Proyecto integrador I

Taller diseño de experimentos

Profesor: Juan Manuel Reyes

Integrantes

Santiago Rodas Rodriguez

Juan Fernando Martinez

Gonzalo De Varona

Santiago de Cali - Marzo 2021

Etapas del diseño de experimento

1. Planeación y realización

A) Entender y delimitar el problema u objeto de estudio.

Para realizar el experimento, primero necesitamos recolectar la información necesaria acerca de los dos métodos de ordenamiento seleccionados:

1. **Burbuja:** Funciona revisando cada elemento de la lista que va a ser ordenada con el siguiente, intercambiandolos de posición si están en el orden equivocado. Es necesario revisar varias veces toda la lista hasta que no se necesiten más intercambios, lo cual significa que la lista queda ordenada. Este algoritmo obtiene su nombre de la forma con la que suben por la lista los elementos durante los intercambios, como si fueran pequeñas "burbujas". Dado que solo usa comparaciones para operar elementos, se lo considera un algoritmo de comparación, siendo uno de los más sencillos de implementar.

2. **Inserción:** Es una manera muy natural de ordenar para un ser humano. Requiere $O(n^2)$ operaciones para ordenar una lista de n elementos. Inicialmente se toma la primera posición (k), después se toma la posición $k + 1$ y se compara con todas las posiciones restantes, deteniéndose cuando se encuentra un elemento mayor (todos los elementos menores han sido desplazados a la izquierda) o cuando ya no se encuentran elementos (todos los elementos fueron desplazados y este es el más pequeño). En este punto se *inserta* el elemento $k + 1$ debiendo desplazar todos los demás.

Tomando en cuenta esto, vemos que ambos métodos de ordenamiento cumple su funcionalidad final (ordenar n elementos), pero no lo hace de la mejor manera posible, ya que sus complejidades temporales son $\Omega(n^2)$ y $O(n^2)$ respectivamente. Lo cual, nos indica que tienen un tiempo cuadrático, validando 2 veces la cantidad total de elementos. Es decir, que en un sistema donde se tenga una gran cantidad de estos, no sale rentable utilizar los métodos anteriormente descritos.

Con lo anterior en mente no es la finalidad del experimento determinar si estos algoritmos son eficientes ya que es evidente que existen otros mucho mejores. El objetivo del estudio es entonces determinar cuál de los dos es más eficiente teniendo en cuenta tres factores comunes que impactan en el tiempo de ejecución de los algoritmos de ordenamiento bubble sort e insertion sort.

B) Elegir la(s) variables(s) de respuesta que será medida en cada punto del diseño y verificar que se mide de manera confiable.

Tomando en cuenta los resultados que esperamos tener, y el diseño que queremos llevar a cabo, decidimos tomar el tiempo de ejecución de los algoritmos como una variable de respuesta, ya que en pocas palabras esta nos servirá para verificar el correcto funcionamiento de cada método en la menor duración posible. Es por esta razón, que realizando una investigación pertinente encontramos la siguiente información:

En C#, el ecosistema .Net nos provee de una herramienta de medición temporal con una gran precisión, en este caso de microsegundo. Está disponible en todos los entornos a partir de .Net Framework 2.0 (incluyendo .Net Core y .Net Standard).

Para usarla y poder medir tiempos en C#, simplemente es necesario instanciarla y llamar a sus métodos Start y Stop, después nos devuelve un TimeSpan con el tiempo transcurrido.

```
Stopwatch timeMeasure = new Stopwatch();
timeMeasure.Start();
int operacion = 10 / 4;
timeMeasure.Stop();
Console.WriteLine($"Tiempo: {timeMeasure.Elapsed.TotalMilliseconds} ms");
```

Pero no se queda ahí, la clase también tiene dos propiedades estáticas (ReadOnly) que nos pueden dar mucha información sobre la medida:

- **IsHighResolution (bool)**: Nos va a indicar si el hardware permite mediciones de alta precisión.

