## CAPITULO 2: GETTING STARTED

### 2.3 Diseño de algoritmos

Puede elegir entre una amplia gama de técnicas de diseño de algoritmos. La ordenación por inserción utiliza el método incremental: para cada elemento A[i], se inserta en su lugar correspondiente en el subarreglo A[1:i], tras haber ordenado previamente dicho subarreglo A[1:i-1].

Esta sección examina otro método de diseño, conocido como "divide y vencerás", que exploraremos con más detalle en el Capítulo 4. Utilizaremos este método para diseñar un algoritmo de ordenación cuyo tiempo de ejecución en el peor de los casos es mucho menor que el de la ordenación por inserción.

Una ventaja de usar un algoritmo que sigue el método de divide y vencerás es que analizar su tiempo de ejecución suele ser sencillo, utilizando técnicas que exploraremos en el Capítulo 4.

#### 2.3.1 El método divide y vencerás

Muchos algoritmos útiles tienen una estructura recursiva: para resolver un problema dado, recurren (se llaman a sí mismos) una o más veces para abordar subproblemas estrechamente relacionados.

Estos algoritmos suelen seguir el método de divide y vencerás: dividen el problema en varios subproblemas similares al original, pero de menor tamaño, los resuelven recursivamente y luego combinan estas soluciones para crear una solución al problema original.

En el método de divide y vencerás, si el problema es lo suficientemente pequeño (el caso base), se resuelve directamente sin recurrir. En caso contrario (el caso recursivo), se realizan tres pasos característicos:

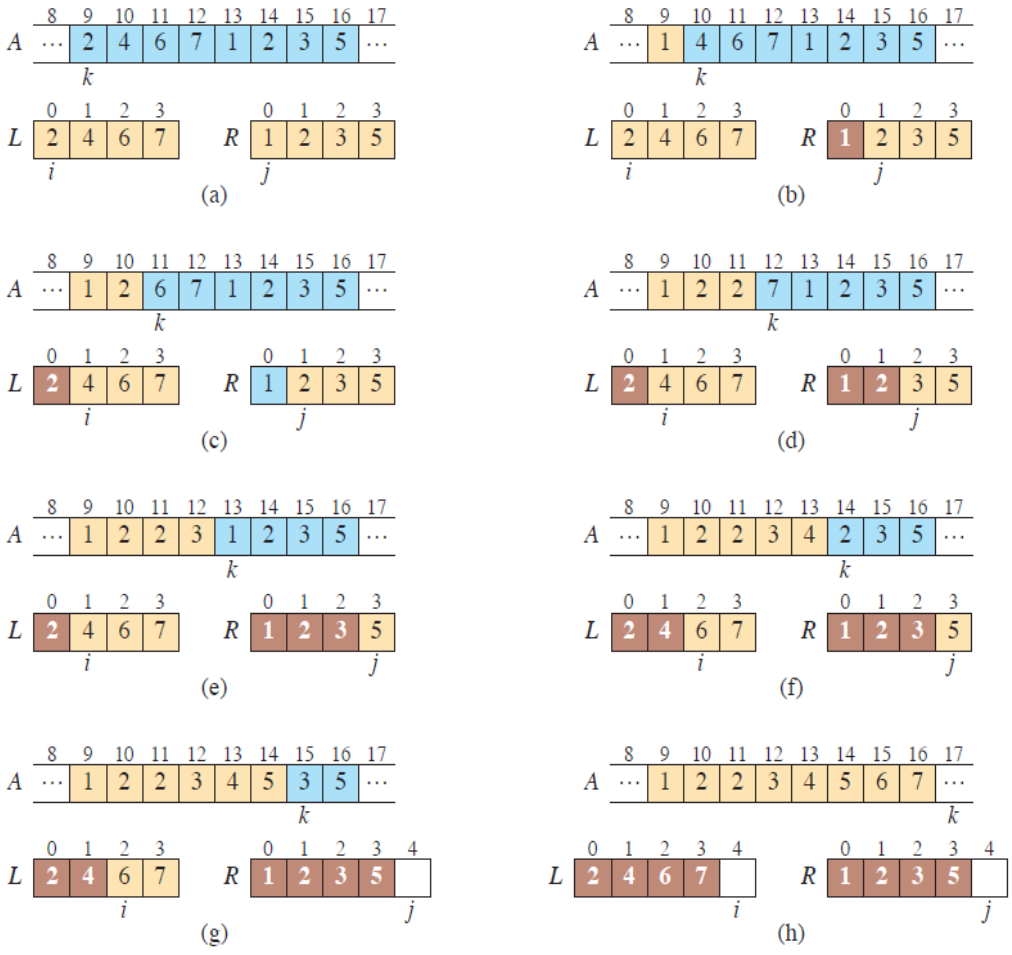
- **Divida** el problema en uno o más subproblemas, que son instancias más pequeñas del mismo problema.
- **Resuelva** los subproblemas recursivamente.
- **Combine** las soluciones de los subproblemas para formar una solución al problema original.

