

# Análisis del algoritmo de ordenamiento de Timsort

Samy Felipe Cuestas Merchan, Camilo Hernández Guerrero, Juan Camilo Mendieta Hernández

24 de febrero de 2022

## Resumen

En este documento se presenta el análisis del ordenamiento de TimSort. Los elementos a presentar son análisis, diseño y algoritmo de Timsort.

## 1. Parte I: Análisis y diseño del algoritmo

### 1.1. Análisis

El problema parte de la necesidad de ordenar una secuencia dada mediante el uso del algoritmo de ordenamiento conocido como Timsort, dicha secuencia está dada por

$$S = \langle s_1, s_2, \dots, s_n \rangle = \langle s_i \in \mathbb{Z} \wedge 1 \leq i \leq n \rangle$$

donde S es la secuencia entrante, n es el tamaño de la secuencia, i es el índice de cada elemento y  $\mathbb{Z}$  es el conjunto de los números enteros.

### 1.2. Diseño

#### 1.2.1. Entradas

Como entradas tenemos la secuencia  $S = \langle s_i \in \mathbb{T} \wedge 1 \leq i \leq n \rangle$  de  $n \in \mathbb{N}$  elementos que pertenecen a un conjunto  $\mathbb{T}$ .

#### 1.2.2. Salidas

Como salidas tenemos la secuencia  $S = \langle s_i \in \mathbb{T} \wedge 1 \leq i \leq n \rangle$  pero con sus elementos cumpliendo la relación de orden parcial  $\leq$  para cada elemento adyacente.

## 2. Parte II: Algoritmos

### 2.1. Timsort

---

**Algorithm 1** Ordenamiento TimSort

---

```
1: procedure TIMSORT( $S$ )
2:    $minRun \leftarrow calcMinRun(|S|)$ 
3:   for  $i \leftarrow 0$  to  $|S|$  of tam  $minRun$  do
4:      $end \leftarrow \text{MIN}(i + MINRUN - 1, N - 1)$ 
5:     INSERTIONSORT( $S, i, END$ )
6:   end for
7:    $size \leftarrow minRun$ 
8:   while  $size < n$  do
9:     for  $left \leftarrow 0$  to  $|S|$  of tam  $2 * size$  do
10:       $mid \leftarrow \text{MIN}(N - 1, LEFT + SIZE - 1)$ 
11:       $right \leftarrow \text{MIN}(LEFT + 2 * SIZE - 1, N - 1)$ 
12:      if  $mid < right$  then
13:        MERGE( $S, LEFT, MID, RIGHT$ )
14:      end if
15:    end for
16:     $size \leftarrow 2 * size$ 
17:  end while
18: end procedure
```

---

### 2.2. Calculo del run minimo

---

**Algorithm 2** calcMinRun

---

```
1: procedure CALCMINRUN( $|S|$ )
2:    $r \leftarrow 0$ 
3:   while  $n \geq 32$  do
4:      $r \leftarrow n \& 1$ 
5:      $r \gg \leftarrow 1$ 
6:   end while
7:   return  $n + r$ 
8: end procedure
```

---

## 2.3. Mezcla de subsecuencias

---

**Algorithm 3** Función merge

---

```
1: procedure MERGE( $S, l, m, r$ )
2:    $len1 \leftarrow m - l + 1$ 
3:    $len2 \leftarrow r - m$ 
4:   for  $i \leftarrow 0$  to  $len1$  do
5:      $left$  append  $S[l + i]$ 
6:   end for
7:   for  $i \leftarrow 0$  to  $len2$  do
8:      $right$  append  $S[m + 1 + i]$ 
9:   end for
10:   $i \leftarrow 0$ 
11:   $j \leftarrow 0$ 
12:   $k \leftarrow l$ 
13:  while  $i < len1$  and  $j < len2$  do
14:    if  $left[i] \leq right[j]$  then
15:       $S[k] \leftarrow left[i]$ 
16:       $i \leftarrow i + 1$ 
17:    else
18:       $S[k] \leftarrow right[j]$ 
19:       $j \leftarrow j + 1$ 
20:    end if
21:     $k \leftarrow k + 1$ 
22:  end while
23:  while  $i < len1$  do
24:     $S[k] \leftarrow left[i]$ 
25:     $k \leftarrow k + 1$ 
26:     $i \leftarrow i + 1$ 
27:  end while
28:  while  $j < len2$  do
29:     $S[k] \leftarrow right[j]$ 
30:     $k \leftarrow k + 1$ 
31:     $j \leftarrow j + 1$ 
32:  end while
33: end procedure
```

---

## 2.4. Ordenamiento por inserción

---

**Algorithm 4** Ordenamiento por inserción

---

```
1: procedure INSERTIONSORT( $S, left, right$ )
2:   for  $i \leftarrow left + 1$  to  $right + 1$  do
3:      $j \leftarrow i$ 
4:     while  $j > left$  and  $S[j] < S[j - 1]$  do
5:        $S[j] \leftarrow S[j - 1]$ 
6:        $S[j - 1] \leftarrow S[j]$ 
7:        $j \leftarrow j - 1$ 
8:     end while
9:   end for
10: end procedure
```

---

## 2.5. Complejidad

Por investigación se encontro que el algoritmo TimSort tiene orden de complejidad  $O(|S| \log(|S|))$  en su caso promedio y en su peor caso, mientras que en su mejor caso tiene orden de complejidad

$\Omega(n)$ . Esto gracias a que aprovecha las secciones de las secuencias que ya están ordenadas para copiarlas directamente en una subsecuencia sin tener que ordenarlas de nuevo.

