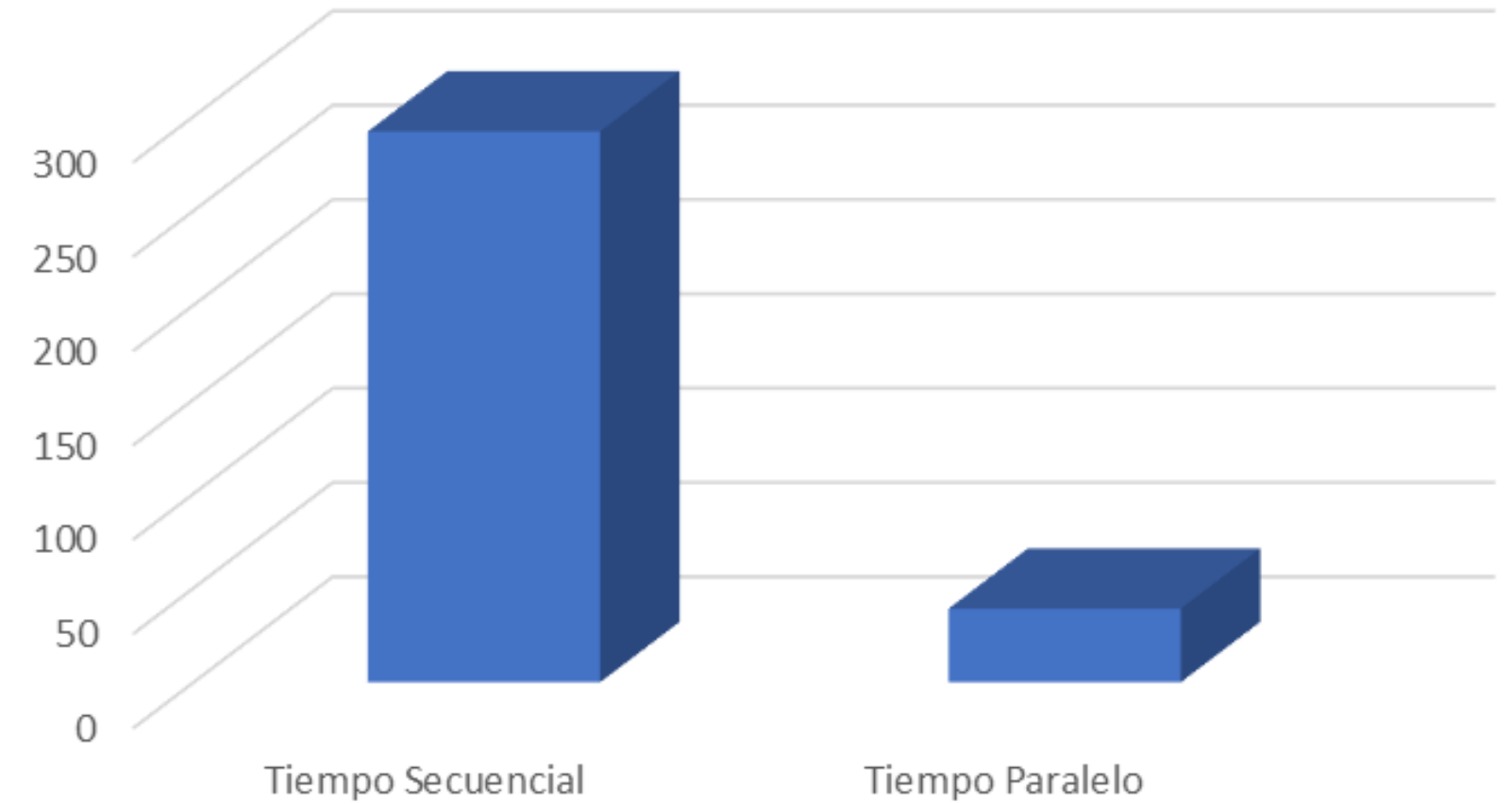
```
Tiempo de ejecución secuencial: 294 segundos
```

```
Tiempo de ejecución paralelo con profundidad máxima 16: 39 segundos
```

