UVa

PRACTICA 1
ADA 2023-2024

INFORME COMPARACIÓN DE ALGORITMOS

Mario Danov Ivanov - Mario Cobreros del Caz
mario.danov@estudiantes.uva.es mario.cobreros@estudiantes.uva.es

Contenido

1 – Proporción del reparto del trabajo:

Mario Cobreros del Caz	50%
Mario Danov Ivanov	50%

2 – Descripción breve de la realización de la práctica:

Hemos creado un programa Java con distintas funciones de apoyo para realizar correctamente la ejecución en bucle de los distintos algoritmos.
Entre ellas están:

- rellenarVector(int n): la cual utilizamos para devolver un vector ya inicializado con n valores que tienen un rango 0-2n para así evitar la aparición de valores repetidos.

- comp/asig(int n): dos funciones que utilizamos en los propios algoritmos de ordenación para añadir, de forma más clara y menos engorrosa, comparaciones y asignaciones a las variables que almacenan esta información

El funcionamiento del programa consiste en tres bucles anidados:
El primero, el cual solo distingue cuál de los tres algoritmos va a ser ejecutado; el segundo, el cual se repetirá 10 veces añadiendo 10.000 elementos al vector en cada repetición y finalmente el tercero, en el que se crea un vector del tamaño actual del segundo bucle y se ejecuta el algoritmo distinguido por el primero. Este proceso se realiza 20 veces, cada repetición con un vector nuevo.

Al finalizar se realizan las medias de las comparaciones, las asignaciones y el tiempo empleado.

Como añadido imprimimos por pantalla el número del algoritmo que se está ejecutando y en que iteración se encuentra como forma de facilitar la espera y evitar pensar que se ha colgado el sistema debido al gran tiempo de ejecución de alguno de los algoritmos.

Una vez realizados estos tres bucles, el programa finaliza, habiendo almacenado todos los datos en sus respectivas filas y columnas en un archivo .csv, el cual es abierto automáticamente en Excel para la rápida comprobación de los datos.

3 – Análisis de la eficiencia:

Para realizar el análisis de la eficiencia de los algoritmos hemos utilizado Microsoft Excel, importando los valores del anterior fichero .csv. Antes de comparar los diferentes algoritmos, presentamos las medias obtenidas para el número de operaciones, numero de asignaciones y tiempo empleado para cada algoritmo y tamaño del vector.

- Algoritmo 1:

Tabla 1: Datos del algoritmo 1

Tamaño	NumComp	NumAsign	TiempoEmpleado
10000	172497	330173	1200000
20000	417670	743346	1650000
30000	682156	1218167	2750000
40000	987601	1712612	4500000
50000	1314397	2228408	5800000
60000	1661050	2764060	6400000
70000	2010208	3302219	7100000
80000	2434131	3915141	8350000
90000	2738740	4535750	10100000
100000	3100717	5107727	11000000

- Algoritmo 2:

Tabla 2: Datos del algoritmo 2

Tamaño	NumComp	NumAsign	TiempoEmpleado
10000	39443223	78748328	78150000
20000	158132881	315358899	367700000
30000	354742380	708249514	829000000
40000	629806868	1258012899	1481650000
50000	985213507	1968005009	2324200000
60000	1418606757	2832826343	3349050000
70000	1931602346	3862090775	4564850000
80000	2521462430	5040283928	5988400000
90000	3194401800	6385629590	7534300000
100000	3940468840	7878875497	9286550000

- Algoritmo 3:

Tabla 3: Datos del algoritmo 3

Tamaño	NumComp	NumAsign	TiempoEmpleado
10000	126459	280593	1600000
20000	273926	603187	2100000
30000	429087	939187	2800000
40000	589877	1290374	4100000
50000	754143	1647374	5250000
60000	921081	2004374	6500000
70000	1090902	2370748	8150000
80000	1263712	2748748	8850000
90000	1437955	3126748	10250000
100000	1613199	3504748	11250000

Desde una primera vista a las tablas, ya se puede apreciar que el algoritmo 2 tiene unos valores bastante elevados para las tres categorías que estudiamos comparado a los otros algoritmos. Es por eso por lo que, al representar la evolución de las asignaciones, comparaciones y el tiempo empleado en los tres algoritmos mediante una gráfica, hemos decidido producir dos versiones para cada categoría (una normal y otra reducida, como con una especie de zoom).