## 2.6. Invariante

- **TimSort** Por cada iteración cada subsecuencia cuyo tamaño se determina por el tamaño de la mínima subsecuencia se encuentra organizada, luego se mezcla dando como resultado una secuencia que cumple con la relación de orden parcial  $\leq$ .

**Inicio:** La secuencia  $S$  está dividida en  $i$  subsecuencias las cuales pueden estar ordenadas y cuyo tamaño es determinado por `minrun`

**Avance:** En la  $i$ -ésima permutación la  $i$ -ésima subsecuencia está ordenada mientras que en cada iteración del `while` se mezclan estas subsecuencias

**Terminación:**  $S'$  es una secuencia ordenada

- **Merge** Las banderas de control `len1` y `len2` indican si las variables  $i$  y  $j$  siguen la relación de orden parcial  $<$ .

**Inicio:** Se inicia con el resultado de `CALCMINRUN`, dicha función determina la mínima longitud de un run entre 32 y 64 para que el tamaño del arreglo sea menor o igual a una potencia de 2.

**Avance:** Se valida que la variable `size` sea menor a  $n$ , de ser así, se elige un punto en el subarreglo para después fusionar el arreglo a su izquierda, el siguiente a este y después se multiplica en 2.

**Terminación:** La variable `size` es mayor que el tamaño del arreglo.

- **InsertionSort** Por cada iteración los primeros  $i$  elementos están ordenados

**Inicio:** El elemento  $i$  está ordenado.

**Avance:** Por cada iteración los elementos desde el inicio de arreglo hasta  $i$  están ordenados.

**Terminación:**  $S'$  es una secuencia ordenada.

## 3. Reporte de pruebas

### 3.1. Tiempo de ejecución por cantidad de elementos en la secuencia

En el siguiente gráfico se puede observar la tendencia en el tiempo (en milisegundos) que existe al aumentar la cantidad de elementos existentes en la secuencia  $S$ .

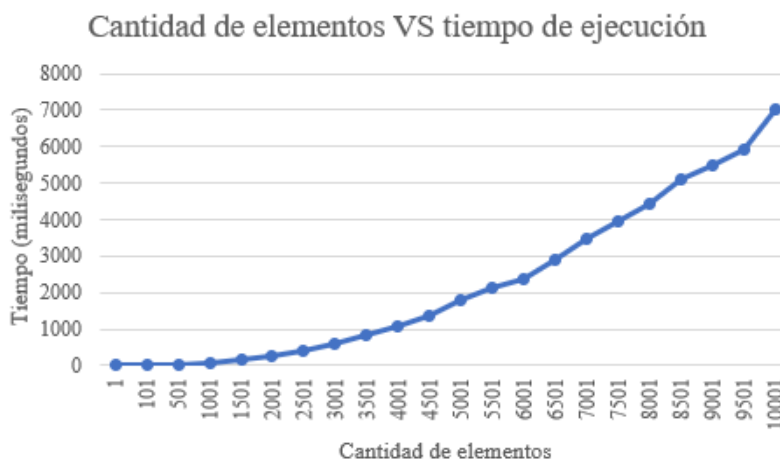


Figura 1: Cantidad de elementos VS Tiempo de ejecución

### 3.2. Tabla de los resultados obtenidos

La siguiente tabla ilustra de manera ordenada y más específica la relación dada en la gráfica de la subsección anterior.

Tiempo (milisegundos) (y)	Elementos (x)
0.004	1
0.674	101
17.677	501
65.881	1001
153.125	1501
279.578	2001
417.878	2501
608.01	3001
834.625	3501
1054.222	4001
1379.562	4501
1801.025	5001
2114.694	5501
2365.756	6001
2885.292	6501
3484.229	7001
3939.506	7501
4455.509	8001
5082.716	8501
5495.812	9001
5906.145	9501
7013.926	10001

## Referencias

- [1] Kumar,A.(26 de junio de 2021) GeeksforGeeks.Recuperado de <https://www.geeksforgeeks.org/timsort/>.
- [2] Thomas, H., Charles, E., Ronald, L., Clifford, S.(2009).Introduction to Algorithms.Massachusetts,USA: The MIT Press
- [3] Hoque,T.(4 de junio de 2019) Youtube.Recuperado de [https://www.youtube.com/watch?v=\\_dlzWEJoU7I](https://www.youtube.com/watch?v=_dlzWEJoU7I).