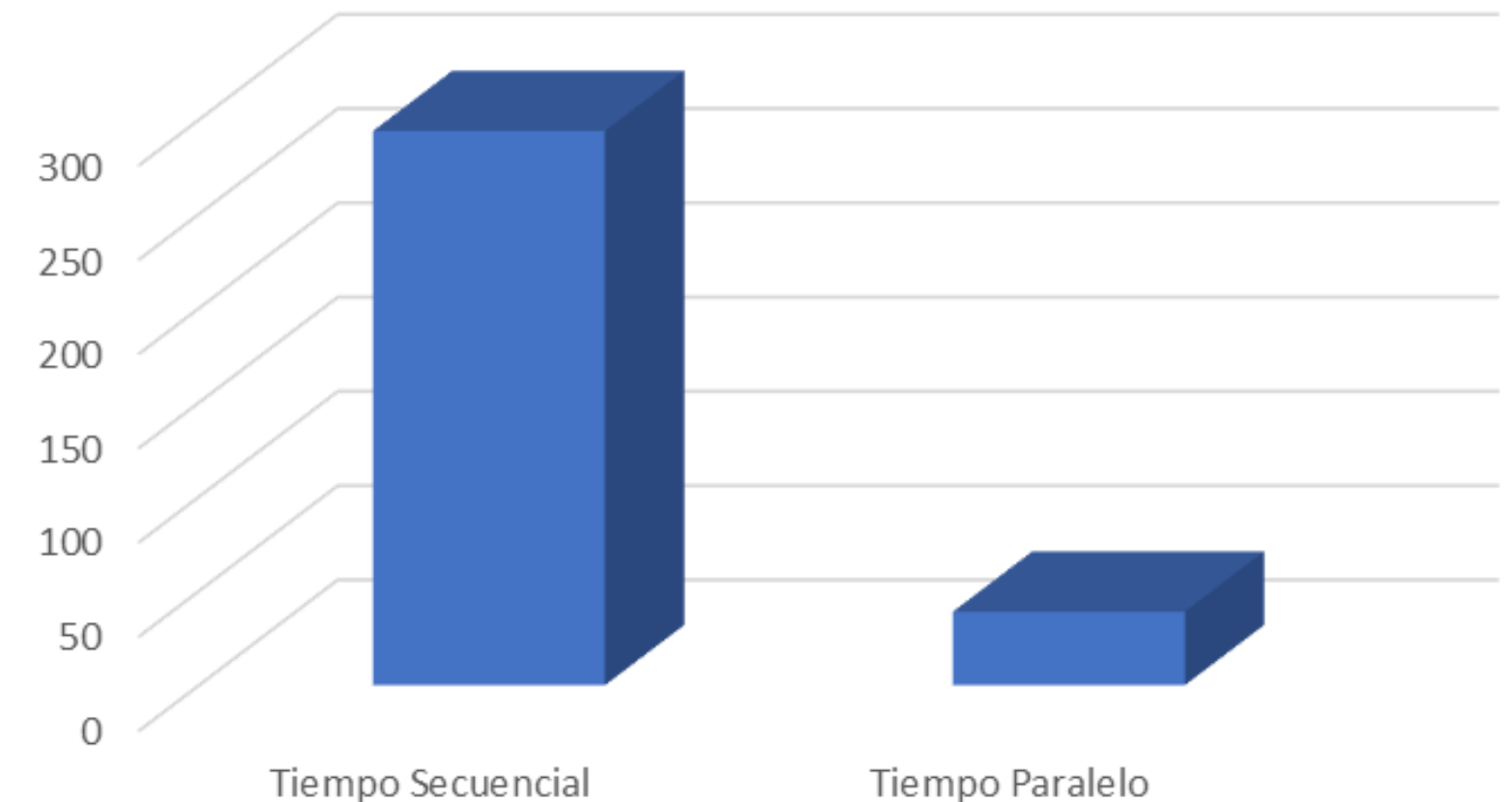
```
SPEEDUP: 7
```

```
EFICIENCIA: 87.5%
```

