Laboratorio Nro. 1: Recursión

Agustín Nieto García
Universidad Eafit
Medellín, Colombia
anietog1@eafit.edu.co

David Immanuel Trefftz Restrepo
Universidad Eafit
Medellín, Colombia
ditrefftzr@eafit.edu.co

2) Actividades propuestas.

1. Explicación del método groupSum5.

En el problema groupSum5, se recorre un arreglo de números de manera recursiva, teniendo como punto de parada que el start sea igual o mayor que el tamaño del arreglo, ahí se pregunta si se restó exactamente la cantidad necesitada. En cada llamado recursivo se pregunta si el número en el que se encuentra es un múltiplo de 5, después se pregunta si el número que le sigue es 1. En el caso de que se cumplan las dos condiciones se ignora el 1 y se sigue recorriendo el arreglo. También para el recorrido se incluyen automáticamente todos los números 5. Luego para cada parada se toman dos opciones, una en la que se resta el número actual del objetivo, o se sigue recorriendo el arreglo sin contar el número. El programa buscará todas las opciones de sumas para ver si en alguna se resta suficiente para que el objetivo sea cero, en este caso retornará true. Si no encuentra ninguna combinación valida de sumas, el programa retornará false.

2. Cálculo de complejidades de las implementaciones en CodingBat.

*****Se hallan en el código.

3. Explicar qué son n, m en los cálculos de complejidad anteriores.

En los cálculos de complejidad de los ejercicios de CodingBat se utilizó únicamente la n, que indicaba siempre el tamaño del problema, definido por la cantidad de enteros dentro del arreglo. Motivo: sin importar cuál fuera la meta de la suma, en el peor de los casos siempre habría que revisar todos los espacios del arreglo recursivamente múltiples veces hasta retornar, finalmente, false; lo que implicaría que no importan ni el target, ni el start (que siempre debe ser 0 para funcionar correctamente) en el cálculo de la complejidad de los ejercicios, sino la cantidad de veces que se repetirá la recursión (el tamaño del arreglo: n).

3) Simulacro de preguntas de sustentación de Proyectos

1. Tablas con tiempos para los métodos arraySum, arrayMax y Fibonacci recursivos.

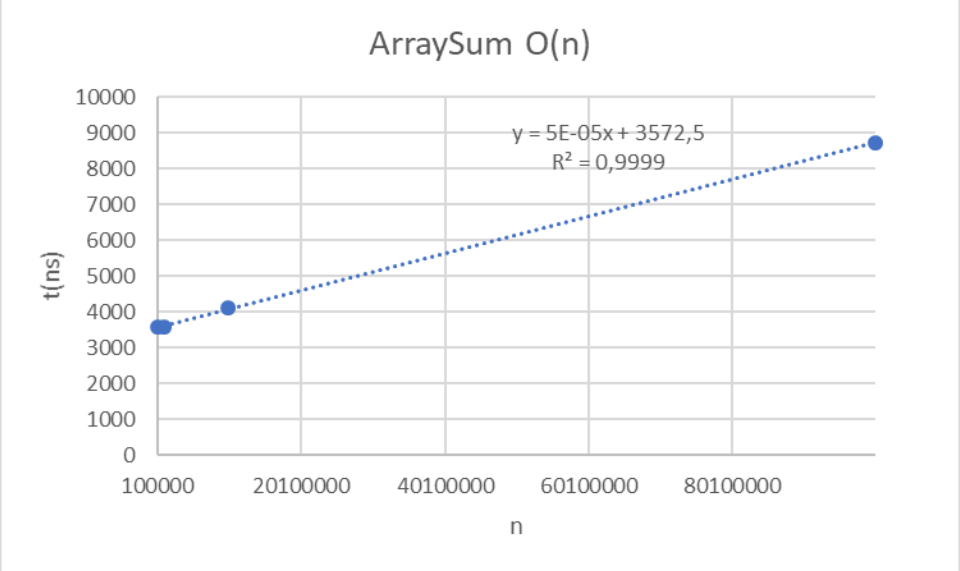
n	100000	1000000	10000000	100000000
ArraySum	3592ns	3593ns	4106ns	8724ns

n	7000	9000	11000	13000	15000
ArrayMax	152074ns	278488ns	226315ns	310819ns	397890ns