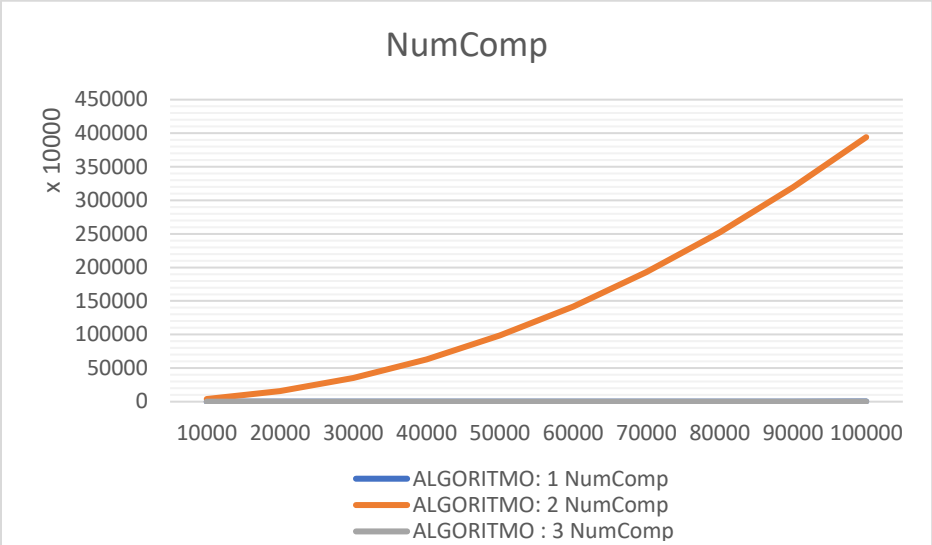


Ilustración 1: Grafica del número de comparaciones de los algoritmos

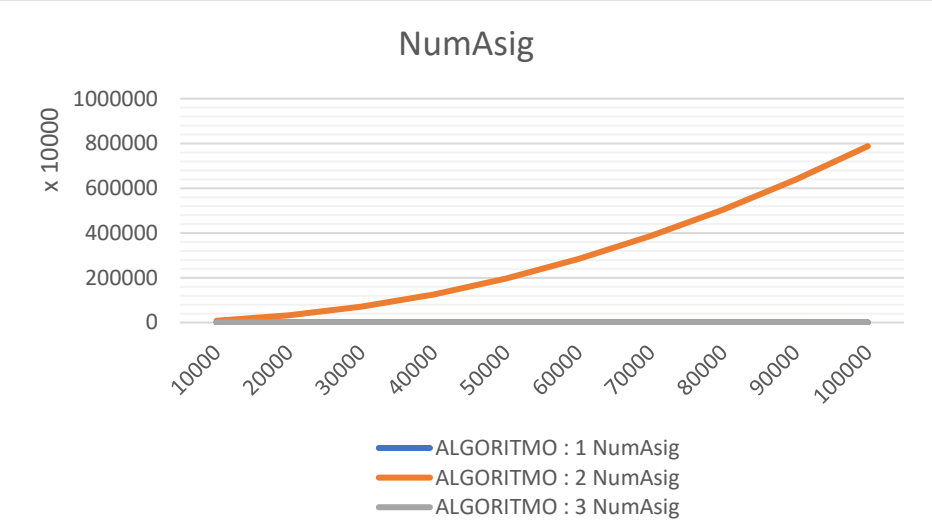


Ilustración 2: Grafica del número de asignaciones de los algoritmos

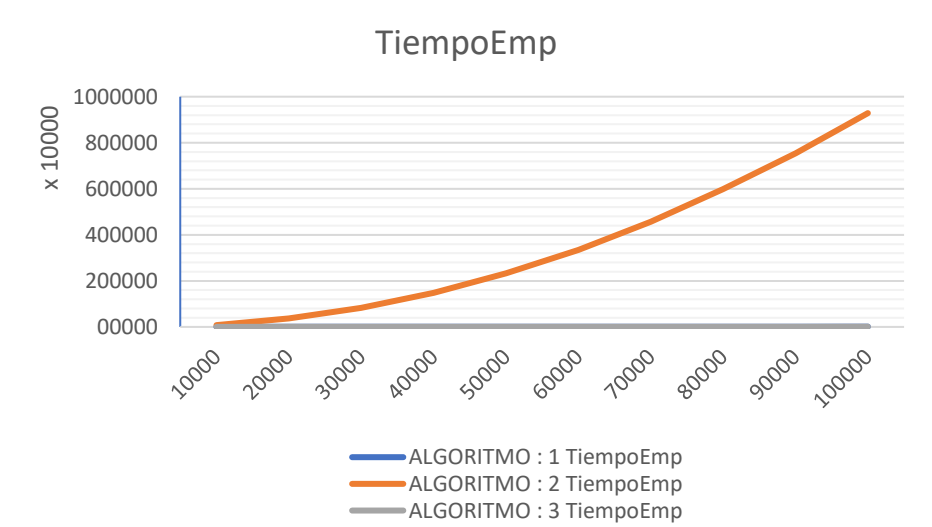


Ilustración 3: Grafica del tiempo empleado de los algoritmos

En estas versiones de las gráficas se puede ver como en el eje vertical, el número medio al que llegan las comparaciones, asignaciones y el tiempo empleado, tiene un rango que abarca todos los valores de los tres algoritmos. Como el segundo algoritmo llega a tener unos valores mucho mayores que los otros dos, estos quedan minimizados, apareciendo como una línea recta y constante en la parte baja de las gráficas.

Aun así, fijándonos solo en el algoritmo 2, podemos apreciar una función ligeramente exponencial en todas las categorías. Basándonos en esto y en sus valores desmesurados, podemos deducir que se trata de un algoritmo de orden O(n^2).

Ahora bien, nos fijamos en la otra versión de las gráficas:

