# ADA PRACTICA 1

## INTRODUCCÍON

El objetivo de la práctica es comparar tres algoritmos de ordenación. Para ello es preciso obtener una estimación, basada en datos experimentales, de la eficiencia en el caso promedio de los algoritmos. Una vez hecha la estimación para los tres algoritmos, hay que determinar, si fuera posible, la curva que más se aproxima a los datos obtenidos. Por último, hay que determinar cuál de los algoritmos es más eficiente. Los algoritmos por analizar son los siguientes:

- Algoritmo 1

```
public void ordena1(int v[], int tam) {
        int i, j, temp;
        i = 1; j = 2;
        while (i < tam) {
                if (v[i - 1] <= v[i]) {
                        i = j; j = j + 1;
                        }
                 else {
                        temp = v[i - 1]; v[i - 1] = v[i];
                         v[i] = temp; i = i - 1;
                        if (i == 0) {
                                i = 1;
                                }

                         }

                }

        }
```

- Algoritmo 2

```
public void ordena2(int[] v, int tam) {

        int k; int n = tam;

         for (k = n / 2; k >= 1; k--) {

                f(v, k, n); }

        k = n;

        while (k > 1) {
```

```
                    g(v, 1, k--); f(v, 1, k);
                    }
        }
        private void f(int[] v, int k, int n) {
            while (2 * k <= n) {
                    int j = 2 * k;
                    if ((j < n) && (v[j - 1] < v[j])) { j++; }
                    if (v[k - 1] >= v[j - 1]) { break; }
                    g(v, k, j); k = j;
            }
        }
        private void g(int[] v, int i, int j) {
             int temp = v[i - 1];
             v[i - 1] = v[j - 1];
            v[j - 1] = temp;
             }
```

- Algoritmo 3

```
public void ordena3(int[] v, int tam) {
        int m = h(v, tam);
        int[] c = new int[m + 1];
        int[] w = new int[tam];
        for (int i = 0; i < tam; i++) {
            c[v[i]] = c[v[i]] + 1;
             }
        for (int i = 1; i < m + 1; i++) {
             c[i] = c[i] + c[i - 1];
            }
        for (int i = tam - 1; i >= 0; i--) {
```

```
                w[c[v[i]] - 1] = v[i]; c[v[i]] = c[v[i]] - 1;
            }
        for (int i = 0; i < tam; i++) {
                v[i] = w[i];
            }
}
private int h(int[] v, int tam) {
        int i;
        int m = v[0];
        for (i = 1; i < tam; i++) {
            if (v[i] > m) { m = v[i]; }
            }
        return m;
}
```

## DESAROLLO DE LA PRACTICA

1. Pasamos las funciones a un código general que las ejecutara 1 a uno cogiendo las variables necesarias que más tarde guardaremos
2. Antes de la ejecución de cada función se lanzará un contador de tiempo con el instante inicial y final
3. Dentro de las funciones localizamos las operaciones de comparación. Para localizarlas nos fijamos en los comparadores como >, <, ==, !=... que estén comparando el vector v. Una vez localizadas usamos el contador comparaciones, que se inicializara como 0 antes de entrar a cada función y añadimos 1 al contador por cada una de estas operaciones.
   A la hora de incrementar el contador tenemos que tener en cuenta si la comparación está dentro de otro bucle o un if, ya que no siempre se va a llegar a esa comparación. Pero en este caso se cumpla o no la comparación debemos sumar al contador.
4. Hacemos un proceso análogo al anterior, pero con el contador de asignaciones. En este caso cada vez que se copie o modifique un valor del vector v con un operador =, ++ deberemos actualizar este contador.

5. Asignamos los tamaños a los arrays que se usaran para ejecutar las funciones, usaremos los tamaños de 20000 a 200000 que se irá incrementando en cada ejecución necesaria para tomar los datos. En cada una de estas se randomizará el contenido del array antes de tomar los datos con cada función.
   En este paso para la tercera función tuvimos que cambiar los tamaños a 100000 hasta 1000000, esto fue debido a que iba muy rápido para nuestro procesador, y el tiempo nos daba 0 en los cálculos, lo cual no nos servía para hacer comparaciones. A la hora de hacer estas deberemos tener en cuenta el cambio de tamaño.
6. Una vez recogidos los datos de cada ejecución, se guardarán en archivos csv para posteriormente ser analizados.
7. Hemos pasado los datos csv a excel para poder crear graficas con ellos y comprarlos
8. Creamos graficas con los valores de cada csv, una por cada medida: asignaciones, comparaciones y tiempo. Y lo hacemos por cada una de las funciones.
9. Una vez tengamos las gráficas buscaremos una fórmula que la aproxime lo máximo posible, dándonos el orden de n real aproximado de cada una de las gráficas.
10. Una vez que tengamos todo esto podemos comenzar a sacar conclusiones de cada una de las funciones, comparando los órdenes y los distintos valores dados en los pasos anteriores.

