The divide-and conquer approach:

- many algos are recursive in nature ⇒ they call themselves recursively one or more times to deal with closely related subproblems
- These algos follow a divide-and-conquer approach.
- The D&C paradigm involves 3 steps at each level of the recursion:

(I) Divide the problem into a number of subproblems (that are similar to the original problem but smaller in size, also independent to each other)

(II) Conquer the subproblems by solving them recursively (if the subproblem sizes are small enough, solve the subproblems in a straightforward manner)

(III) Combine the solutions to the subproblems into the solution for the original problem.

- One advantage of D&C algos is that their running times are often easily determined using techniques of recurrences.

eg The merge sort algorithm follows the D&C paradigm.

Divide Divide the n-element input sequence into two subsequences of $n/2$ elements each

Conquer Sort the two subsequences recursively using merge sort.

Combine Merge the two sorted subsequences to produce the sorted answer.

- Here, the recursion bottoms out when the sequence to be sorted has length 1, since every sequence of length 1 is already in sorted order.
- The key operation of the merge sort algorithm is the merging of two sorted sequences in the combine step.

Merge Procedure (A, p, q, r)

This procedure merges two sorted subarrays A[p..q] and A[q+1..r] to form a single sorted subarray that replaces the current subarray A[p..r]

Here A : an array

p, q, r : indices numbering the elements of the array such that $p \le q < r$

* it should be noted that the subarrays A[p..q] and A[q+1..r] both can have only single element when

$p = q$ and

$q+1 = r$

Basic Idea

(Analogy)

- we have two piles of cards face up on a table, each pile is sorted with smallest cards on top
- goal is to merge the two piles into a single sorted output pile.
- the basic step is choosing the smaller of the two cards on top of the face-up piles, removing it from its pile and placing this card onto the output pile.
- repeat this step until one input pile is empty, and then place remaining cards of another pile onto the output pile
- in order to avoid checking whether either pile is empty, we will place on the bottom of each pile a sentinel card (∞)
  - which contains a special value
  - it can't be smaller than other cards.

Merge (A, p, q, r)

```
1.   n1 ← q-p+1
2.   n2 ← r-(q+1)+1
3.   create arrays L[1..n1+1] and R[1..n2+1]
4.   for i ← 1 to n1
5.       L[i] ← A[p+i-1]
6.   for j ← 2 to n2
7.       R[j] ← A[q+1+j-1]
8.   L[n1+1] ← ∞
9.   R[n2+1] ← ∞
10.  i ← 1
11.  j ← 1
12   for k ← p to r
13       if L[i] ≤ R[j]
14           A[k] ← L[i]
15           i ← i+1
16       else A[k] ← R[j]
17           j ← j+1
```

working line 1 computes the length n_1 of subarray $A[p..q]$

line 2 " n_2 " $A[q+1..r]$

line 3 creates arrays L & R of lengths (n_1+1) & (n_2+1)

line 4 & 5 copies the subarray $A[p..q]$ into $L[1..n_1]$

line 6 & 7 " $A[q+1..r]$ into $R[1..n_2]$

8 & 9 put sentinels at the end of L & R

10 to 17 perform the basic $r-p+1$ basic steps by maintaining

L.I The subarray $A[p..k-1]$ contains the $k-p$ smallest elements of $L[1..n_1+1]$ and $R[1..n_2+1]$ in sorted order. Moreover, $L[i]$ and $R[j]$ are the smallest elements of their arrays that have not copied back into A.

Initialization Prior to the 1st iteration of the loop, we have $k=p$, so the subarray $A[p..p-1]$ is empty

$\Rightarrow$ this empty subarray contains $k-p=0$ smallest elements of L & R.

Also since $i=j=1$, both $L[i]$ and $R[j]$ are the smallest elements of their arrays that have not been copied back into A.

Maintenance Each iteration maintains L.I.

Let suppose $L[i] \leq R[j]$, then $L[i]$ is the smallest element not yet copied back into A. The subarray $A[p..k-1]$ contains the $k-p$ smallest elements and after line 14 copies $L[i]$ into $A[k]$, then the subarray $A[p..k]$ will contain the $k-p+1$ smallest elements. Incrementing k & i reestablishes the L.I. for next iteration.

if $L[i] > R[j]$ then lines 16 & 17 perform the appropriate action to maintain the L.I.

Termination At termination, $k = r+1$

the subarray $A[p..k-1]$ which is $A[p..r]$ contains $k-p = r-p+1$ smallest elements of L and R in sorted order.

Running time of Merge (A, p, q, r) procedure

no. of elements to be merged(n) : $r-p+1$

so running time is expressed in terms of 'n'.

Lines		
	1-3	constant time
	4-5	$\Theta(n_1)$
	6-7	$\Theta(n_2)$
	8-11	constant time
	12-17	$\Theta(r-p+1) = \Theta(n)$

$\Theta(n_1) + \Theta(n_2) + \Theta(n)$

$\Theta(n_1+n_2) + \Theta(n)$

$\Theta(n) + \Theta(n)$

$\Theta(n)$

$n_1 = q-p+1$

$n_2 = r-(q+1)+1$

$= r-q$

$n_1+n_2 = r-p+1$

$= n$

Merge sort

- The procedure sorts the elements in the subarray $A[p..r]$
- If $p \geq r$ the subarray has atmost one element (or empty) and is therefore already sorted.
- Otherwise, partition the subarray $A[p..r]$ into two subarrays by simply computing an index q such that

$A[p..q]$: $\lceil n/2 \rceil$ elements and

$A[q+1..r]$: $\lfloor n/2 \rfloor$ elements

$$q = \lfloor (p+r)/2 \rfloor$$

Eg $A[9..15]$

$p = 9$

$r = 15$

total elements $= 15 - 9 + 1 = 7$

$q = \lfloor (9+15)/2 \rfloor = 12$

$A[9..12]$

$A[13..15]$

Merge-sort (A, p, r)

1. if $p < r$ ▷ check for base case
2. $q \leftarrow \lfloor (p+r)/2 \rfloor$ ▷ Divide
3. Merge-sort (A, p, q) ▷ Conquer
4. Merge-sort $(A, q+1, r)$ ▷ Conquer
5. Merge (A, p, q, r) ▷ Combine

Initial call : Merge-sort $(A, 1, n)$