cuadrático, aunque a primera vista su tiempo de ejecución parece ser lineal.

En el caso de la multiplicación aún podría ser razonable considerarla una operación elemental para operandos suficientemente pequeños. Desafortunadamente, es más fácil producir operandos grandes por multiplicaciones repetidas que por adiciones, por lo tanto es muy importante asegurar que las operaciones aritméticas no se desborden. Aún más, el tiempo requerido para efectuar una adición crece linealmente con respecto al tamaño de los operandos, pero el tiempo requerido para efectuar una multiplicación se cree que crece más rápido que eso.

Un problema similar puede ocurrir cuando analizamos algoritmos que involucran números reales si la precisión requerida aumenta con el tamaño de los ejemplares a resolver. Un ejemplo típico de este fenómeno es cuando se usa la fórmula de de Moivre para calcular los valores de la secuencia de fibonacci. Esta fórmula nos dice que f_n el $n-$ésimo término en la secuencia, es aproximadamente igual a $\phi^n/\sqrt{5}$ donde $\phi = (1+\sqrt{5})/2$ es la *proporción dorada*. La aproximación es tan suficientemente buena, que en principio podemos obtener el valor exacto de f_n tomando simplemente el entero más cercano. No obstante, vimos que se necesitan alrededor de 45496 bits para representar exactamente a f_{65535}. Esto significa que tendríamos que calcular la aproximación con el mismo grado de exactitud que el requerido para obtener al número exacto. La aritmética de punto flotante de precisión simple o doble, usando una o dos palabras de computadora, ciertamente no sería suficientemente exacta. En la mayoría de las situaciones prácticas, no obstante, el uso de aritmética de punto flotante de precisión simple o doble ha sido satisfactoria, a pesar de la inevitable pérdida de precisión. Cuando esto es así, es razonable considerar dichas operaciones como de costo unitario.

Sumarizando, decidir cuando una instrucción tan aparentemente inofensiva como "$j \leftarrow i + j$" puede ser considerada como elemental o no, es algo que siempre requiere de nuestro juicio. En lo que sigue, consideraremos como operaciones elementales a las adiciones, sustracciones, multiplicaciones, divisiones, operaciones módulo, operaciones Booleanas, comparaciones y asignamientos; y por lo tanto que pueden ejecutarse con costo unitario, al menos que explícitamente enunciemos lo contrario.

Ejercicio 15

1. Use la fórmula de de Moivre para f_n y demuestre que f_n es el entero más cercano a $\phi^n/\sqrt{5}$ para todo $n \geq 1$.
2. Sea $g(n)$ el número de maneras de escribir una cadena de n ceros y unos tal que nunca hay dos ceros consecutivos. Por ejemplo, cuando $n = 1$ las cadenas posibles son 0 y 1, así $g(1) = 2$; cuando $n = 2$ las posibles cadenas son 01, 11 y 10, así $g(2) = 3$; cuando $n = 3$ las cadenas posibles son 010, 011, 101, 110 y 111, así $g(3) = 5$. Demuestre que $g(n) = f_{n+2}$.
3. ¿Es razonable, como asunto práctico, considerar a la división como una operación elemental: (a) siempre, (b) algunas veces, (c) nunca? Justifique su respuesta. Si lo considera necesario, puede tratar separadamente la división de enteros y la división de números reales.
4. En la subsección 2.4, vimos que el teorema de Wilson podría ser usado para probar la primalidad de cualquier número en tiempo constante, si los factoriales y las pruebas de divisibilidad de enteros se contarán de costo unitario, sin importar el tamaño de los números involucrados. Claramente esto no sería razonable. Use el teorema de Wilson junto con el teorema binomial de Newton para diseñar un algoritmo capaz de decidir en un tiempo en el orden de $\log n$ cuando o no un entero n es primo, considerando que son contadas con costo unitario las adiciones, multiplicaciones, y pruebas por divisibilidad de enteros, pero no así los factoriales y exponenciales. Lo más importante de este ejercicio no es proporcionar un algoritmo útil, sino demostrar que no es razonable, en general, considerar a las multiplicaciones como operaciones elementales.