El algoritmo de ordenamiento por fusión sigue de cerca el método de divide y vencerás. En cada paso, ordena un subarreglo A[p:r], comenzando con el arreglo completo A[1:n] y recorriendo hacia abajo hasta subarreglos cada vez más pequeños. Así es como funciona el ordenamiento por fusión:

- **Divida** el subarreglo A[p:r] que se va a ordenar en dos subarreglos adyacentes, cada uno de la mitad del tamaño. Para ello, calcule el punto medio q de A[p:r] (mediando p y r) y divida A[p:r] en los subarreglos A[p:q] y A[q+1:r].
- **Resuelva** ordenando recursivamente cada uno de los dos subarreglos A[p:q] y A[q+1:r] mediante ordenación por fusión.
- **Combine** fusionando los dos subarreglos ordenados A[p:q] y A[q+1:r] en A[p:r], lo que genera la solución ordenada.

La recursión toca fondo al alcanzar el caso base: cuando el subarreglo A[p:r] a ordenar tiene solo un elemento, es decir, cuando p es igual a r. Como se indicó en el argumento de inicialización del invariante de bucle de INSERTION-SORT, un subarreglo con un solo elemento siempre se ordena.

Para entender cómo funciona el procedimiento MERGE, volvamos a nuestro tema de juego de cartas. Supongamos que tenemos dos montones de cartas boca arriba sobre una mesa. Cada montón está ordenado, con las cartas de menor valor arriba. Deseas combinar los dos montones en un único montón de salida ordenado, que estará boca abajo sobre la mesa. El paso básico consiste en elegir la carta más pequeña de las dos que están sobre los montones boca arriba, retirarla de su montón (lo que expone una nueva carta superior) y colocarla boca abajo sobre el montón de salida. Repite este paso hasta que un montón de entrada esté vacío; en ese momento, puedes simplemente tomar el montón de entrada restante y volcarlo sobre todo el montón, colocándolo boca abajo sobre el montón de salida.