Tiempo de Ejecución

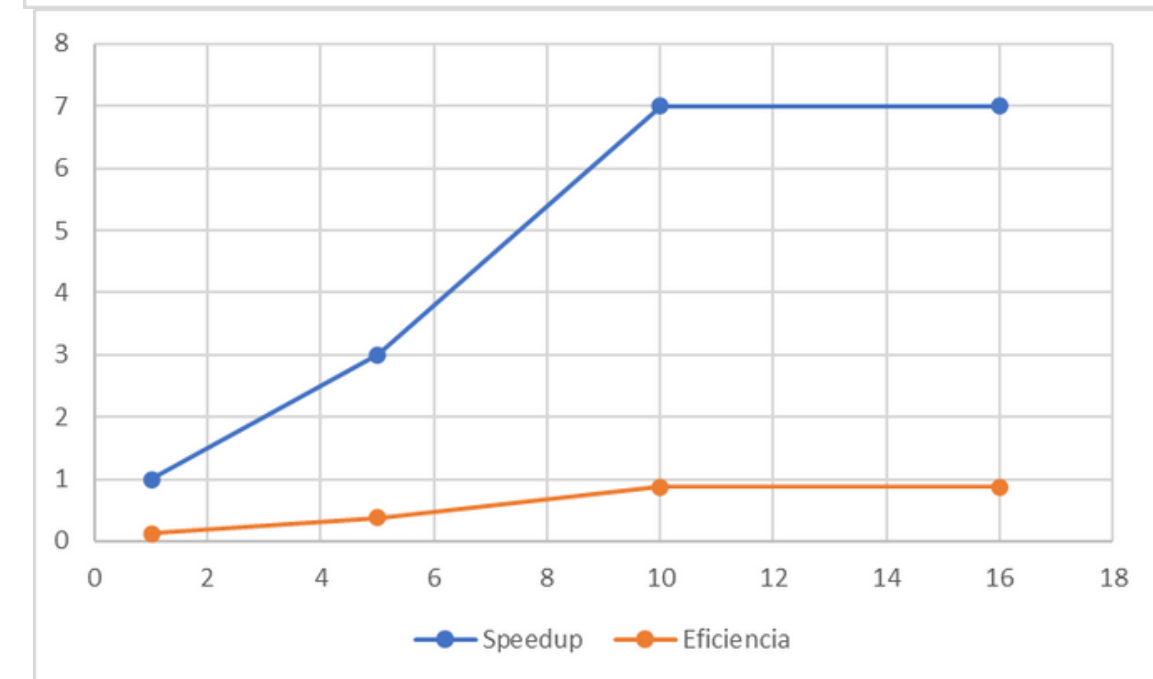
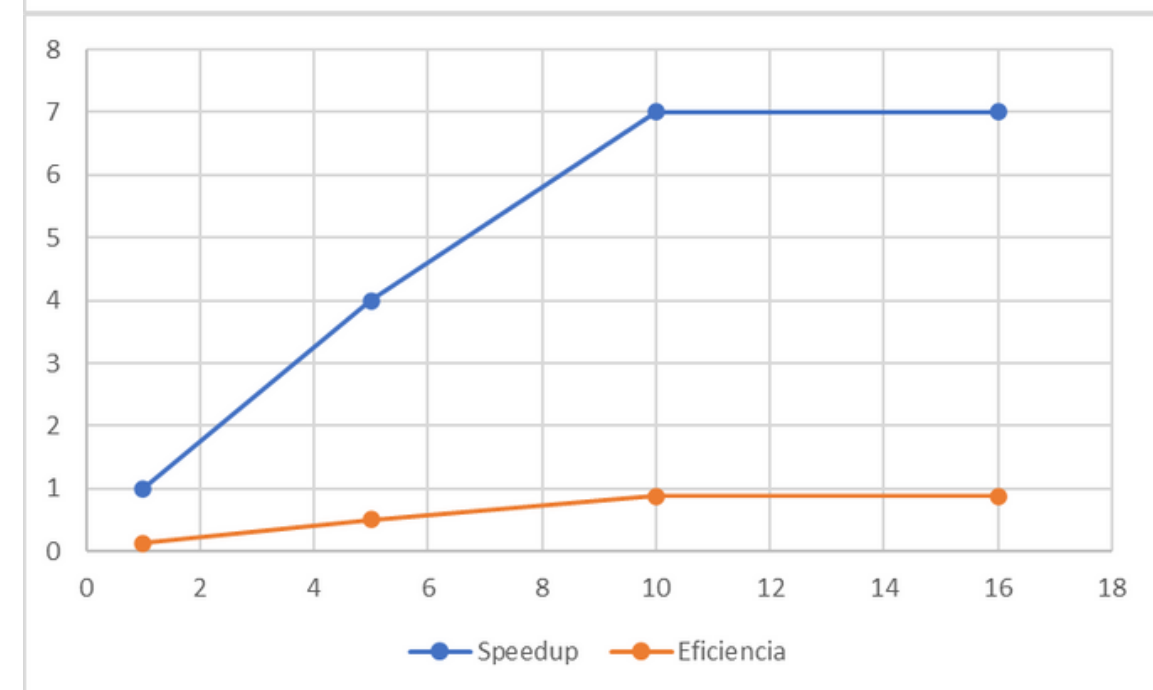
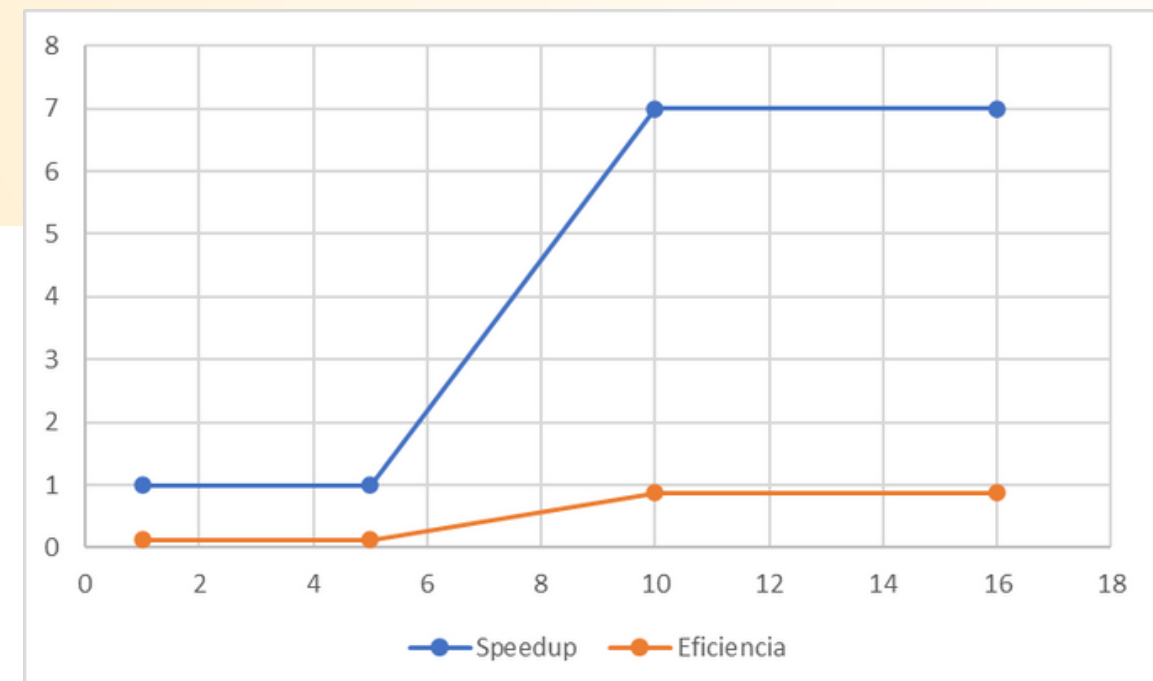