- **Frequency (long)**: Nos va a dar el valor de ticks (mediciones) por segundo que soporta nuestro hardware.

```
using System.Diagnostics;
//------
Stopwatch timeMeasure = new Stopwatch();
timeMeasure.Start();
int operacion = 10 / 4;
timeMeasure.Stop();
Console.WriteLine($"Tiempo: {timeMeasure.Elapsed.TotalMilliseconds} ms");
Console.WriteLine($"Precision: {(1.0 / Stopwatch.Frequency).ToString("E")}
if (Stopwatch.IsHighResolution)
    Console.WriteLine("Alta precisión");
else
    Console.WriteLine("Baja precisión");
```

C) Determinar la unidad experimental.

Debido a la naturaleza del problema, donde buscamos experimentar con 2 algoritmos de ordenamiento, la unidad experimental es un conjunto de ejecuciones de los algoritmos con diferentes condiciones iniciales porque a partir de cada una se puede evaluar el comportamiento de cada algoritmo con condiciones que producen diferentes resultados, los cuales se podrán analizar a detalle en base a la variable de respuesta.

D) Determinar cuáles factores deben estudiarse o investigarse, de acuerdo a la supuesta influencia que tienen sobre la respuesta.

Los factores estudiados elegidos para realizar el experimentos son los siguientes 3:

A) Tamaño del arreglo, el cual está lleno en todos los casos. Los tamaños posibles pueden ser cualquier número entero positivo, limitado por los recursos del sistema y el lenguaje de programación. Los ordenamientos toman más tiempo conforme aumenta el tamaño del arreglo.

B) Hardware del computador: Procesador y cantidad de memoria ram, el hardware es evidente que influye en el tiempo que tardan los ordenamientos. La memoria ram usualmente toma valores entre 4 y 32GB siendo los más comunes entre 4-16 GB. Los modelos de procesadores son bastante variados con Intel y AMD las empresas más grandes que los fabrican.

C) Orden inicial en el arreglo: La forma como se ordenan los datos dentro de la estructura. Pueden estar ordenados de menor a mayor, de mayor a menor y aleatoriamente. El tiempo de ejecución de los ordenamientos también depende de este factor en relación con el desorden de los elementos.

A continuación se muestra un ejemplo del arreglo (tamaño = 10) en concreto:

Factores controlables no estudiados: Modo de consumo energético del sistema (Balanceado, ahorrador de energía, máximo rendimiento), se fija en máximo rendimiento.

Los factores no controlables que inciden en el proyecto son: el desgaste de los componentes internos de los computadores donde se realizará el experimento, la totalidad de tareas de los computadores que se están ejecutando en el momento que se realice el experimento (sólo algunas tareas se pueden controlar pero algunas tareas internas no), frecuencia a la cual trabaja el procesador, al ser dinámica.

E) Seleccionar los niveles de cada factor, así como el diseño experimental adecuado a los factores que se tienen y al objetivo del experimento.

Para las variables de estudio escogidas se escogen los siguientes niveles que resultan en un total de 45 tratamientos para ambos algoritmos.

1. Tamaño del arreglo, de los tamaños posibles se escogieron 5:
    a. 10
    b. 100
    c. 1,000
    d. 10,000
    e. 100,000

2. Hardware del computador, el hardware del computador de cada miembro del equipo.
    a. (Santiago) Intel Core i5-8250U y 8GB de ram.
    b. (Gonzalo) Intel Core i5-8250U y 16 GB de ram.
    c. (Juan) Intel Core i7-7700HQ y 16 GB de ram.

3. Orden de los elementos en el arreglo:
    a. Ordenado de mayor a menor
    b. Ordenado de menor a mayor
    c. En orden aleatorio

Este experimento se realizará con 100 repeticiones para cada tratamiento, pero el usuario tendrá la opción de realizar una mayor cantidad de repeticiones. Claramente, tomando en cuenta cada uno de los aspectos anteriormente nombrados. Además, como las condiciones pueden ser distintas, se espera que cada experimento sea diferente. Es decir, un caso puede ser un arreglo de 10.000 posiciones, 8 GB de RAM, procesador Intel Core i5-8520U y números ordenados. Mientras tanto, otro caso puede ser de 1.000.000 posiciones, 16 GB de RAM, procesador Intel Core i7-7700HQ y números aleatorios.

De esta manera, no solo se piensa obtener resultados de alta precisión, sino también podremos saber la resolución de cada una de las medidas alcanzadas.

F) Planear y organizar el trabajo experimental.