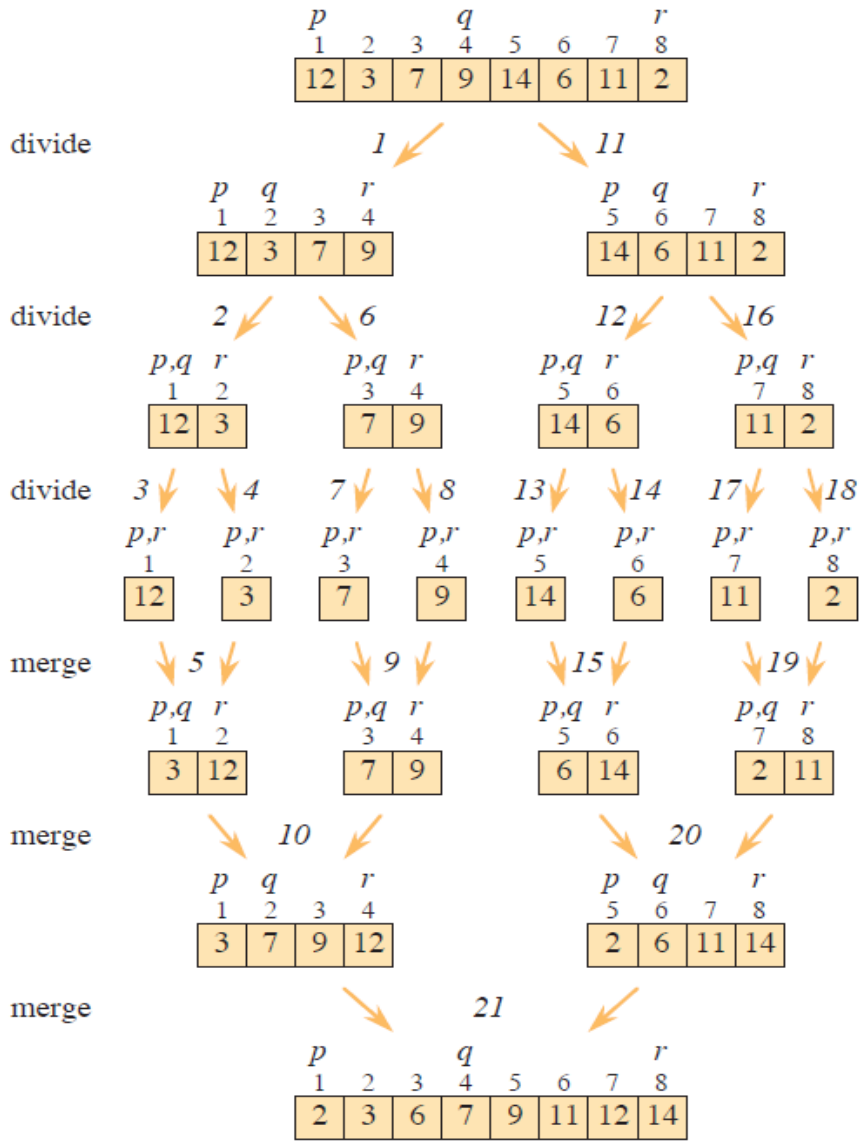



## 2.3.2 Análisis de algoritmos de divide y vencerás

Cuando un algoritmo contiene una llamada recursiva, a menudo se puede describir su tiempo de ejecución mediante una ecuación de recurrencia o recurrencia, que describe el tiempo total de ejecución en un problema de tamaño n en términos del tiempo de ejecución del mismo algoritmo con entradas más pequeñas. Posteriormente, se pueden

utilizar herramientas matemáticas para resolver la recurrencia y establecer límites para el rendimiento del algoritmo.

Una recurrencia para el tiempo de ejecución de un algoritmo de divide y vencerás se desprende de los tres pasos del método básico. Al igual que en el caso de la ordenación por inserción, sea T(n) el tiempo de ejecución en el peor caso posible para un problema de tamaño n. Si el tamaño es suficientemente pequeño, digamos $n < n_0$ para una constante $n_0 > 0$, la solución directa requiere un tiempo constante, que se escribe como $\Theta(1)$. Supongamos que la división del problema genera subproblemas a, cada uno con un tamaño n/b, es decir, 1/b del tamaño del original. Para la ordenación por fusión, tanto a como b son 2, pero veremos otros algoritmos de divide y vencerás en los que $a \neq b$. Se necesita un tiempo T(n/b) para resolver un subproblema de tamaño n/b, por lo que se necesita un tiempo T(n/b) para resolver todos los a. Si se necesitan D(n) tiempos para dividir el problema en subproblemas y C(n) tiempos para combinar las soluciones de los subproblemas en la solución del problema original, obtenemos la recurrencia.



**Análisis de merge sort**

Aquí se explica cómo establecer la recurrencia para T(n), el tiempo de ejecución en el peor caso de merge sort en n números.

- **Dividir:** El paso de división solo calcula el medio del subarreglo, lo que toma un tiempo constante. Así, $D(n) = \Theta(1)$.
- **Conquistar:** Resolver recursivamente dos subproblemas, cada uno de tamaño n/2, contribuye $2T(n/2)$ al tiempo de ejecución (ignorando los pisos y techos, como se discutió).
- **Combinar:** Dado que el procedimiento MERGE en un subarreglo de n elementos toma $\Theta(n)$ tiempo, tenemos $C(n) = \Theta(n)$.

Cuando sumamos las funciones D(n) y C(n) para el análisis de merge sort, estamos sumando una función que es $\Theta(n)$ y una función que es $\Theta(1)$. Esta suma es una función lineal de n. Es decir, es aproximadamente proporcional a n cuando n es grande, y por lo tanto, los tiempos de división y combinación de merge sort juntos son $\Theta(n)$. Sumando $\Theta(n)$ al término $2T(n/2)$ del paso de conquista, obtenemos la recurrencia para el tiempo de ejecución en el peor caso T(n) de merge sort:

$$T(n) = 2T(n/2) + \Theta(n). \quad (2.3)$$

El Capítulo 4 presenta el "teorema maestro", que muestra que $T(n) = \Theta(n \lg n)$. Comparado con insertion sort, cuyo tiempo de ejecución en el peor caso es $\Theta(n^2)$, merge sort intercambia un factor de n por un factor de lg n. Debido a que la función logaritmo crece más lentamente que cualquier función lineal, este es un buen intercambio. Para entradas lo suficientemente grandes, merge sort, con su tiempo de ejecución en el peor caso de $\Theta(n \lg n)$, supera a insertion sort, cuyo tiempo de ejecución en el peor caso es $\Theta(n^2)$.