## TABLAS DE DATOS

- Algoritmo 1

| Tamaño | Comparaciones promedio | Asignaciones promedio | Tiempo promedio |
|---|---|---|---|
| 20000 | 99869985 | 299549958 | 200976500 |
| 40000 | 399923976 | 1199651931 | 824274200 |
| 60000 | 901463659 | 2704210980 | 1754868850 |
| 80000 | 1600845509 | 4802296531 | 3061580900 |
| 100000 | 2498919033 | 7496457103 | 4860570550 |
| 120000 | 3599472651 | 10798057956 | 7207361300 |
| 140000 | 4899643572 | 14698510721 | 9328022100 |
| 160000 | 6401189135 | 19203087410 | 11999547250 |
| 180000 | 8098903923 | 24296171772 | 15070546600 |
| 200000 | 9997964886 | 29993294661 | 18809551800 |

- Algoritmo 2

| Tamaño | Comparaciones promedio | Asignaciones promedio | Tiempo promedio |
|---|---|---|---|
| 20000 | 510798 | 805369 | 2479350 |
| 40000 | 1101500 | 1730464 | 5048300 |
| 60000 | 1721329 | 2699307 | 7491100 |
| 80000 | 2362898 | 3700598 | 10268900 |
| 100000 | 3019556 | 4724743 | 13406400 |
| 120000 | 3682803 | 5758904 | 16101750 |
| 140000 | 4356921 | 6808782 | 18653100 |
| 160000 | 5045896 | 7881454 | 21625150 |
| 180000 | 5740486 | 8962238 | 24860250 |
| 200000 | 6439225 | 10049569 | 27527550 |

- Algoritmo 3

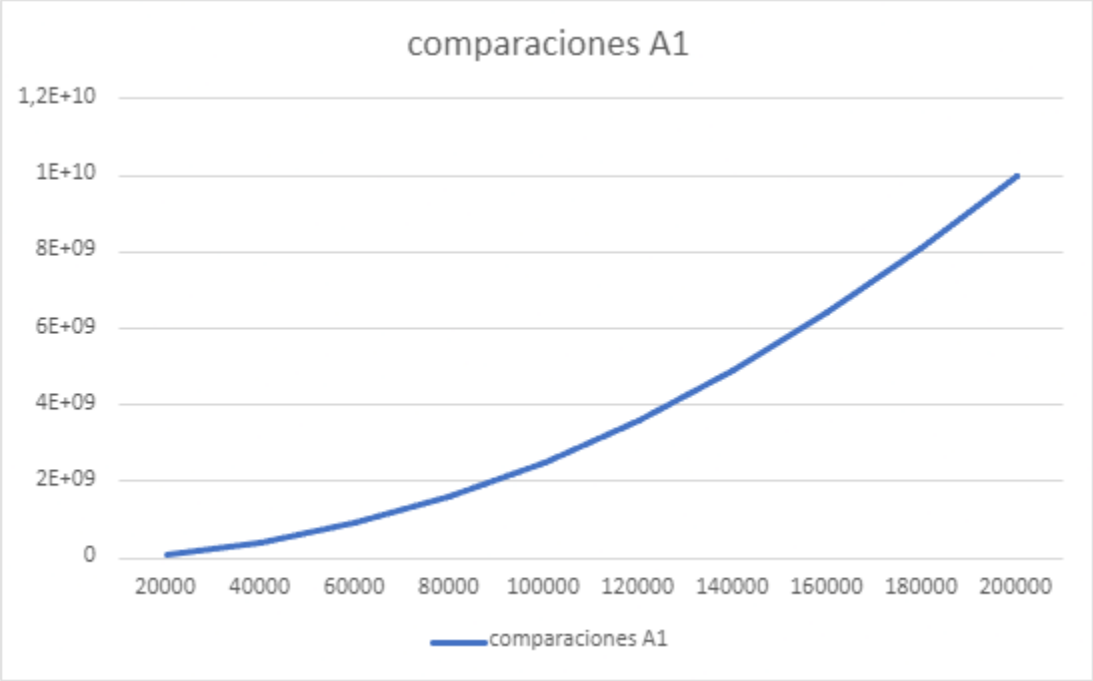
| Tamaño | Comparaciones promedio | Asignaciones promedio | Tiempo promedio |
|---|---|---|---|
| 100000 | 99999 | 400011 | 3142350 |
| 200000 | 199999 | 800013 | 4455800 |
| 300000 | 299999 | 1200013 | 7254050 |
| 400000 | 399999 | 1600013 | 8495950 |
| 500000 | 499999 | 2000014 | 11221600 |
| 600000 | 599999 | 2400014 | 13852550 |
| 700000 | 699999 | 2800014 | 16226800 |
| 800000 | 799999 | 3200014 | 19934200 |
| 900000 | 899999 | 3600014 | 28328300 |
| 1000000 | 999999 | 4000015 | 37454100 |