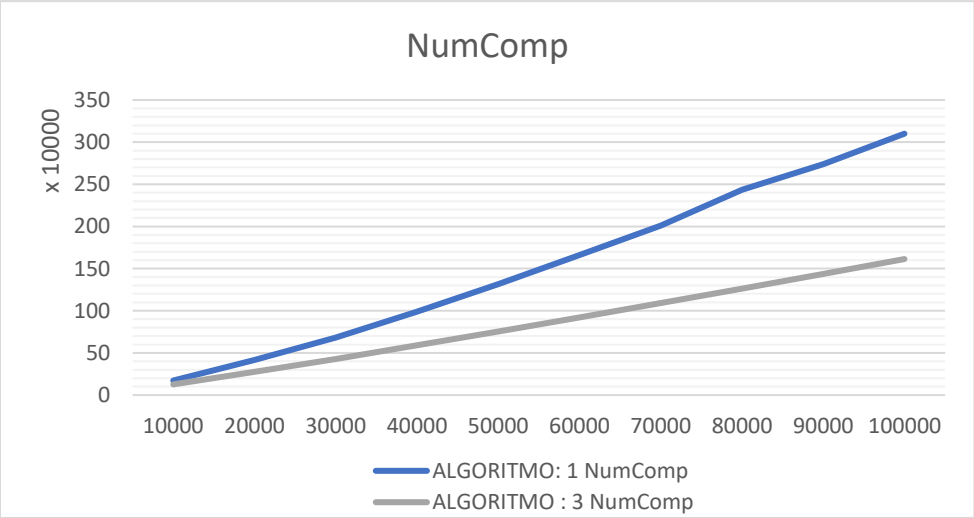


Ilustración 4: Grafica del número de comparaciones de los algoritmos

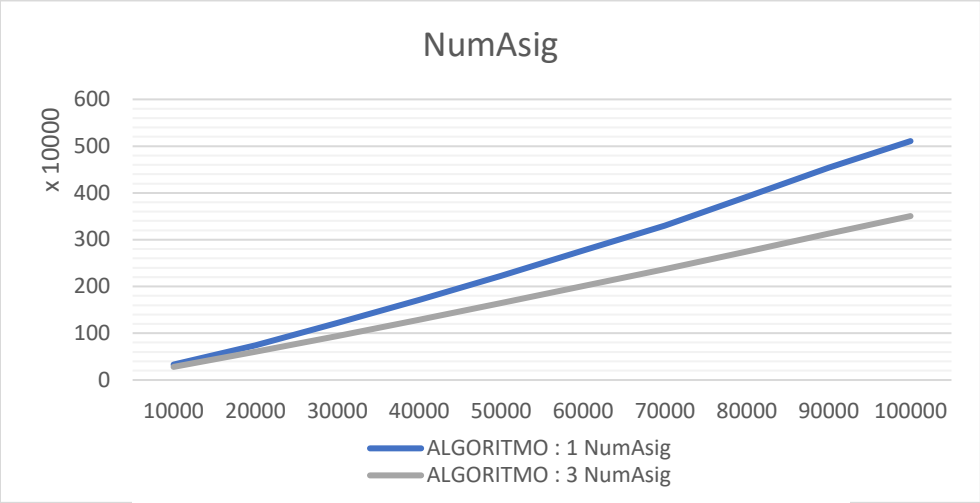


Ilustración 5: Grafica del número de asignaciones de los algoritmos

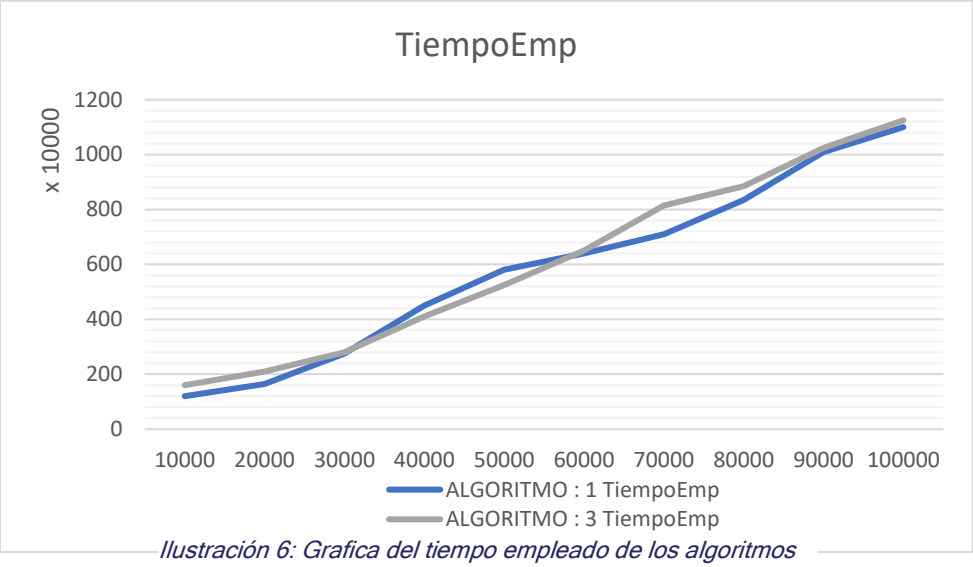


Ilustración 6: Grafica del tiempo empleado de los algoritmos

En estas versiones de las gráficas el rango del eje vertical se ha acortado, abarcando solo los valores relevantes a el primer y tercer algoritmo. Como se puede apreciar los valores del algoritmo dos son tan altos que ni aparecen en estas graficas.

Analizando ahora los dos algoritmos juntos podemos ver que tienen costes muy similares, siendo el algoritmo 1 ligeramente mayor en comparaciones y asignaciones. Aun así, deducir de que orden son estos algoritmos no será tan fácil como el anterior, ya que no podemos llegar a una conclusión solo a partir de sus gráficas y valores.

