Notes, 5(a)

ECE 606

Design Strategy II: Divide-n-Conquer

Divide the problem into a number of subproblems.

Conquer the subproblems by solving them recursively. If the subproblem sizes are small enough, however, just solve the subproblems in a straightforward manner.

Combine the solutions to the subproblems into the solution for the original problem.

Typically, subproblem is of some non-constant size in size of original problem.

Example: binary-search

- Divide: given problem of size n, subproblem is of size $n/2$.
- Combine: simply return same `true` or `false` result we get from subproblem.

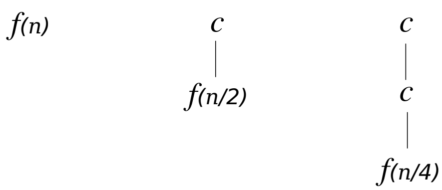
Recurrence for time-efficiency, $f(n)$:

$$f(n) = \begin{cases} f(n/2) + \Theta(1) & \text{if } n > 1 \\ \Theta(1) & \text{otherwise} \end{cases}$$

Where each "$\Theta(1)$" represents a function which is $\Theta(1)$.

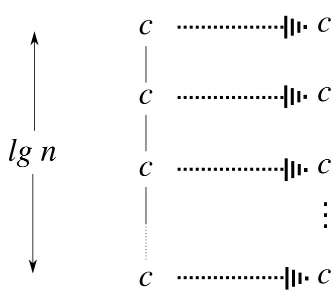
Each theta(1) is a concrete function which is constant in n.

Solution for the recurrence: use a "recursion tree." First adopt some representative function for $\Theta(1)$, e.g., a constant c.



(a) (b) (c)

For this kind of tree, an edge represents addition. And for this tree, its sum of every node represents our running time.



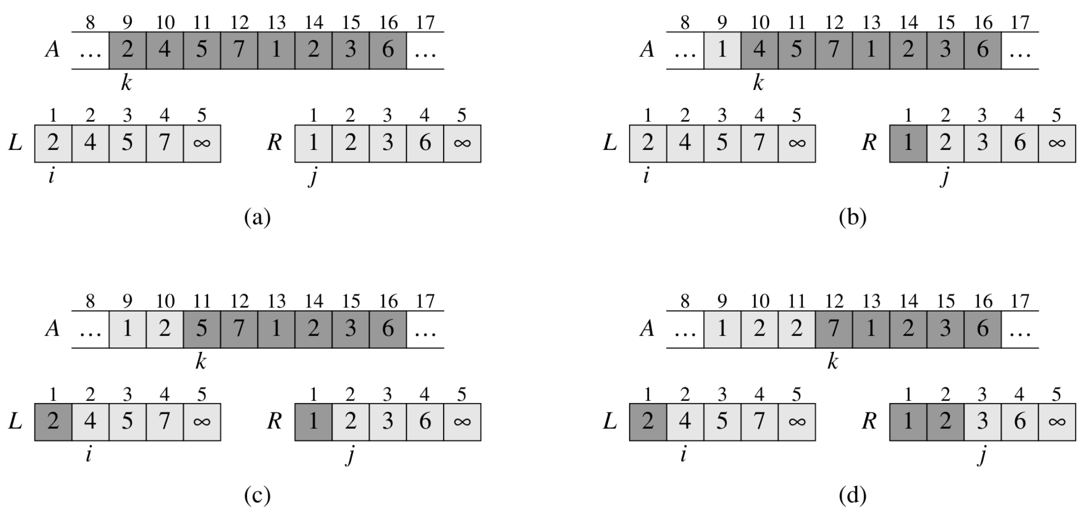
(d)

The height of the tree is $\lg n$ because the value of x for which $n/2^x = 1$ is $x = \log_2 n$.

So total = $c \lg n = \Theta(\lg n)$.

```
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)
```

Merge, via example:



Claim 1. *The number of comparisons* MERGE(A, p, q, r) *performs is* $\Theta(n)$*, where* $n = r - p + 1$*.*