## Graficas resultantes

Nota: cuando aparezca n en las gráficas resultantes, significara tamaño

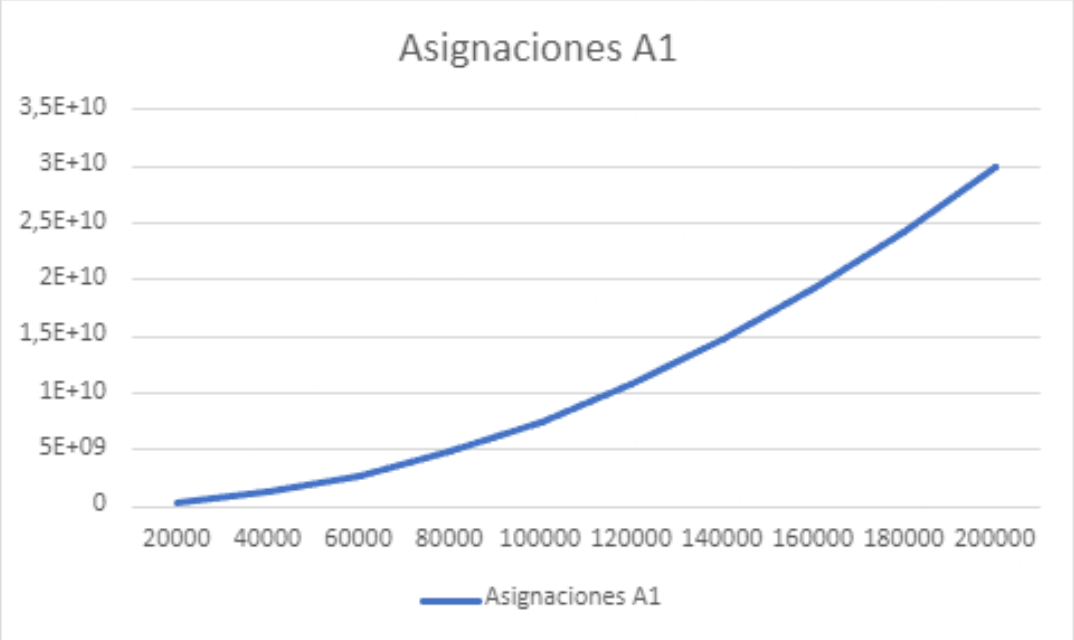
- **Algoritmo1**

Comparaciones



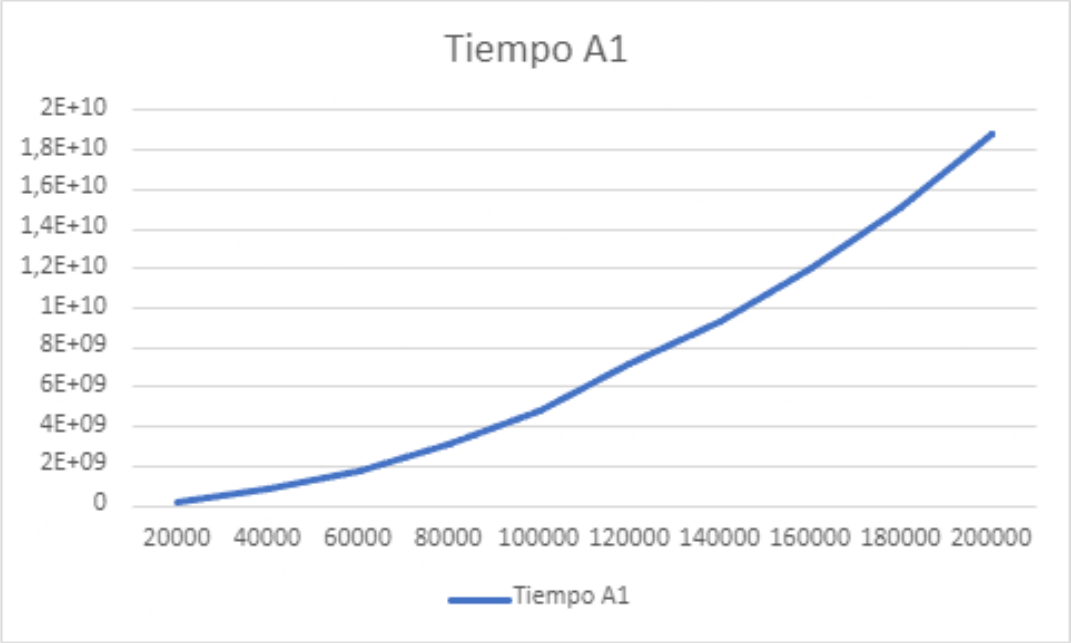

Comparaciones en función al tamaño del vector

## Asignaciones




Asignaciones en función al tamaño del vector

### Tiempo




Tiempo en función al tamaño del vector

- **Algoritmo 2**

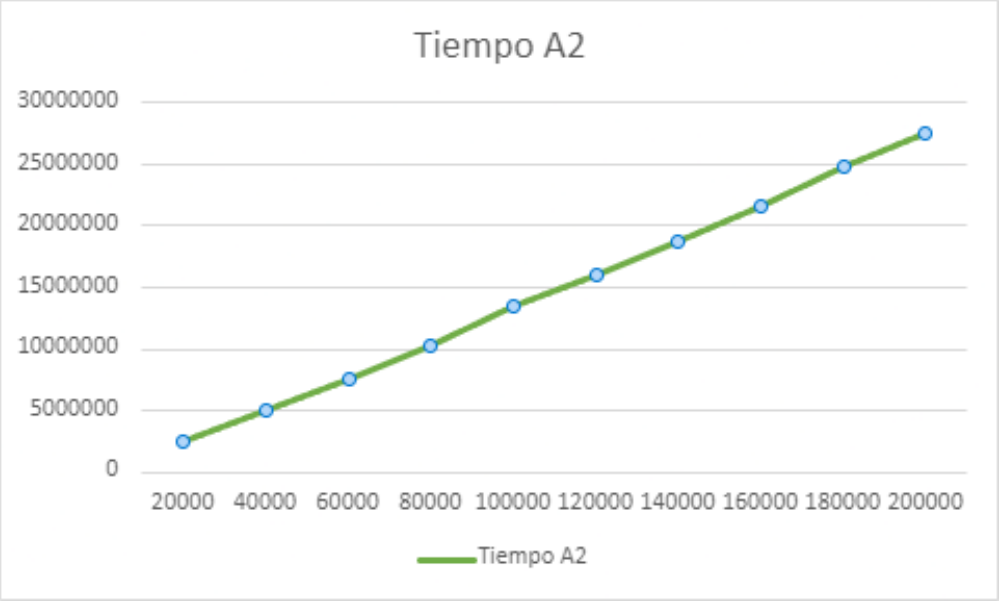