En este diseño experimental se tomarán en cuenta los siguientes aspectos:

1. Las personas involucradas en el diseño y análisis del experimento serán todos los integrantes.

2. Las personas involucradas en la codificación y programación del experimento serán todos los integrantes.

3. Las personas involucradas en la toma de resultados e interpretación de los datos serán todos los integrantes.

Se piensa realizar en conjunto, ya que cuando los miembros colaboran, su capacidad para el éxito se multiplica. Sin embargo, los equipos exitosos no se crean de la noche a la mañana. En lugar de eso, mediante un proceso que implica la gestión de proyectos, el desarrollo del flujo de trabajo y la creación de procedimientos, los equipos encuentran el éxito, un paso a la vez. Además, cuando los empleados y miembros del equipo trabajan juntos en un entorno alentador, los beneficios son innegables. El hecho de forjar la comunicación dentro del grupo y otras condiciones abiertas ayudará a crear un sentido de cohesión, inversión y confiabilidad entre todos los miembros del equipo. Esto conduce a una colaboración y una gestión general de proyectos más efectivas.

G) Realizar el experimento.

En este subpunto se toma las siguientes decisiones:

1. En caso de un imprevisto o de un problema, este inmediatamente será notificado a la siguiente persona: Juan Fernando Martinez.

2. Los datos serán puestos en un archivo Excel, donde podremos visualizarlos por medio de gráficas comparativas y/o algunas herramientas de análisis de datos preestablecidas.

3. Todo se visualizará en el repositorio principal: link en GitHub.

2. Análisis

El análisis se llevará a cabo mediante el uso de indicadores de estadística descriptiva para comparar los diferentes tratamientos. Las comparaciones a realizar se limitan de acuerdo a lo siguiente:

1. La exactitud de medición: El cronómetro permite medir con una exactitud de milisegundos, y aquellas mediciones que tardaron menos de este valor no pueden ser analizadas con el mismo nivel detalle que las demás

2. Resultados exactos: En el caso de que los resultados puedan tomarse como constantes para un tratamiento dado. Está también ligado a lo anterior, por ejemplo cuando los tiempos son constantes y son 0.

3. Niveles para los cuales hay resultados invariables: Cuando los niveles analizados son limitados por los dos puntos anteriores y varios tienen resultados aparentemente equivalentes

Caso especial, tamaños de arreglo 10 y 100:

Estos tamaños tienen la peculiaridad de que los tiempos son en extremo pequeños tanto que en la mayoría de los tratamientos con estos dos niveles resultan en un tiempo de 0 ms. Para el tamaño 10 todos los resultados fueron 0, más para ambos algoritmos. Para el tamaño 100, los computadores todos también mostraron un tiempo mediano de 0 ms para ambos algoritmos y órdenes iniciales de los arreglos. Se puede usar tanto la media como la mediana debido a que realmente no hay asimetría.

Caso especial, insertion sort en orden creciente:

Para todos los tamaños de arreglo, ordenamientos iniciales y computador el resultado de tiempo de ordenamiento es 0 ms, la justificación de este comportamiento puede ser consultada en la siguiente sección.

Demás casos:

Una vez mencionados los casos especiales se prosigue con los demás casos. En las tablas 1 y 2 _se encuentra un resumen donde se tienen las medianas de todos los tratamientos que involucran tamaños de arreglos entre 1000 y 100000. Se escogió dicho estadístico debido a la asimetría alta que presentaron algunos de los tratamientos, principalmente para un tamaño de arreglo de 1000 lo que invalida el empleo de la media como medida representativa de las muestras. También en estos casos el coeficiente de variación pierde significado ya que los valores muy cercanos a en la media genera un aumento extremo y pierde relevancia su uso.