Para encontrar el orden de estos algoritmos tenemos que analizar su funcionamiento e implementación.
Empezando con el primero podemos ver que lo primero que hace es calcular un variable que va a utilizar como rango para comparar elementos del vector.

```
while (h <= r/9) {
  h = 3*h+1;
}
```

Después divide el vector en dos partes y va comparando elementos de la primera parte con la segunda, intercambiando elementos entre las dos partes cuando es necesario. Cuando ordena todos los elementos que puede para ese rango, disminuye el rango y vuelve a empezar.

```
while ( h > 0 ) {
  for (i = h; i <= r; i++) {
    j = i;
    w = v[i];
    while ((j >= h) && (w<v[j-h])) {
      v[j] = v[j-h];
      j = j - h;
    }
    v[j] = w;
  }
  h = h / 3;
}
```

Este comportamiento es indicativo del algoritmo Shellsort, un algoritmo popular cuyo mejor caso es de orden O(n*log(n)) y peor caso es de orden $O(n^2)$, por lo que su caso promedio oscila entre estos dos órdenes. El orden del caso promedio entonces tendrá que ser $O(n^k)$ siendo k un numero entre 1 y 2.

Para determinar k obtenemos las constantes y las aproximaciones sustituyendo k por varios números dentro del intervalo entre 1 y 2, y viendo cual se ajusta mas a los datos reales.

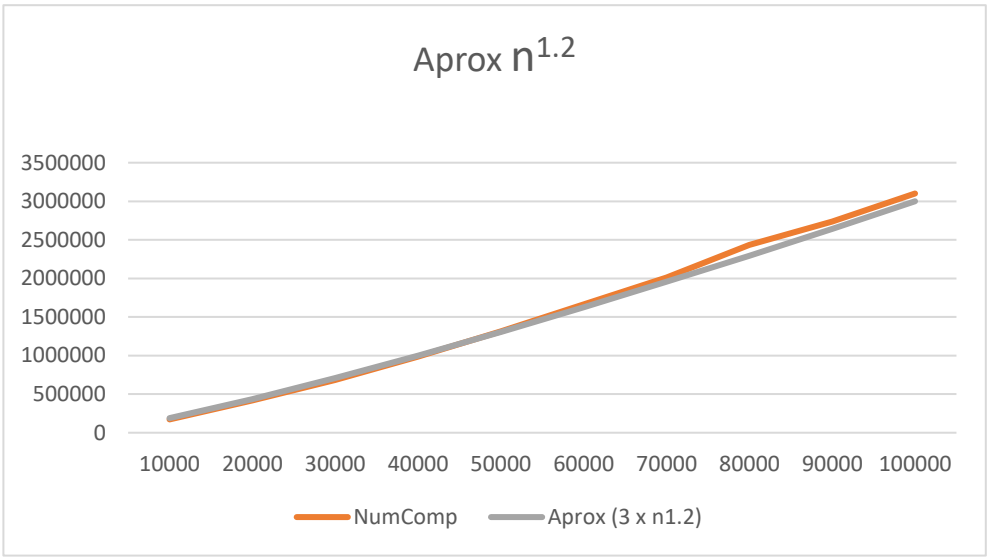
Constante Comp con $n^{1.2}$	Constante Comp con $n^{1.3}$	Constante Comp con $n^{1.4}$
2,73389321	1,088382491	0,433292874
2,881358439	1,070272306	0,39754957
2,892931815	1,03187258	0,368056038
2,96557923	1,027787975	0,356202967
3,019691343	1,023447462	0,346871448
3,066205124	1,020436724	0,33960256
3,084063766	1,010679705	0,331210229
3,181529562	1,028790545	0,332673315
3,107848168	0,993197328	0,317403193
3,100717	0,98053281	0,3100717
3,003381766	**1,027539993**	**0,353293389**