### Comparaciones


Comparaciones en función al tamaño del vector

### Asignaciones

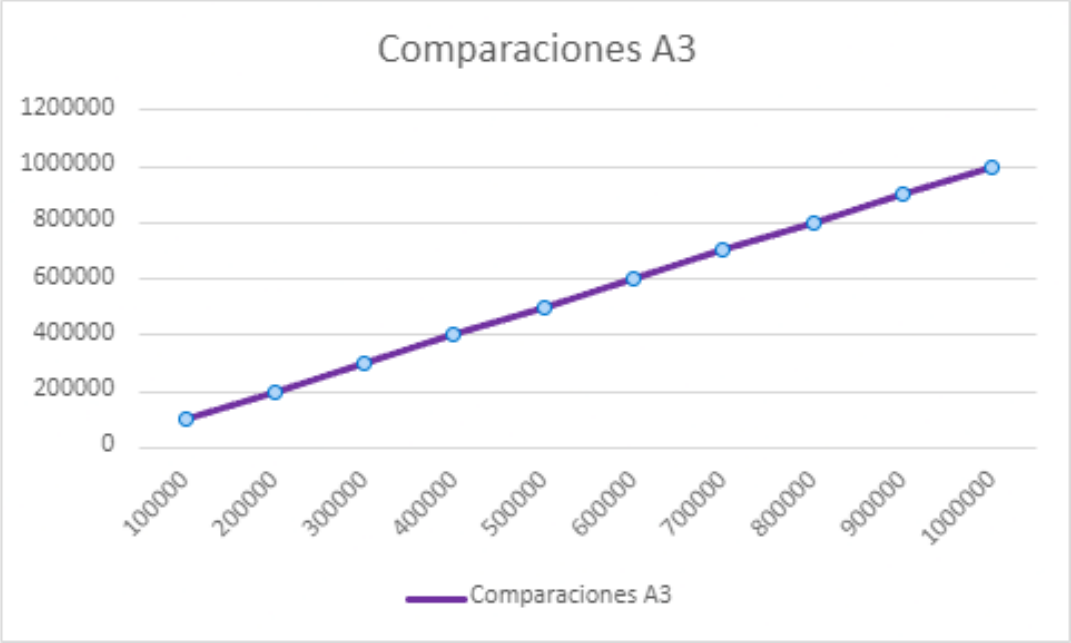

Asignaciones en función al tamaño del vector

- Tiempo




Tiempo en función al tamaño del vector

- **Algoritmo 3**
  - Comparaciones

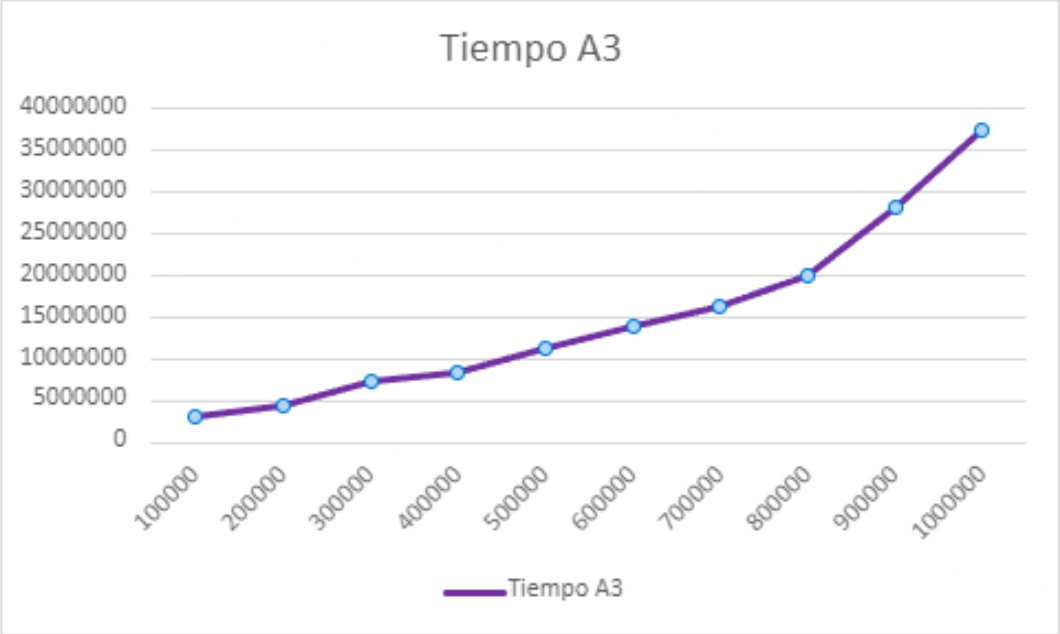



Comparaciones en función al tamaño del vector

- Asignaciones


Asignaciones en función al tamaño del vector

- Tiempo




Tiempo en función al tamaño del vector