2.5. La importancia de la eficiencia

Ya que las computadoras se hacen cada vez más y más rápidas, pudiera parecer que no vale la pena gastar nuestro tiempo tratando de diseñar algoritmos más eficientes. ¿No sería más fácil esperar a la siguiente generación de computadoras? Los principios establecidos en las subsecciones anteriores muestra que esto no es verdad. Suponga, para ilustrar el argumento, que para solucionar un problema particular tiene disponible un algoritmo exponencial y una computadora que puede correr este algoritmo sobre ejemplares de tamaño n en $10^{-4} \times 2^n$ segundos. Así el programa puede resolver ejemplares de tamaño 10 en $10^{-4} \times 2^{10}$ segundos o alrededor de un décimo de segundo. Resolver un ejemplar de tamaño 20 consumiría alrededor de mil veces más o aproximadamente dos minutos. Para resolver un ejemplar de tamaño 30 tomaría mil veces más, por lo cual no serían suficientes los cálculos de un día entero. Suponiendo que se pudiera ejecutar una computadora sin interrupción y sin errores por una año completo, sólo se podría resolver un ejemplar de tamaño 38.

Suponga que necesita resolver ejemplares más grandes que esto y que tiene el dinero suficiente para comprar una computadora nueva un ciento de veces más rápida que la primera. Con el mismo algoritmo ahora se puede resolver un ejemplar de tamaño n en sólo $10^{-6} \times 2^n$ segundos. Podría sentir que ha desperdiciado su dinero cuando se dé cuenta que, si corre la nueva máquina por un año entero, ni siquiera podrá resolver un ejemplar de tamaño 45. En general, si antes se podía resolver ejemplares de tamaño n en un tiempo dado, con la nueva computadora en el mejor de los casos resolverá ejemplares de tamaño $n + \log 100$ o alrededor de $n + 7$, en el mismo tiempo.

Suponga que mejor decide invertir en algoritmia y que habiendo gastado la misma cantidad de dinero, ha conseguido un algoritmo cúbico para solucionar el problema. Imagine por ejemplo, que usando la primera máquina con el nuevo algoritmo se pude resolver un ejemplar de tamaño n en $10^{-2} \times n^3$ segundos. Así, para resolver un ejemplar de tamaño 10 consumirá 10 segundos y un ejemplar de tamaño 20 requerirá entre uno y dos minutos. Pero ahora un ejemplar de tamaño 30 se puede resolver en cuatro y medio minutos y en un día se pueden resolver ejemplares de tamaño mayor a 200; con los cálculos de un año se pueden resolver ejemplares de tamaño cercano a 1500.

El nuevo algoritmo no sólo ofrece una mejora mucho mayor que la compra del nuevo hardware, suponiendo que puede afrontar ambos gastos, haría mucho más productiva la compra del nuevo hardware. Si puede hacer uso del nuevo algoritmo y de la máquina 100 veces más rápida, entonces podrá resolver ejemplares cuatro o cinco veces más grandes que con sólo el nuevo algoritmo, en la misma cantidad de tiempo, el factor exacto es $\sqrt[3]{100}$. Compare esto a la situación con el viejo algoritmo, donde se le podía *sumar* 7 al tamaño del ejemplar, aqui se puede *multiplicar* el tamaño de los ejemplares por cuatro o cinco. No obstante, el nuevo algoritmo no debe ser usado descuidadamente sobre todos los ejemplares del problema, en particular sobre los muy pequeños. Vimos que sobre la primera máquina el algoritmo nuevo consume 10 segundos para resolver un ejemplar de tamaño 10, lo cual es un ciento de veces más lento que el algoritmo exponencial. El nuevo algoritmo es más rápido sólo para ejemplares de tamaño 20 o mayores, ver Figura 1. Naturalmente, es posible combinar ambos algoritmos en un tercero que verifique el tamaño del ejemplar que hay que resolver, antes de decidir que método usar.

Ejercicio 16

1. Algunas veces se afirma que el hardware de hoy en día es tan barato y la mano de obra tan cara, que no vale la pena gastar el tiempo de un programador para *"rasurarle"* unos pocos segundos al tiempo de ejecución de un programa. ¿Esto significa que la algoritmia está destinada a ser únicamente una ocupación teórica con fines de formalidad, sin aplicación práctica? Justifique su respuesta.