Quiz #4

Estudiantes:
Nicolás Florez Jiménez
Jiacheng Tan

Problema 1. Bubble Sort vs Quick Sort

a. **¿Cuál algoritmo es más rápido y por qué?**

Quicksort es mucho más rápido que el bubble sort. Esto se debe a que Quicksort tiene una complejidad promedio de O(nlog(n)) mientras que Bubble Sort tiene una complejidad de O(n^2). Esto hace que para arreglos pequeños el Bubble Sort sea parecido al Quicksort o incluso más rápido, pero conforme los tamaños aumentan la diferencia entre ambos algoritmos aumenta más y más (Quicksort mantiene una velocidad de ejecución relativamente consistente).

b. **¿El tiempo de ejecución será el mismo si la implementación del algoritmo es iterativa o recursiva?**

No, el tiempo de ejecución no será necesariamente el mismo. Aunque la lógica del algoritmo sea la misma, por recursividad se puede dar una sobrecarga (límite de recursión) debido al uso de la pila de llamadas, lo cual implica más uso de memoria y operaciones adicionales para manejar las funciones.

c. **¿Es posible que exista un algoritmo de ordenamiento que sea muy eficiente en consumo de recursos pero que a la vez sea relativamente rápido?**

Sí, es posible. Un ejemplo de un algoritmo que es eficiente en consumo de recursos y relativamente rápido es Heap Sort con un uso eficiente de memoria (O(1)O(1)O(1)) y un tiempo de ejecución de O(nlog(n)) en su peor, mejor y caso promedio. El algoritmo se basa en un concepto de estructura de datos llamado "heap binario" (un árbol binario completo que satisface la propiedad del heap).

d. **Si se planea ejecutar el algoritmo en un sistema con extremadamente bajos recursos de memoria, ¿cuál de los dos algoritmos de ordenamiento escogería y por qué?**

Escogería Bubble Sort, a pesar de su lentitud. Esto es porque Bubble Sort no requiere memoria adicional significativa, ya que todo el proceso se realiza dentro del arreglo original, con solo unas pocas variables adicionales.

Quicksort, puede requerir espacio adicional debido a su naturaleza recursiva, especialmente en su versión más simple. Aunque se pueden optimizar algunos aspectos del consumo de memoria de Quicksort, en su implementación estándar utiliza más memoria que Bubble Sort, lo que podría ser un problema para este caso.

Pruebas de los Algoritmos y sus Tiempos en Microsegundos

Arreglos	BubbleSort	QuickSort
1000	0.03967547416687012 0.02277541160583496 0.023578643798828125 0.0233004093170166 0.02374434471130371 0.02315807342529297 0.024563074111938477 0.02460503578186035 0.02300262451171875 0.024722814559936523	0.0009894371032714844 0.0010797977447509766 0.0009100437164306641 0.0011608600616455078 0.0009169578552246094 0.0009016990661621094 0.0010488033294677734 0.000896453857421875 0.001007080078125 0.001287221908569336
2000	0.1958611011505127 0.12077951431274414 0.10543227195739746 0.10831761360168457 0.12375164031982422 0.11368656158447266 0.1119527816772461 0.1228630542755127 0.09982562065124512 0.10530591011047363	0.00180816650390625 0.0015130043029785156 0.0014524459838867188 0.0013260841369628906 0.0015535354614257812 0.001359701156616211 0.0014338493347167969 0.0013799667358398438 0.0013113021850585938 0.0013217926025390625
3000	0.4152348041534424 0.23899269104003906 0.24309563636779785 0.24229168891906738 0.23441529273986816 0.2161729335784912 0.1750497817993164 0.18505239486694336 0.18010330200195312 0.17441892623901367	0.002205371856689453 0.0022363662719726562 0.0020558834075927734 0.002791881561279297 0.0023000240325927734 0.002268075942993164 0.002315998077392578 0.0020606517791748047 0.0030257701873779297 0.0025987625122070312
4000	0.7191920280456543 0.46738314628601074 0.36058878898620605 0.3158383369445801 0.32041478157043457 0.3137052059173584 0.31591796875 0.30791807174682617 0.31598424911499023 0.31496167182922363	0.0033521652221679688 0.0031740665435791016 0.002740144729614258 0.0027251243591308594 0.0028328895568847656 0.00412297248840332 0.003966331481933594 0.0033750534057617188 0.004715681076049805 0.0036606788635253906
5000	0.8695499897003174 0.5001049041748047 0.5306649208068848 0.636699914932251 0.5284979343414307 0.5082361698150635 0.481351375579834 0.4963855743408203 0.5341756343841553 0.5042893886566162	0.004119873046875 0.0042133331298828125 0.00389862060546875 0.004357576370239258 0.003947257995605469 0.004335641860961914 0.004006147384643555 0.0044786930084228516 0.004074573516845703 0.004215717315673828