## ANÁLISIS DE LOS ALGORITMOS

**Nota: cuando nos referiremos a “n” en las fórmulas esta significa el tamaño del vector**

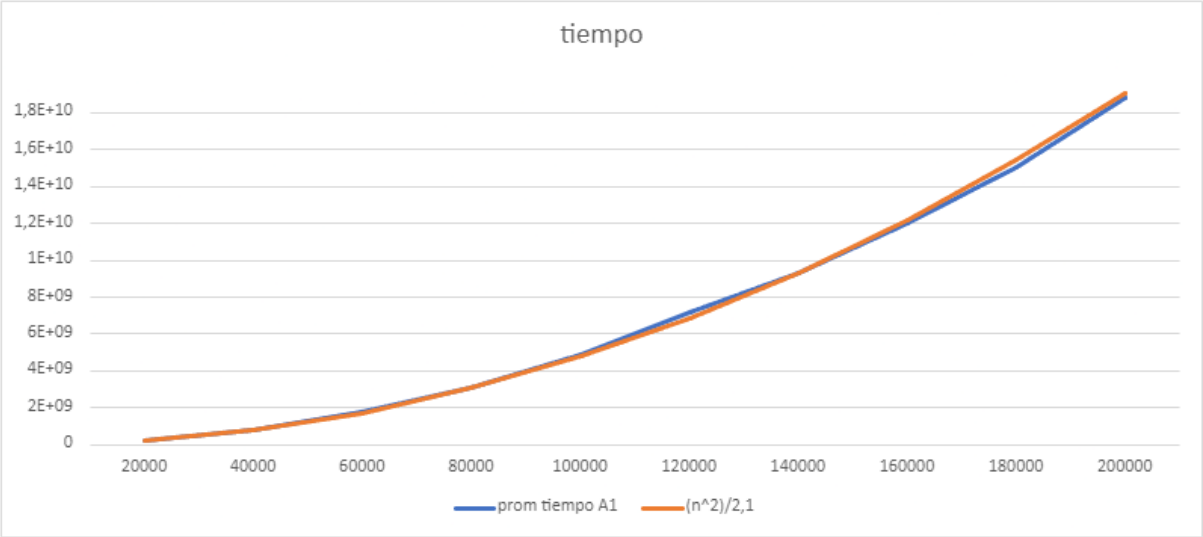
- Algoritmo 1


comparaciones
1,2E+10
1E+10
8E+09
6E+09
4E+09
2E+09
0
20000 40000 60000 80000 100000 120000 140000 160000 180000 200000
prom comparaciones A1
0,24(n^2) + 18n

O(n^2)


asignaciones
3,5E+10
3E+10
2,5E+10
2E+10
1,5E+10
1E+10
5E+09
0
20000 40000 60000 80000 100000 120000 140000 160000 180000 200000
prom asignaciones A1
0,73n^2

