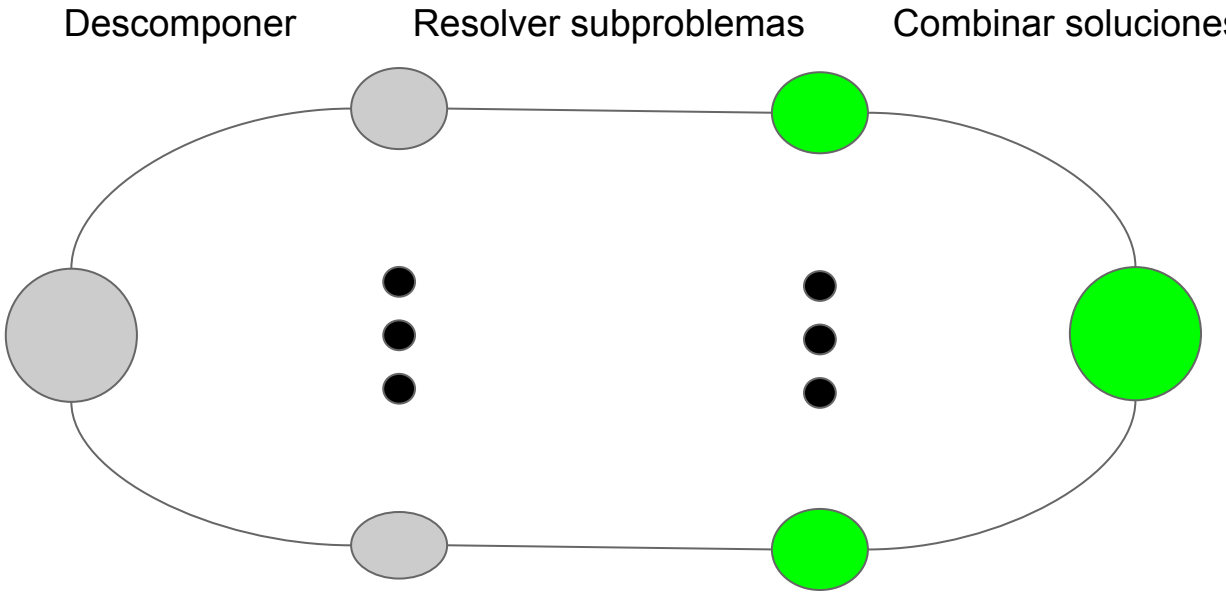
Estructuras de Datos

Divide and Conquer

Algoritmo Divide and Conquer

La idea de los algoritmos de Dividir y Conquistar se basa en **descomponer** un problema en instancias más chicas del mismo problema, resolverlas de forma independiente, y luego **combinar** todas las soluciones parciales para formar una solución al problema principal.



Ejemplo - Ordenación por Merge (Mergesort)

El algoritmo de ordenación *Mergesort* procede de la siguiente forma:

- Si la longitud de la lista a ordenar es 0 ó 1, la lista está ordenada y la retorna.
- Si la longitud de la lista a ordenar es mayor a 1, se **parte** la lista a aproximadamente la mitad, generando dos sublistas desordenadas L_1, L_2. Ambas sublistas **se ordenan recursivamente utilizando** Mergesort nuevamente, y así obtenemos dos sublistas ordenadas SL_1, SL_2. Finalmente, las dos sublistas ordenadas, SL_1 y SL_2, se **combinan** produciendo el resultado de ordenar la lista original y lo devolvemos.

Ejemplo - Ordenación por Merge (Mergesort)

Analizando el algoritmo presentado en la slide anterior observamos que para listas con más de un elemento lo que se hace es:

- Partir la lista en dos mitades aproximadamente iguales
- Ordenar recursivamente ambas listas
- A partir de las dos listas ordenadas obtener el resultado

La complejidad del algoritmo dependerá entonces de la complejidad de partir una lista, y de unir los resultados parciales. Las llamadas recursivas volverán a realizar el mismo procedimiento.

Ejemplo - Ordenación por Merge (Mergesort)

- Dada una lista L, de longitud par n, dividir la lista en dos sublistas de longitud n/2 involucra la creación e inicialización de dos nuevas listas, podemos asumir entonces que la complejidad es O(n). Eventualmente se puede optimizar y realizar el ordenamiento dentro del mismo arreglo aunque como veremos no mejorará drásticamente la ejecución del algoritmo.
- Unir dos listas ordenadas de longitud n, nos lleva a crear y seleccionar 2*n elementos manteniendo el orden de los mismo. Por lo que la complejidad de unir las dos listas manteniendo el orden será en el caso general de O(2*n).

Es decir, los procedimientos de unir y dividir una lista tienen una complejidad de O(n) cada uno, y si realizamos ambos procedimientos tenemos 2*O(n) ≅ O(n).

Ejemplo - Ordenación por Merge (Mergesort)

Para calcular la complejidad de Mergesort asumimos que recibe como argumento una lista de longitud $m = 2^n$ con n natural. Esto nos permite a todo momento realizar la división de forma regular garantizando que siempre podremos dividir la lista como argumento en dos.

Dado que lo que dicta la recursión es la longitud de la lista podemos analizar su complejidad siguiendo la siguiente fórmula:

$$TMerge(m) = TMerge(m/2) + TMerge(m/2) + O(m)$$

Es decir, el costo de una llamada a Merge es el costo de llamar recursivamente a Merge en cada mitad más el costo de dividir y unir, ya que ambos utilizan $O(m)$ podemos unificarlo en uno sólo.

Ejemplo - Ordenación por Merge (Mergesort)

Si analizamos la forma que nos provee la recursión tenemos

$$TMerge(n) = 2*TMerge(n/2) + O(n) = 2*(2*TMerge(n/4) + O(n/2)) + O(n)$$

$$= 4*TMerge(n/4) + 2*O(n/2) + O(n)$$

$$= 8*TMerge(n/8) + 4*O(n/4) + 2 * O(n/2) + O(n)$$

…

Dado que asumimos que n era una potencia de dos, se puede subdividir la lista una cantidad total de log(n) veces.

$$TMerge(n) \cong log(n)*O(n) \cong O(n*log(n))$$

Divide and Conquer

Los algoritmos de Divide and Conquer se basan en la división de trabajo, y el cómputo de los resultados parciales para devolver una solución general del problema. Por lo que suelen generar una fórmula de recursión de la forma:

$T(n) = b * T(n/a) + f(n)$

Donde:

- b representa la cantidad de subproblemas generados
- a representa la fracción en la que se divide el problema general
- f(n) que representa el trabajo de dividir el problema y unir las soluciones parciales.

Divide and Conquer - Teorema Maestro

Por suerte contamos con un teorema, llamado Teorema Maestro, que resuelve los casos más comunes.

Dada una recursión $T(n) = b * T(n/a) + f(n)$ donde $f(n) = c*n^s$.

- Si $b < a^s$, entonces $T(n) \cong O(n^s)$
- Si $b = a^s$, entonces $T(n) \cong O(n^s * \log_a n)$
- Si $b > a^s$, entonces $T(n) \cong O(n^{(\log_a b)})$

Si revisamos la recurrencia TMerge, tenemos los valores $a = b = 2$ y $s=1$. Es decir, deberíamos aplicar el segundo caso con lo que quedaría $TMerge(n) \cong O(n^1 * \log(n))$ como habíamos calculado originalmente.