Problema 2. Aplicaciones de los algoritmos

1. **¿Cuál es la diferencia entre el algoritmo de búsqueda lineal y búsqueda por interpolación?**

Para el interpolation search se tiene esta fórmula:

$$mid = Lo + \frac{(Hi - Lo) * (X - A[Lo])}{A[Hi] - A[Lo]}$$

Donde la nomenclatura es:
A = lista
Lo = índice más bajo de la lista
Hi = índice más alto de la lista
A[n] = valor almacenado en el índice *n* de la lista

Esta técnica es muy similar a la búsqueda binaria y requiere que la lista esté ordenada para funcionar correctamente. Sin embargo, sigue un enfoque más estructurado y lógico. Los pasos para llevarla a cabo son los siguientes:

1. Comenzar buscando desde el centro de la lista.
2. Si el valor en el centro es el que se busca, se retorna el índice del elemento y se finaliza la búsqueda.
3. Si no es una coincidencia, calcular la posición de prueba (*probe position*).
4. Dividir la lista utilizando la fórmula de interpolación y encontrar un nuevo punto medio.
5. Si el dato que se busca es mayor que el valor en el medio, continuar la búsqueda en la sublista superior.
6. Si el dato es menor, buscar en la sublista inferior.
7. Repetir el proceso hasta encontrar una coincidencia o agotar las opciones.

Por otro lado la búsqueda lineal solo recorre la lista de posición en posición, es mucho más rudimentario pero para casos donde solo se requiere extraer un dato en una lista no ordenada conviene más ya que el Interpolation search implicaría tener que ordenar toda la lista previo a siquiera usarla. En casos donde la lista ya estaba ordenada o se van a buscar múltiples datos, resulta útil usar interpolation, a largo plazo funciona mejor.

2. **Suponga que se tiene que buscar un elemento en una lista desordenada, pero se desea optimizar el tiempo de búsqueda por sobre cualquier otra métrica ¿Cómo se podría hacer eso?**

Primero que todo lo que se haría es ordenar la lista, esto es gracias al hecho de que los algoritmos de búsqueda más eficientes tienen esto como requisito al no tener que revisar posiciones de forma innecesaria, si priorizamos el tiempo de búsqueda púramente este sería el mejor método para hacerlo y posterior a ello usar algoritmos como el previamente mencionado Interpolation search.

3. **Busque y explique alguna aplicación de la vida real donde el tiempo de búsqueda en una lista o en un arreglo sea crítico para que la aplicación se pueda dar.**

Quizá para arreglos de información pequeños no hace mucha diferencia el tiempo de ejecución, un humano no puede diferenciar entre micro o milisegundos, pero si hablamos de casos en un mundo con tanta información digital se esas diferencias pueden volverse problemas reales. Por ejemplo, para empresas como Amazon, buscar un producto en la barra de búsqueda implica que tiene que extraerse de una base de datos enorme (con millones de productos) para siquiera poder confirmar y mostrar lo que se busca. Si no se tuviera un sistema eficiente de búsqueda con con algoritmos tan buenos como ellos tienen se tendría que esperar mucho tiempo para que se muestran los resultados. El tiempo de retorno es crítico en Amazon, mejora la experiencia del usuario y por consiguiente los hace mucho más competitivos.