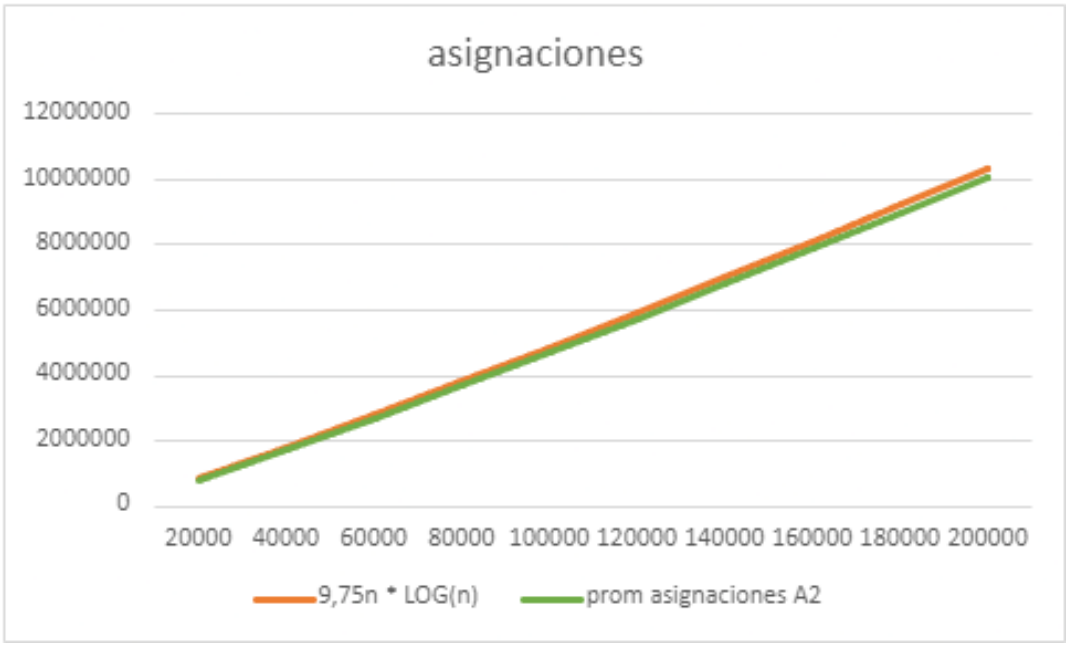
O(n^2)



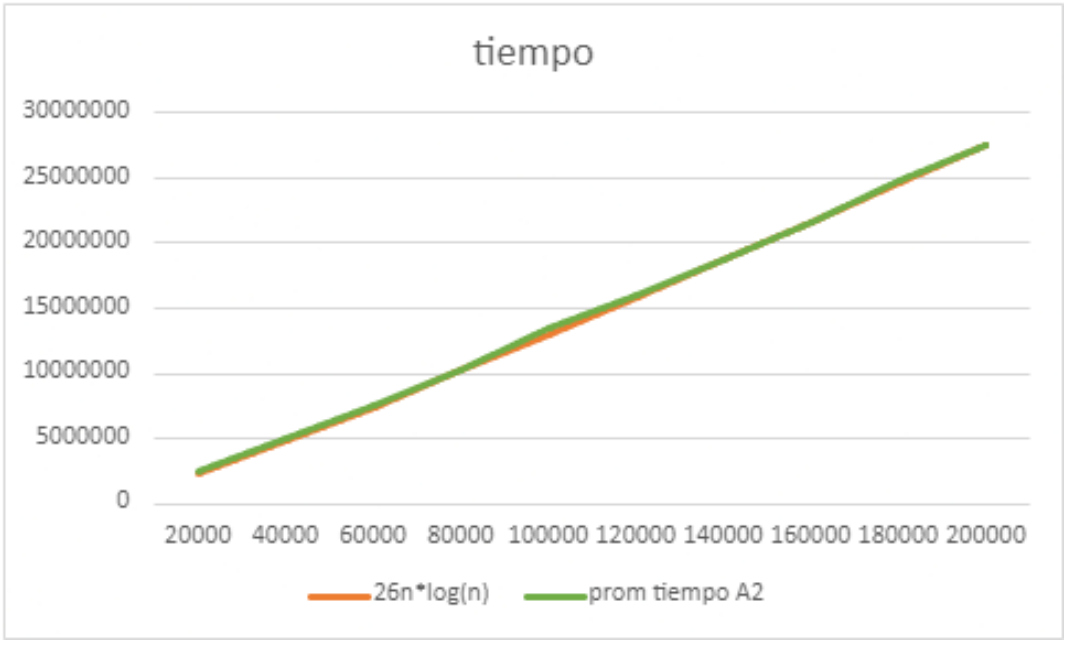

O(n^2)

- Algoritmo 2


O(nlogn)