Tiempo de Ejecución






# >SPEED UP, EFICIENCIA

Palabras	Profundidad	Speedup	Eficiencia	Tiempo Secuencial	Tiempo Paralelo
10000	1	1	12.50%	0.3253	0.2249
10000	5	2	25.00%	0.3075	0.1389
10000	10	3	37.50%	0.3219	0.1008
10000	16	3	37.50%	0.312	0.095
100000	1	1	12.50%	33	33
100000	5	1	12.50%	32	22
100000	10	7	87.50%	33	4
100000	16	7	87.50%	34	4
200000	1	1	12.50%	128	121
200000	5	4	50.00%	131	30
200000	10	7	87.50%	130	17
200000	16	7	87.50%	134	17
300000	1	1	12.50%	287	213
300000	5	3	37.50%	288	75
300000	10	7	87.50%	292	39
300000	16	7	87.50%	294	39





# Conclusiones

-  **Tras analizar los resultados, se observa que la implementación paralela del algoritmo Quicksort tiende a ofrecer tiempos de ejecución más rápidos en comparación con la versión secuencial para conjuntos de datos de cierto tamaño y/o profundidad**
-  **A medida que aumenta el tamaño del conjunto de datos, se evidencia la capacidad de la implementación paralela del algoritmo Quicksort para escalar de manera más efectiva y manejar cargas de trabajo más grandes en comparación con la versión secuencial.**
-  **Se discute la aplicabilidad de los resultados en diferentes contextos y se identifican posibles limitaciones del estudio, como la dependencia del hardware y la complejidad del algoritmo.**





# Referencias

Dawson, N. (2023, julio 12). Unraveling QuickSort: The fast and versatile sorting algorithm. Medium. <https://medium.com/@nathaldawson/unraveling-quicksort-the-fast-and-versatile-sorting-algorithm-2c1214755ce9>

del Chelela, C. [@clandelchelela4235]. (2020, marzo 19). Quick Sort algoritmo de ordenamiento explicado al detalle. Youtube. <https://www.youtube.com/watch?v=YzHDivxOQcl>

Wikipedia contributors. (s/f). C. A. R. Hoare. Wikipedia, The Free Encyclopedia. [https://es.wikipedia.org/w/index.php?title=C.\\_A.\\_R.\\_Hoare&oldid=158018962](https://es.wikipedia.org/w/index.php?title=C._A._R._Hoare&oldid=158018962)



Parallel and concurrent programming

**iMuchas gracias!**

---