Tabla 1. Resumen de medianas para los tratamientos más significativos(Bubble sort)

Bubble sort (Mediana del tiempo de ejecución en ms)									
Tamaño	1000			10000			100000		
Participante	Santiago	Gonzalo	Juan	Santiago	Gonzalo	Juan	Santiago	Gonzalo	Juan
Creciente	1	1	1	152	153	147.5	23746.5	15949.5	15682
Decreciente	3	2	2	391.5	287	300	40061.5	29216	29891
Aleatorio	2	2	2	368	350	380	52126	36895	40870

Tabla 2. Resumen de medianas para los tratamientos más significativos(Insertion sort)

Insertion sort (Mediana del tiempo de ejecución en ms)									
Tamaño	1000			10000			100000		
Participante	Santiago	Gonzalo	Juan	Santiago	Gonzalo	Juan	Santiago	Gonzalo	Juan
Creciente	0	0	0	0	0	0	0	0	0
Decreciente	2	1	2	303.5	212.5	222.5	30645.5	22180.5	22536
Aleatorio	1	1	1	109	98	107	15301.5	10128.5	11202.5

Tiempo de ordenamiento en relación al tamaño del arreglo:

Una tendencia general que se observa para todos los ordenamientos iniciales y especificaciones de los sistemas es el aumento del tiempo cuando aumenta el tamaño del arreglo, esto era de esperarse porque un arreglo más grande aporta inherentemente mayor complejidad a los algoritmos de ordenamiento.

Este aumento en el tiempo no se da de manera lineal. Si se toma en cuenta que la relación entre los tamaños escogidos se basa en que un tamaño es 10 veces mayor al anterior, al observar los tiempos de ejecución de los algoritmos para cada sistema y algoritmo no se sigue este mismo patrón. Por ejemplo, para el equipo de Gonzalo los tiempos de ejecución medios (mediana) para tamaños 10000 y 100000 en orden aleatorio con el algoritmo de bubble sort, son 350 y 36895 ms cuya relación es ~100:1. Ya que 350ms*100 ~ 35000 ms que es muy cercano a 36895 ms, la relación es entonces aproximadamente n^2, donde n representa la razón de aumento en el tamaño de la entrada. El razonamiento anterior es aplicable para ambos algoritmos, aunque, debido a que no se tiene en cuenta toda la población de cómo ordenar un arreglo puede haber una diferencia en la relación entre los tiempos.

Tiempo de Ordenamiento en relación al ordenamiento inicial:

A nivel general al comparar los tiempos de ejecución en el caso que los elementos están ordenados inicialmente de forma creciente con los demás ordenamientos iniciales siempre son menores. Haciendo alusión que este es el mejor caso posible para estos algoritmos y era de esperarse que fuera el que tardara menos.

Aquí también es importante notar que el algoritmo insertion sort se comporta mucho mejor en estos casos al tardar medianamente 0ms en completarse y la diferencia con bubble es abismal en los tamaños más grandes (>= 10000).

En el ordenamiento de manera descendente y el creciente se observa un comportamiento distinto. El ordenamiento de forma descendente es el peor caso posible para ambos algoritmos por lo que se espera que este sea el que tarde más de los tres casos posibles. No obstante, para bubble sort esto no se cumple, se observa que el orden decreciente tarda un tiempo medio menor que el aleatorio, lo cual es desconcertante ya que en los tres equipos y para los tamaños entre 10,000 y 100,000 ocurre lo mismo. El insertion sort cumple con lo esperado y tiene un tiempo de ejecución medio mayor en el caso de ordenamiento decreciente en comparación con el aleatorio para todos los tamaños y sistemas medidos. Por último, también es notorio que insertion sort tarda significativamente menos en ordenar los arreglos que bubble sort.

Tiempo de Ordenamiento en relación a las especificaciones del equipo

Con los tamaños de arreglos de 1000 y menos realmente no hay una diferencia notoria entre los equipos ya que las medianas son muy similares entre sí para todos los casos. En los tamaños más grandes se empieza a ver un mayor grado de diferencia con la tendencia general de que Gonzalo tiene el equipo que más rápido completa los ordenamientos en términos de la mediana, seguido de Juan y luego de Santiago. La tendencia se mantiene para ambos algoritmos pero los equipos de Gonzalo y Juan se acercan lo suficiente como para que no se note diferencia entre ambos, excepto en el caso de bubble sort con arreglo en orden decreciente donde la diferencia es grande.

3. Interpretación

Caso especial, tamaños de arreglo 10 y 100:

De las 5 opciones existentes, estas dos eran las que tenían menor nivel de elementos. Por ende, lógicamente podremos concluir que al tener un arreglo de tamaño n, donde n no sea un número superior 100, este realizará pocas comparaciones y los tiempos de respuesta serán mucho más rápidos, lo que se refleja en que son menores a 1ms. Además, como sabemos por la investigación anteriormente hecha, este tipo de métodos de ordenamiento funcionan mejor en arreglos de baja envergadura, así que por eso encontramos un buen rendimiento, en el caso de insertion sort este es bastante más rápido que algoritmos como quicksort en arreglos pequeños.