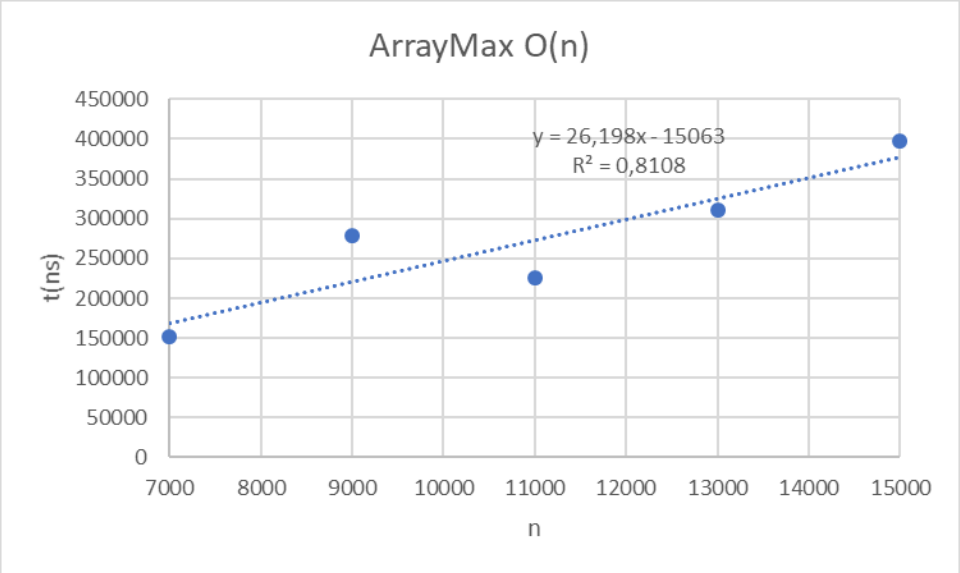
n	6	8	10	12	14
Fibonacci	1520ns	7409ns	9499ns	23178ns	59560ns

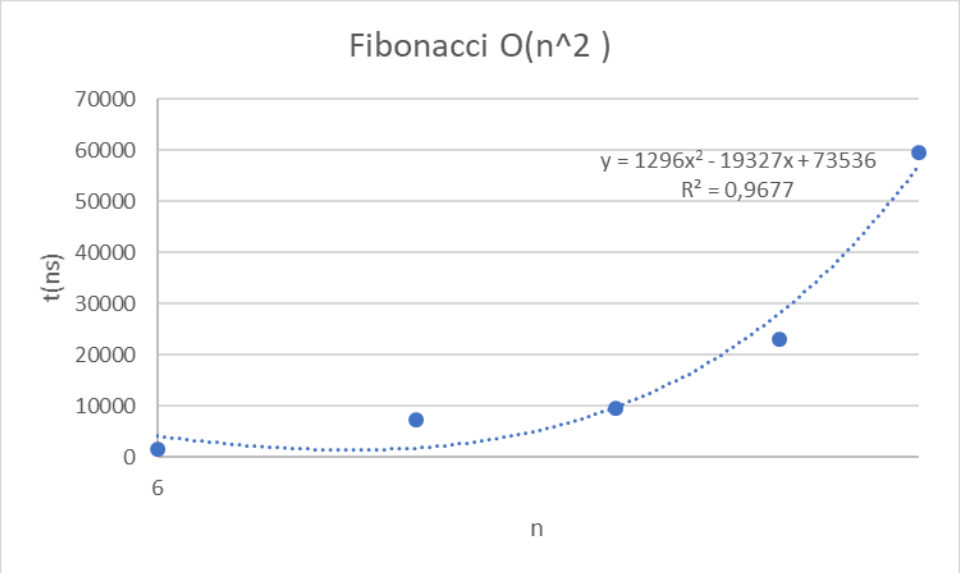
****Acerca de la diferencia de tamaño del problema con que fue medido cada método:**
Para fibonacci se tomaron valores de n pequeños debido a los largos tiempos de ejecución que iba tomando. Sin embargo, para arrayMax, los tamaños bajos de n son debidos a que pronto lanzaba un StackOverFlowError.