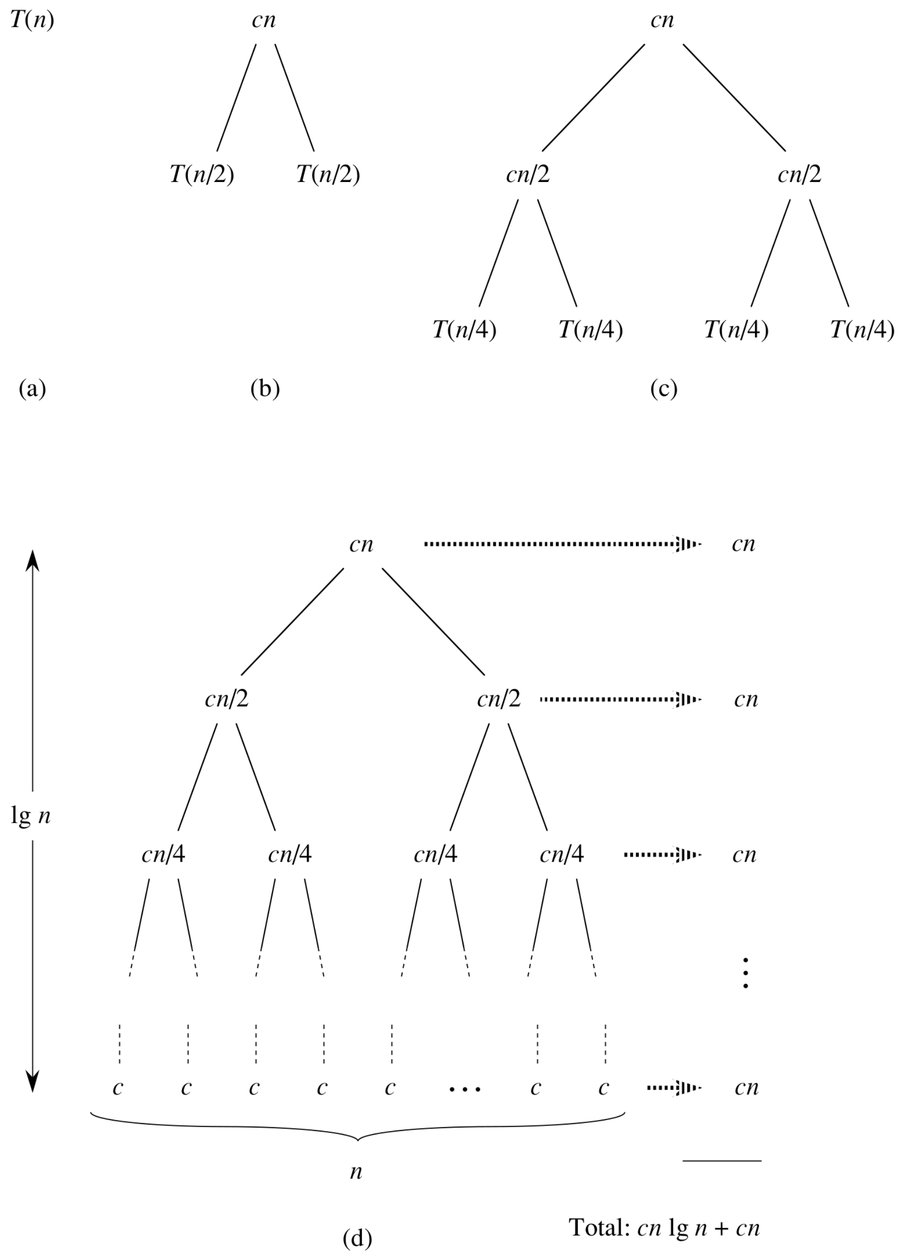
Proof. With each comparison, we "eliminate" exactly one of entries from $A[p, \ldots, q, \ldots, r]$. □

Recurrence for time-efficiency, $T(n)$, of MERGE-SORT:

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

2 Mergesort needs to be performed, so 2T(n/2), and merge has n comparisons, hence theta(n).
For this recursion tree, we adopt cn for theta(n) and c for theta(1).

To solve recurrence, use a recursion tree. Solution: $T(n) = \Theta(n \lg n)$.



Can think of recursion tree as "string rewriting":

$$
\begin{aligned}
T(n) &= \left(2^1\right)T\left(\frac{n}{2^1}\right) + cn \\
&= \left(2^2\right)T\left(\frac{n}{2^2}\right) + \left(2^1\right) \times \left(\frac{cn}{2^1}\right) + \left(2^0\right) \times \left(\frac{cn}{2^0}\right) \\
&= \left(2^2\right)T\left(\frac{n}{2^2}\right) + 2 \times cn \\
&= \left(2^3\right)T\left(\frac{n}{2^3}\right) + 3 \times cn \\
&\ldots \\
&= \left(2^{\log_2 n}\right)T\left(\frac{n}{2^{\log_2 n}}\right) + \log_2 n \times cn \\
&= n \times T(1) + \log_2 n \times cn \\
&= cn + cn\log_2 n = \Theta(n \lg n)
\end{aligned}
$$