Caso especial, insertion sort en orden creciente:

Este es un caso especial, ya que podremos comprobar que ante los otros experimentos, este en particular retorna buenos valores de tiempo. Pero, ¿por qué ocurre esto?

Bueno, resulta que el arreglo ya se encuentra ordenado, entonces por mas que haga comparaciones, en ningún momento el sistema tiene que realizar algún tipo de cambio en particular. Es decir, como no hay cambios en las variables o en las posiciones, la complejidad temporal resulta siendo el mejor caso de todos que es O(n)

```
public static void InsertionSort(int[] numArray)
{
    int aux;
    for (int i = 1; i < numArray.Length; i ++)
    {
        aux = numArray[i];
        for (int j = i - 1; j >= 0 && numArray[j] > aux; j--)
        {
            numArray[j + 1] = numArray[j];
            numArray[j] = aux;
        }
    }
}
```

Figura 4. Implementación del método Insertion Sort

Tiempo de ordenamiento en relación al tamaño del arreglo:

Como lo hemos podido ver en los anteriores casos, generalmente el tiempo que tarda un algoritmo es basado en el número de elementos que debe de organizar. Por ende, claramente podemos concluir que entre más objetos de un mismo tipo se tenga, junto a un mayor número de repeticiones, los tiempos del sistema tardarán más en resultar. Y al mismo tiempo, si la cantidad es menor en ambos casos, los milisegundos en retornar la información serán mínimos. Entonces, al final podremos concluir que estos dos métodos elegidos son eficaces para cantidades pequeñas, pero en cantidades grandes se recomienda mirar o analizar otro tipo de métodos.

Tiempo de Ordenamiento en relación a al ordenamiento inicial:

Al fijarse en las desviaciones tomando por ejemplo el caso de Juan, con desviaciones 12.55(decreciente) y 15.61(aleatorio) y medias 300.44 384.26 los datos no se solapan en su mayoría(300.44 +- 12.55 no solapa con 384.26 +-) por lo que no podría argumentarse que son similares y se tiene que la diferencia no es producto de la variabilidad y hay algún factor que se desconoce que influencie estos resultados. También puede ser el caso de que no se hayan tomado las medidas suficientes.

Además, también hay que tener en cuenta la lógica básica de un algoritmo de ordenamiento: no puede ordenar algo que ya está ordenado. Por ende, en un caso concreto como orden inicial creciente, el tiempo de respuesta será mínimo y extremadamente eficaz. Mientras que en un caso contrario, el tiempo será, valga la redundancia, todo lo contrario.

Tiempo de Ordenamiento en relación a las especificaciones del equipo

Estos experimentos se realizaron en 3 computadores distintos, claro está, uno perteneciente a cada integrante del grupo de trabajo. Al final, logramos ver los siguientes aspectos:

1. Entre más capacidad tenga la RAM del sistema, la velocidad de ejecución de cada uno de los métodos será mayor. Por ende, tendrá mayor eficiencia un computador con 16 de RAM, que uno de 8.

2. 2 de los 3 procesadores son de la misma referencia y potencia. Por ende, se pudo visualizar una desviación estándar bastante común en estos casos. Sin embargo, al referenciar estos casos con el otro procesador, si se logró evidenciar algunos cambios importantes, como por ejemplo: al momento de tomar insertion en un tamaño global de 100.000 casos. (mira bien los datos)

4. Conclusiones finales

- Insertion es mejor que bubble.

- Ambos algoritmos muestran experimentalmente un comportamiento asintótico de aproximadamente $O(n^2)$ para los peores casos.

- Hay factores que afectan el comportamiento esperado de bubble sort, se desconoce cuales son, pero se sabe que no están relacionados con la implementación del método ni su correctitud pero sí con su complejidad.

- Este tipo de métodos funcionan mejor en ambientes de trabajo donde la cantidad de componentes no sea muy grande. En un caso así, se recomienda analizar o programar otro tipo de métodos mucho más eficientes.