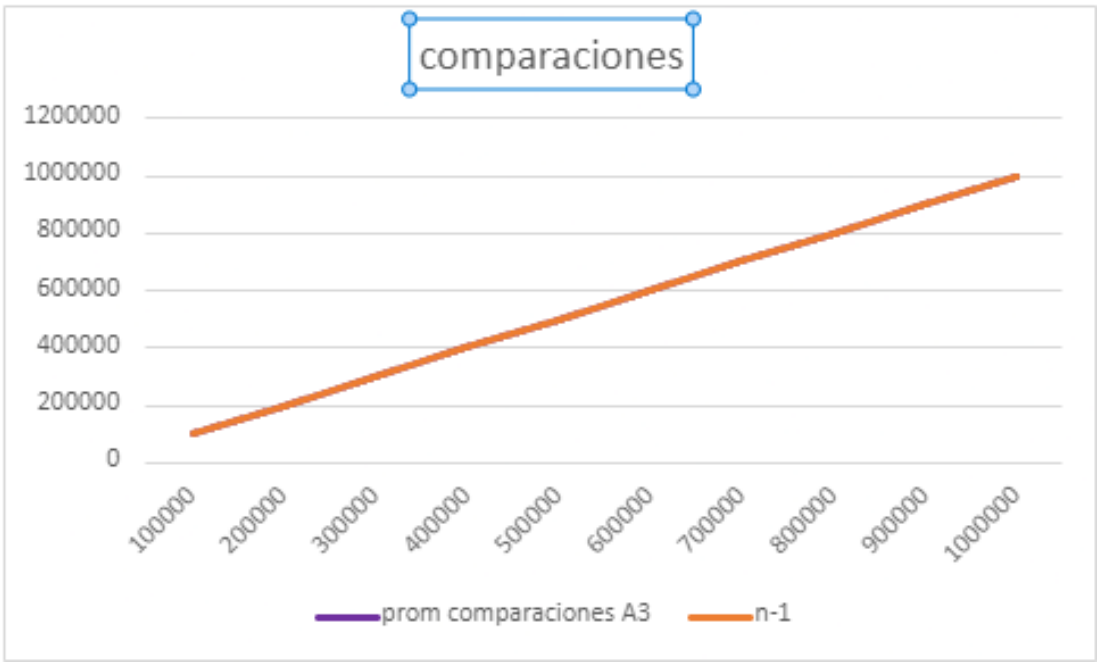



O(logn)



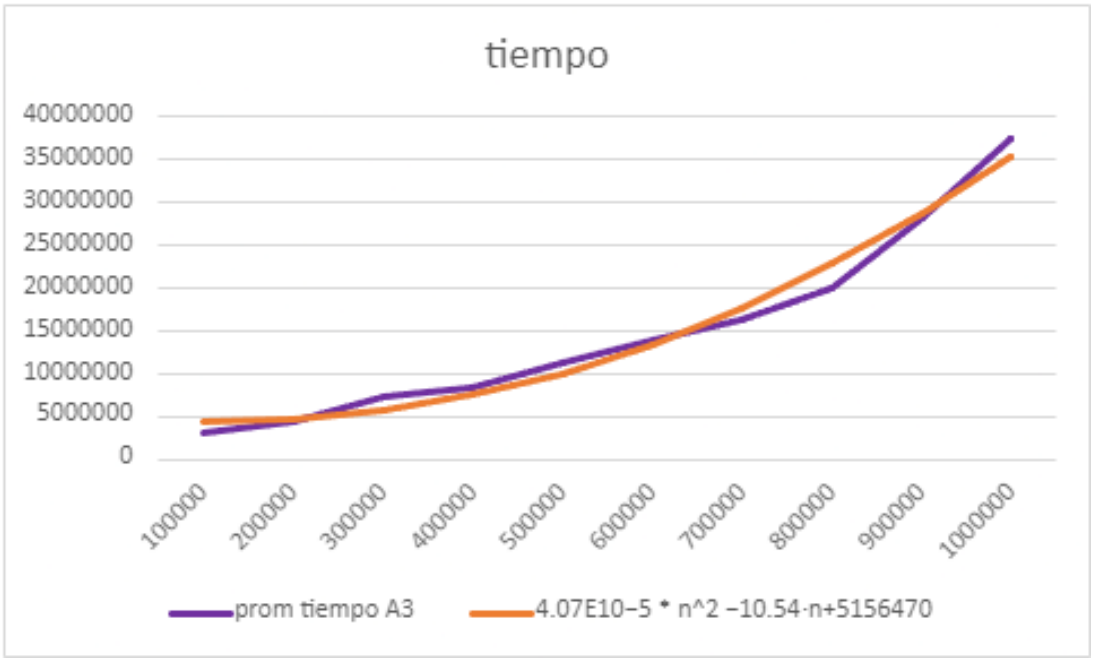

O(nlogn)