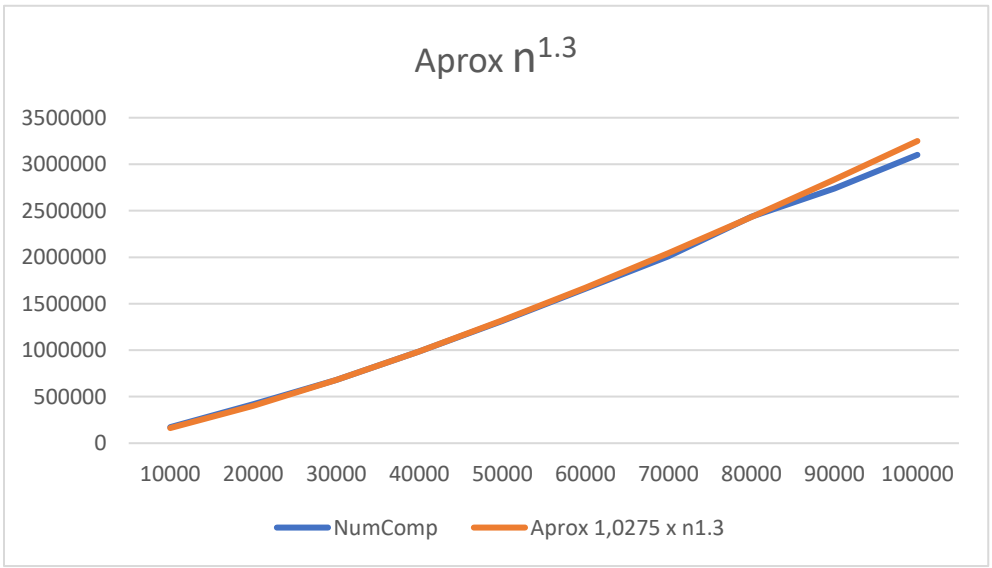
Aclaramos que las constantes finales obtenidas son los valores medios escritos en negrita al final de cada columna.
Además, aunque solo estemos utilizando las comparaciones como ejemplo, estamos determinando el orden de todo el algoritmo, por lo que también nos servirá para las asignaciones y el tiempo empleado.

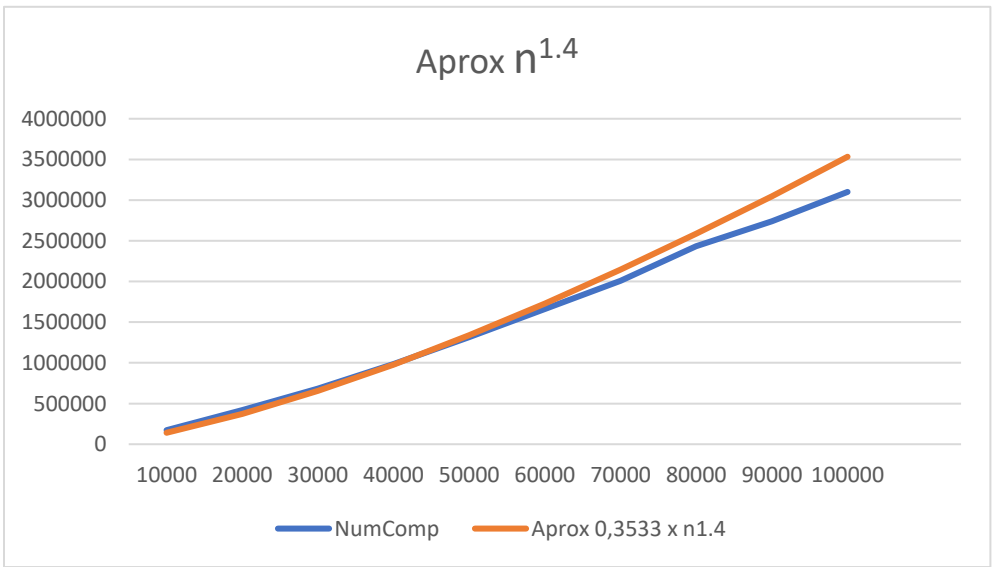
Después de obtener las constantes calculamos las aproximaciones del número de comparacione para poder compararlo con los datos reales y determinar el orden del algoritmo y poder ver cual se aproxima más a los valores reales.

