2. Usando la técnica llamada "memoría virtual", es posible liberar al programador de la mayoría de las consideraciones sobre el tamaño real de almacenamiento disponible en la máquina. ¿Significa esto que la cantidad de almacenamiento usado por un algoritmo nunca es de interés en la práctica? Justifique su respuesta.

3. Suponga que mide el desempeño de un programa, quizas usando alguna clase de traza en tiempo de ejecución, luego optimiza las partes muy usadas del código. No obstante se asegura de no cambiar el algoritmo subyacente. ¿Qué esperaría obtener: (a) una ganancia en eficiencia mediante un factor constante, o (b) una ganancia en eficiencia que es proporcionalmente mayor conforme el tamaño del ejemplar aumenta? Justifique su respuesta.

4. Un algoritmo de ordenamiento consume 1 segundo para ordenar 1000 artículos en su computadora local.¿Cuánto consumiría para ordenar 10000 artículos si (a) cree que el algoritmo consume tiempo aproximadamente proporcional a n^2 y (b) cree que el algoritmo consume tiempo aproximadamente proporcional a $n \log n$?

5. Dos algoritmos consumen respectivamente n^2 días y n^3 segundos para resolver un ejemplar de tamaño n. Muestre que sólo sobre ejemplares que requieren más de 20 millones de años en ser resueltos, el algoritmo cuadrático supera al algoritmo cúbico.

6. Dos algoritmos consumen respectivamente n^2 días y 2^n segundos para resolver un ejemplar de tamaño n. ¿Cuál es el tamaño del ejemplar más pequeño para el cual el primer algoritmo supera al segundo? ¿Aproximadamente cuánto tiempo consumiría dicho ejemplar en ser resuelto?

7. Cierto algoritmo consume $10^{-4} \times 2^n$ segundos para resolver un ejemplar de tamaño n. Muestre que en un año sólo podría resolver un ejemplar de tamaño 38. ¿Qué tamaño de ejemplar podría ser resulto en un año con una máquina un ciento de veces más rápida? Un segundo algoritmo consume $10^{-2} \times n^3$ segundos para resolver ejemplares de tamaño n. ¿Qué tamaño de ejemplar puede resolver en un año? ¿Qué tamaño de ejemplar puede resolver en un año con una máquina un ciento de veces más rápida? Muestre que el segundo algoritmo es más lento que el primero para ejemplares de tamaño menor que 20.

2.6. Un algoritmo lineal para ordenamiento

Se puede demostrar que ningún algoritmo de ordenamiento, que trabaje comparando los elementos a ser ordenados puede ser más rápido que el orden de $n \log n$. No obstante, se pueden encontrar otros algoritmos de ordenamiento más eficientes para casos *muy especiales*. Suponga por ejemplo que los elementos a ordenar son enteros que se sabe están en el rango de 1 a 10000. Entonces el siguiente algoritmo puede usarse.

```
procedure CASILLAS(T[1...n])
 1  array U[1...10000]
 2  for k ← 1 to 10000 do
 3     U[k] ← 0
 4  for i ← 1 to n do
 5     k ← T[i]
 6     U[k] ← U[k] + 1
 7  i ← 0
 8  for k ← 1 to 10000 do
 9     while U[k] ≠ 0 do
10        i ← i + 1
11        T[i] ← k
12        U[k] ← U[k] − 1
```

Aquí U es un arreglo de "casillas" para los elementos que se van a ordenar. Debe haber una casilla para cada elemento posible que se pueda encontrar en T. El primer ciclo limpia las casillas, el segundo pone cada elemento de T en el lugar adecuado y el tercero los saca de U para volverlos a poner en T en orden ascendente. Es fácil mostrar que este algoritmo y sus variantes consumen un tiempo en el orden de n. (La constante multiplicativa oculta depende del número que acota los valores de los elementos a ordenar, en este caso 10000.) Cuando este algoritmo se aplica vence a cualquier otro algoritmo que trabaje con comparaciones, por otro lado el requerimiento de que debe poder usar una casilla por cada elemento posible, implica que es aplicable mucho menos veces en comparación a los otros algortimos de ordenamiento.

Ejercicio 17

1. Se le pide ordenar un archivo que contiene enteros entre 0 y 999999. Pero no puede usar un millón de casillas, así que decide usar casillas numeradas desde 0 hasta 999. Empieza a ordenar poniendo cada entero en la casilla que le corresponde de acuerdo a sus primeros tres dígitos. Después usa ordenamiento por inserción mil veces, para ordenar separadamente los contenidos de cada casilla y finalmente vacía las casillas para obtener una secuencia totalmente ordenada. ¿Esperaría que esta técnica sea más rápida, más lenta o igual, con respecto a usar ordenamiento por inserción sobre toda la secuencia (a) en promedio y (b) en el caso peor? Justifique sus respuestas.

2. Muestre que ordenamiento por casillas consume un tiempo en el orden de n para ordenar n elementos que están dentro de las cotas.

2.7. Especificación completa de un algoritmo

Al inicio dijimos que la ejecución de un algoritmo normalmente no debe involucrar decisiones subjetivas, ni debe requerir el uso de la intuición o creatividad; posteriormente dijimos que casi siempre debemos estar contentos si demostramos que nuestros algoritmos son correctos en abstracto, ignorando las limitaciones prácticas; después, propusimos considerar que la mayoría de las operaciones aritméticas son elementales al menos que explícitamente aclaremos lo contrario.

Todo esto está muy bien, pero ¿qué debemos hacer si consideraciones prácticas nos obligan a abandonar esta posición conveniente y tenemos que tomar en cuenta las limitaciones de las máquinas disponibles? Por ejemplo, cualquier algoritmo que calcule el valor exacto de f_{100} será forzado a considerar que ciertas operaciones aritméticas, ciertamente la adición y también posiblemente la multiplicación no son operaciones elementales (recuerde que f_{100} es un número con 21 dígitos decimales). Más probablemente esto se tomará en cuenta para usar un paquete de programas que permitan operaciones aritméticas sobre enteros muy largos. Si no especificamos exactamente de qué manera el paquete debe implementar aritmética de precisión multiple, entonces la elección de qué método usar podría ser considerada una decisión subjetiva y el algoritmo propuesto estaría especificado incompletamente. ¿Esto importa?

La respuesta es que en ciertos casos sí importa. Posteriormente veremos algoritmos cuyo desempeño en verdad depende del método que se use para multiplicar enteros largos. Para tales algoritmos (y formalmente hablando para *cualquier* algoritmo) no es suficiente escribir simplemente una instrucción como $x \leftarrow y \times z$, dejándole al lector elegir cualquier técnica que tenga a la mano para implementar esta multiplicación. Para especificar completamente al algoritmo, también debemos especificar como serán implementadas las operaciones aritméticas necesarias.

No obstante, para hacernos la vida más fácil, continuaremos usando la palabra algoritmo para ciertas descripciones incompletas de este tipo. Los detalles serán completados cuando el análisis así lo requiera.