2. Gráficas de los tiempos de arraySum, arrayMax y fibonacci recursivos.



DOCENTE MAURICIO TORO BERMÚDEZ
Teléfono: (+57) (4) 261 95 00 Ext. 9473. Oficina: 19 - 627
Correo: mtorobe@eafit.edu.co





3. Conclusiones de la teoría de complejidad (notación O) comparada con los resultados experimentales.

La complejidad se asemeja a los resultados obtenidos, aunque en la práctica diversos factores puedan hacer que varíen los tiempos de ejecución. Por ejemplo, la gráfica de los tiempos de ejecución de arrayMax tiene variaciones que podrían ser consideradas ilógicas.

4. ¿Qué aprendimos del StackOverflowError?

Los parámetros y datos locales se encuentran almacenados en el stack de nuestro computador. Este stack tiene una cantidad de almacenamiento limitada, y existen cierto tipo de errores de programación que causan que el stack intente utilizar más de la memoria disponible. A este error se le conoce como "StackOverflowError"; usualmente se ve por causa de un llamado recursivo que no tiene una correcta condición de parada, o que se extiende demasiado.

5. Valor máximo hallado para Fibonacci y motivo.

El valor máximo que hallamos para Fibonacci corresponde al número 20365011074, cuyo n corresponde a n = 52. Y cancelamos la ejecución al minuto 12 de haber iniciado esta. El principal motivo por el que no continuamos buscando un fibonacci mayor es porque tardaba demasiado e iniciamos el programa con la esperanza de no tener que cancelarlo, sino que este parara por un StackOverflowError, cosa que no ocurrió.

6. ¿Cómo podríamos calcular valores de Fibonacci mucho mayores?

El uso de la recursión puede traer muchas utilidades, pero también puede terminar en un proceso mucho más lento, como se puede ver en el caso de números grandes de fibonacci. El problema resuelto con recursión tiene una complejidad de O(2^n), el cual cuando se toman valores más altos, tiende a complicarse mucho. Para este tipo de problemas, existe una solución más eficaz, llamada programación dinámica. Este tipo de programación reduce la complejidad de fibonacci a O(n), almacenando el resultado de los valores previos para utilizarlos más tarde, en vez de tener que recalcularlos una y otra vez.

7. Recursión 1 vs Recursión 2 (CodingBat)

En cuanto a la complejidad de los ejercicios de CodingBat Recursión 1 y Recursión 2, se puede decir lo siguiente:

- Lo ejercicios de Recursión 1 constaban de un único llamado recursivo, mientras los de Recursión 2 constan de 2 llamados recursivos.
- La complejidad asintótica BigO de los de Recursión 1 es O(n), mientras la complejidad de los de Recursión 2 es O(2^n) lo que es una abismal diferencia en rendimiento con procesamiento recursivo.

- Nos fue notablemente más complejo solucionar los ejercicios de Recursión 2 comparados con los de Recursión 1, debido a que requerían, de cierto modo, otra forma de ver el problema.

4) Simulacro de Parcial

1. start+1, nums, target
2. a
3.
   - n-a, a, b, c
   - res, solucionar(n-b, a, b, c) +1
   - res, solucionar(n-c, a, b, c) + 1
4. e

DOCENTE MAURICIO TORO BERMÚDEZ
Teléfono: (+57) (4) 261 95 00 Ext. 9473. Oficina: 19 - 627
Correo: mtorobe@eafit.edu.co