Aprox (3 x n1.2)	Aprox 1,0275 x n1.3	Aprox 0,3533 x n1.4
189287,2033	162847,7755	140651,2634
434867,7982	400978,2581	371180,9093
707402,777	679265,3505	654807,1215
999063,8489	987324,2847	979552,2923
1305825,845	1319601,608	1338756,657
1625184,813	1672547,483	1728046,354
1955414,822	2043663,002	2144277,031
2295245,999	2431077,555	2585053,998
2643700,578	2833329,561	3048478,6
3000000	3249240,296	3533000

Comparamos ahora con los datos reales:







Podemos ver ahora que la aproximación más similar a la real es la que se consigue con $n^{1.3}$ por lo que concluimos que el orden del primer algoritmo es específicamente $O(n^{1.3})$.

Si nos fijamos ahora en el algoritmo 3, vemos que es un algoritmo recursivo. Su funcionamiento se basa en dividir recursivamente el vector en partes iguales hasta llegar al caso base de un solo elemento.

```
ordena3rec(v, w, l, m);
ordena3rec(v, w, m + 1, r);
```

Seguidamente fusiona de manera ordenada todos los elementos en otro array.

```
while ((ia <= m) && (ib <= r)) {
  if (v[ia] < v[ib]) {
    w[ic] = v[ia];
    ia++;
    ic++;
  } else {
    w[ic] = v[ib];
    ib++;
    ic++;
  }
}
while (ia <= m) {
  w[ic] = v[ia];
  ia++;
  ic++;
}
while (ib <= r) {
  w[ic] = v[ib];
  ib++;
  ic++;
}
for (int i = l; i <= r; i++) {
  v[i] = w[i];
}
```

Esta implementación es propia del algoritmo MergeSort, otro algoritmo popular cuyo mejor, peor y caso promedio son todos O(n*log(n)), lo que explica su gran similitud al algoritmo 1.

Ahora que ya tenemos los órdenes de todos los algoritmos, podemos usarlos para calcular los números constantes “x” por los cuales aumentan el número de

asignaciones, comparaciones y nanosegundos que tardan los algoritmos en completarse dependiendo del tamaño “n” del vector en una determinada ejecución.

Sean x los números constantes que buscamos y f(n) la función dentro del orden de un algoritmo, seguiremos esta fórmula (Ponemos el número de comparaciones como un ejemplo).

$$X * f(n) = \text{Numero de comparaciones}$$

Si despejamos x tenemos la siguiente formula:

$$X = \text{Numero de Comparaciones} / f(n)$$

Ahora aplicamos esta fórmula también para las asignaciones y el tiempo empleado de cada algoritmo, calculamos la media entre los diferentes valores de cada resultado y tendremos las constantes para los tres algoritmos, lo que nos permite una mejor comparación.

- Algoritmo 1:

Tamaño	NumComp	NumAsign	TiempoEmpleado	Constante Comp	Constante Asig	Constante Tiempo
10000	172497	330173	1200000	1,088382491	2,083250793	7,571488134
20000	417670	743346	1650000	1,070272306	1,904811544	4,228097074
30000	682156	1218167	2750000	1,03187258	1,842676933	4,15982502
40000	987601	1712612	4500000	1,027787975	1,782300767	4,683111792
50000	1314397	2228408	5800000	1,023447462	1,73513673	4,516135751
60000	1661050	2764060	6400000	1,020436724	1,698051433	3,931726942
70000	2010208	3302219	7100000	1,010679705	1,66026885	3,569693239
80000	2434131	3915141	8350000	1,028790545	1,654742511	3,529144919
90000	2738740	4535750	10100000	0,993197328	1,644878587	3,66274017
100000	3100717	5107727	11000000	0,98053281	1,615205099	3,478505426
				1,027539993	**1,762132325**	**4,333046847**

- Algoritmo 2:

Tamaño	NumComp	NumAsign	TiempoEmpleado	Constante Comp	Constante Asig	Constante Tiempo
10000	39443223	78748328	78150000	0,39443223	0,78748328	0,7815
20000	158132881	315358899	367700000	0,395332203	0,788397248	0,91925
30000	354742380	708249514	829000000	0,3941582	0,786943904	0,921111111
40000	629806868	1258012899	1481650000	0,393629293	0,786258062	0,92603125
50000	985213507	1968005009	2324200000	0,394085403	0,787202004	0,92968
60000	1418606757	2832826343	3349050000	0,394057433	0,786896206	0,930291667
70000	1931602346	3862090775	4564850000	0,39420456	0,788181791	0,931602041
80000	2521462430	5040283928	5988400000	0,393978505	0,787544364	0,9356875
90000	3194401800	6385629590	7534300000	0,394370593	0,788349332	0,930160494
100000	3940468840	7878875497	9286550000	0,394046884	0,78788755	0,928655
				0,39422953	**0,787514374**	**0,913396906**

- Algoritmo 3:

Tamaño	NumComp	NumAsign	TiempoEmpleado	Constante Comp	Constante Asig	Constante Tiempo
10000	126459	280593	1600000	3,161475		40
20000	273926	603187	2100000	3,184423269	7,012122685	24,41275697
30000	429087	939187	2800000	3,194664425	6,992491727	20,84672896
40000	589877	1290374	4100000	3,204418245	7,009763032	22,27263447
50000	754143	1647374	5250000	3,209822575	7,011638714	22,34532246
60000	921081	2004374	6500000	3,212822114	6,991455813	22,67264631
70000	1090902	2370748	8150000	3,216511649	6,990122448	24,03017864
80000	1263712	2748748	8850000	3,221723452	7,007693126	22,56230261
90000	1437955	3126748	10250000	3,224968852	7,012503804	22,98815382
100000	1613199	3504748	11250000	3,226398	7,009496	22,5
				3,205722758	**7,004143039**	**24,46307243**

Ahora que ya tenemos las constantes de todos los algoritmos, lo único que queda es escribir las formulas para todas las comparaciones, asignaciones y el tiempo empleado de los tres algoritmos. Estas fórmulas seguirán una estructura obtenida de despejar el número de comparaciones de la anterior formula, utilizada para calcular las constantes.

X = Numero de Comparaciones / f(n)

Si despejamos el número de comparaciones:

Numero de Comparaciones = X * f(n)

Como hemos dicho antes, esta fórmula también sirve para las asignaciones y el tiempo empleado. Sustituimos en las formulas los constantes y el f(n) de cada algoritmo.

- Algoritmo 1:
  - Numero de Comparaciones = 1,027539993 * $n^{1.3}$
  - Numero de Asignaciones = 1,762132325 * $n^{1.3}$
  - Tiempo empleado = 4,333046847 * $n^{1.3}$
- Algoritmo 2:
  - Numero de Comparaciones = 0,39422953 * n^2
  - Numero de Asignaciones = 0,787514374 * n^2
  - Tiempo empleado = 0,913396906 * n^2
- Algoritmo 3:
  - Numero de Comparaciones = 3,205722758 * n * log(n)
  - Numero de Asignaciones = 7,005211235 * n * log(n)
  - Tiempo empleado = 24,46307243 * n * log(n)

4 - ¿Qué algoritmo es mejor?

Finalmente, después de haber realizado el análisis basado en datos de los tres algoritmos, podemos concluir cuál de ellos es el mejor.

En primer lugar, podemos descartar el algoritmo dos, ya que sabemos que tiene un orden mayor que el algoritmo uno y tres. Incluso si nos fijamos solo en los datos, se ve claramente que el algoritmo dos es el peor de los tres por orden de magnitud.

Ahora solo nos quedaría debatir entre el algoritmo uno y tres. Aunque el algoritmo tres sea de mejor orden que el algoritmo uno ($O(n * \log(n)) \subset O(n^{1.3})$) y tenga mejores resultados en comparaciones y asignaciones, el algoritmo tres es recursivo, lo que genera un sobrecoste en el tiempo de ejecución

Esto quiere decir que además de que el algoritmo tres produce un gasto adicional de memoria, acaba teniendo unos tiempos ligeramente peores a los obtenidos con el algoritmo 1.

Es por todas estas razones que finalmente podemos llegar a la conclusión de que el mejor algoritmo de los tres es el algoritmo uno.