Analysis of Divide-and-Conquer Algorithms

<u>The divide-and-conquer paradigm (Ch.2)</u>

- **divide** the problem into a number of subproblems
- **conquer** the subproblems by solving them
- **combine** the subproblem solutions to get the solution to the problem

add all these steps at the first level to get recurrence relation for T(n)

Example: Merge-Sort: an optimal sorting algorithm

- **divide** the n-element input sequence to be sorted into two n/2-element subsequences.
- **conquer** the subproblems recursively using merge sort.
- **combine** the resulting two sorted n/2-element sequences by merging.

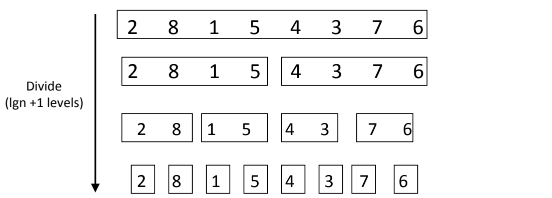
```
Merge-Sort(A,p,r)
if p < r then
   q = ⌊(p+r)/2⌋
   Merge-Sort(A,p,q)
   Merge-Sort(A,q+1,r)
   Merge(A,p,q,r)

Initial call:
Merge-sort(A,1,length(A))
```

The Merge subroutine takes linear time to merge *n* elements that are divided into two *sorted* arrays of *n/2* elements each.

```
Merge(A,p,q,r)
n1 = q-p+1; n2 = r-q;
Create arrays
 L[1...n1+1] and
 R[1...n2+1]
for i = 1 to n1
   L[i] = A[p+i-1]
for i = 1 to n2
   R[i]= A[q+i]
L[n1+1] = R[n2+1] = ∞
i = j = 1
for k = p to r
  if L[i] ≤ R[j]
      A[k] = L[i]
      i = i+1
  else A[k] = R[j]
      j = j+1
```

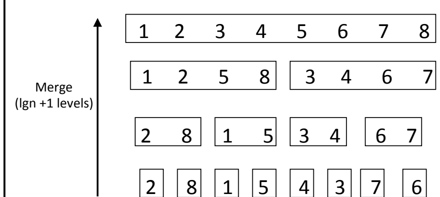
Analyzing Merge-Sort



Why are there lgn + 1 levels? *Because lgn + 1 is the number of steps it takes to divide n by 2 until the size of the result is <= 1*

How long does it take to find the midpoint of an array? $D(n) = \theta(1)$

Analyzing Merge-Sort



$C(n) = \theta(n)$

$$T(n) = \begin{cases} \theta(1) & \text{if } n = 1 \\ 2T(n/2) + \theta(n) & \text{otherwise} \end{cases}$$

Recurrence for worst-case running time for Merge-Sort

Analyzing Merge-Sort

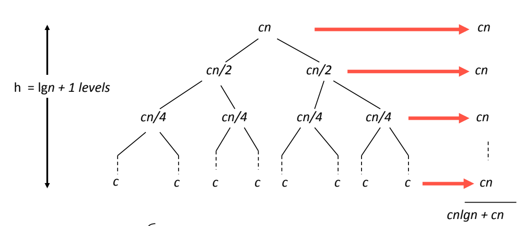
$$T(n) = \begin{cases} \theta(1) & \text{if } n = 1 \\ 2T(n/2) + \theta(n) & \text{otherwise} \end{cases}$$

Recurrence for worst-case running time for Merge-Sort

$$aT(n/b) + D(n) + C(n)$$

- a = 2 (two subproblems)
- n/b = n/2 (each subproblem has size approx $n/2$)
- $D(n)$ = $\Theta(1)$ (just compute midpoint of array)
- $C(n)$ = $\Theta(n)$ (merging can be done by scanning sorted subarrays)

Recursion Tree for Merge-Sort



$$T(n) = \begin{cases} c & \text{if } n = 1 \\ 2T(n/2) + cn & \text{otherwise} \end{cases}$$

Recurrence tree for Merge-Sort