- Algoritmo 3



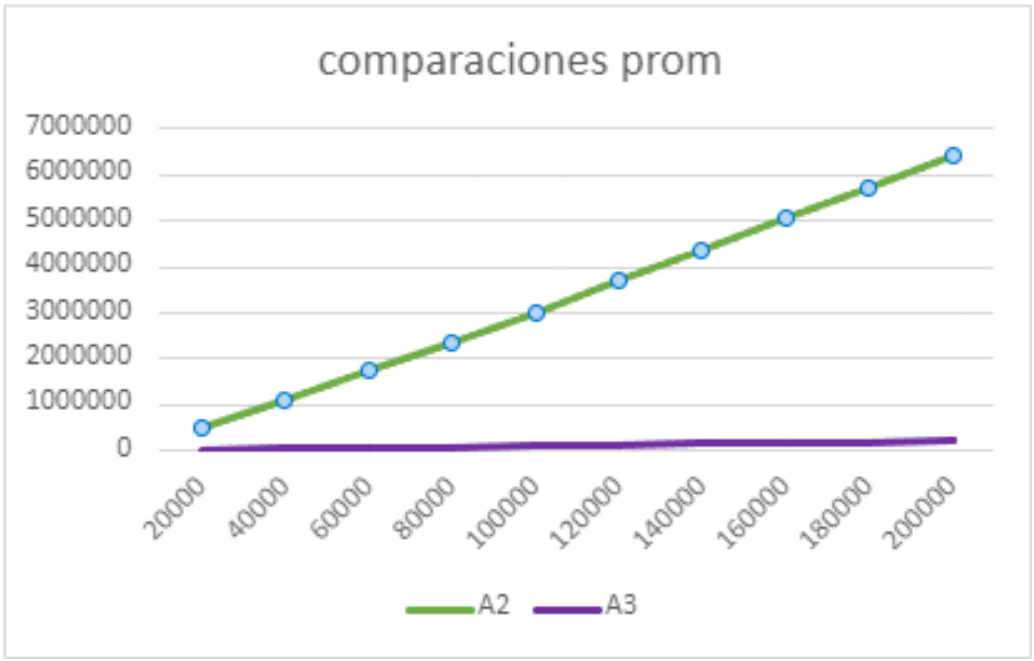

O(n)


O(n)




O(n^2)

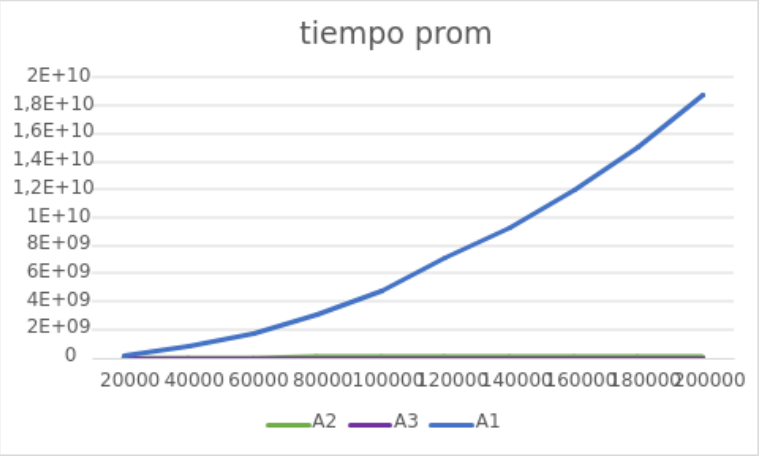
## Comparaciones entre Algoritmos




asignaciones prom
12000000
10000000
8000000
6000000
4000000
2000000
0
20000 40000 60000 80000 100000 120000 140000 160000 180000 200000
A2
A3


tiempo prom
30000000
25000000
20000000
15000000
10000000
5000000
0
20000 40000 60000 80000 100000 120000 140000 160000 180000 200000
A2
A3

Nota: La razón por la que solo hemos comprado con 2 de los algoritmos es debido a que la diferencia del 1 con los demás es tan grande que hace imposible el sacar conclusiones comparándolos de forma visual. Como se puede observar en la grafica




## CONCLUSIONES

- El algoritmo 1 es el más lento y por tanto menos optimo de los 3, dado a que todos sus órdenes son de n^2, incluyendo el número de comparaciones y asignaciones. Aumentando por mucho el número de operaciones a realizar por ejecución. Dado por esto no pudimos comparar los 3 en la misma grafica.
- El algoritmo 3 es notablemente más rápido, para poderse comparar con otros tuvimos que aumentar 5 veces el tamaño del vector sobre el que tomábamos las medidas
- El algoritmo 3 minimiza el número de comparaciones y asignaciones comparando con el resto, dado que su orden para ello es O(n), mientras que los otros dos rondan por O(n^2) y O(nlogn)
- En cuanto tiempo del algoritmo 3 es O(n^2) y el tiempo del algoritmo 2 es O(nlogn), si bien en el orden bajo el 3 es más optimo. Llegaría un punto hipotético muy grande, donde el 2 llegaría a ser más rápido por virtud de su O(nlogn). Podemos sacar como conclusión de esto que el que el orden sea mayor o menor no determina siempre cual es mejor para el tamaño que estemos tratando, ya que dependerá completamente del número de elementos totales, cual es el mejor algoritmo.

Debido a las anteriores conclusiones, hemos determinado que el algoritmo 3 es el más eficiente y predecible en cuanto a asignaciones y comparaciones realizas, por lo que, entre los 3 algoritmos, es el mejor.

# REPARTO DE TRABAJO

El reparto de la cantidad de trabajo por parte de los participantes es:

- Roger Daniel González Niebla 50%
- Mario García Benito 50%