```
MERGESORT(A[1..n]):
  if n > 1
      m ← ⌊n/2⌋
      MERGESORT(A[1..m])          ⟨⟨Recurse!⟩⟩
      MERGESORT(A[m+1..n])        ⟨⟨Recurse!⟩⟩
      MERGE(A[1..n], m)
```

```
MERGE(A[1..n], m):
  i ← 1;  j ← m+1
  for k ← 1 to n
      if j > n
          B[k] ← A[i];  i ← i+1
      else if i > m
          B[k] ← A[j];  j ← j+1
      else if A[i] < A[j]
          B[k] ← A[i];  i ← i+1
      else
          B[k] ← A[j];  j ← j+1

  for k ← 1 to n
      A[k] ← B[k]
```

recurrance: function defined in terms of itself

$T(n)$: worst case time to run MergeSort(A[1..n])

$T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil)$

$+\Theta(n)$

$= 2T(n/2) + \Theta(n)$

Recursion Trees

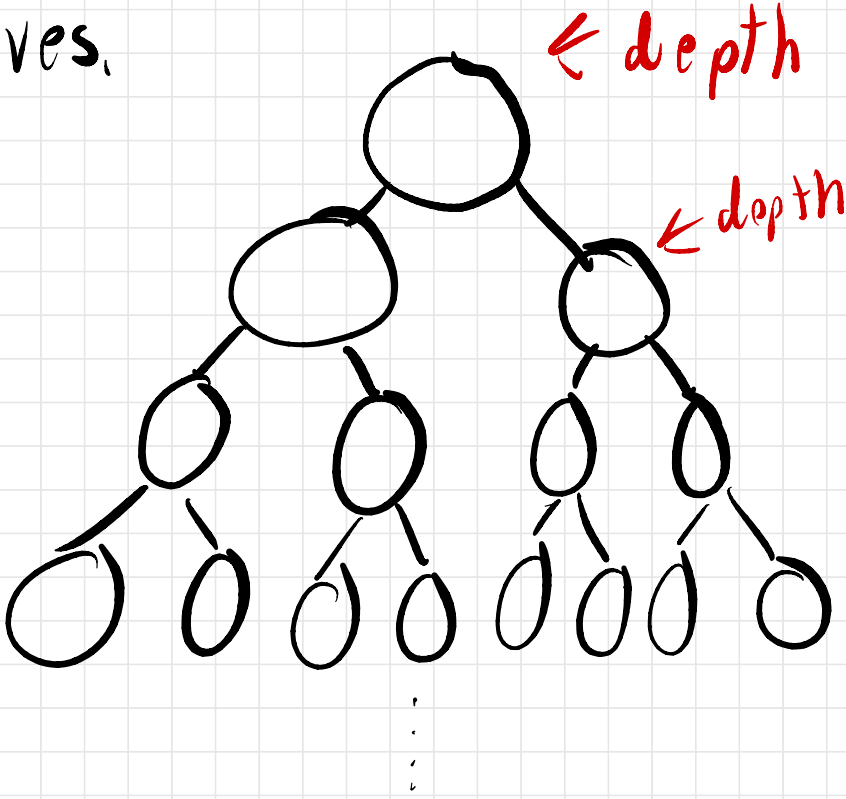
rooted tree

each node represents a single recursive call

root is for MergeSort(A[1..n])

i.e. $T(n)$

Children of a node represent direct recursive calls. Base cases are leaves.



level i: nodes at depth i

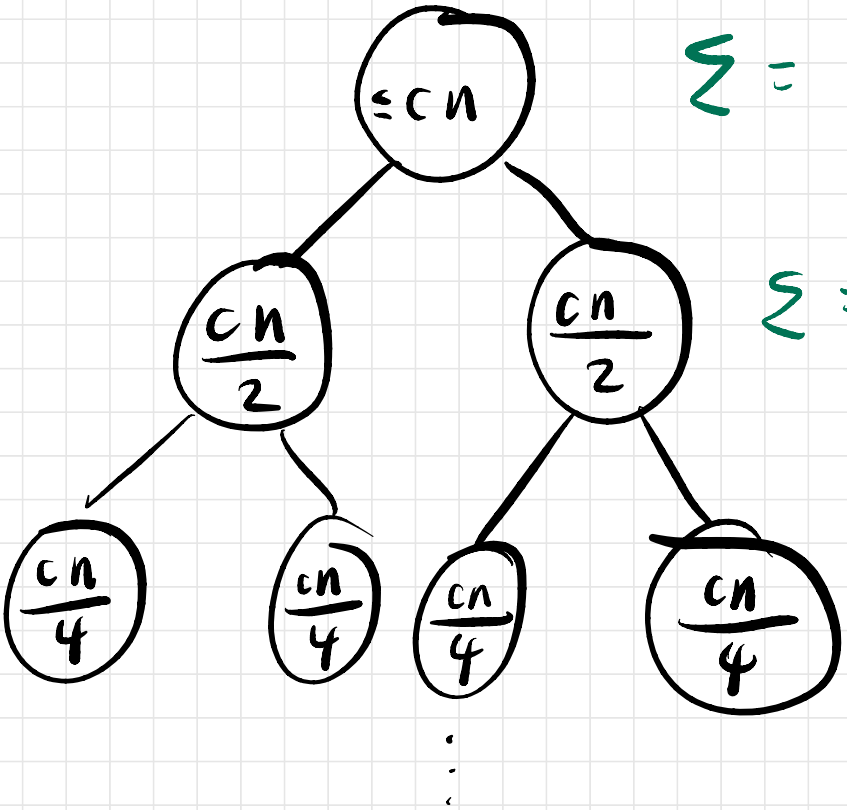
Node's value is its contribution to $T(n)$ outside recursive calls.

$\Rightarrow T(n)$ = sum of node values

$T(n) = 2T(n/2) + \Theta(n)$

by definition of big-Oh, value at root is $\leq cn$ as long as n is "big".

start by finding a big-Oh bound



$\log n$ 0 0 0 0 0 0 0

To find sum of values, sum up the levels' sums.

$T(n) \leq$ #levels $\cdot\ cn$ + (base cases)

$\leq O(\log n) \cdot cn = O(n \log n)$

same math:

$$T(n) \geq \Omega(n \log n)$$

$$\Rightarrow T(n) = \Theta(n \log n)$$

```
QuickSort(A[1..n]):
  if (n > 1)
      Choose a pivot element A[p]
      r ← Partition(A, p)
      QuickSort(A[1..r−1])    ⟨⟨Recurse!⟩⟩
      QuickSort(A[r+1..n])    ⟨⟨Recurse!⟩⟩
```

rank of the pivot

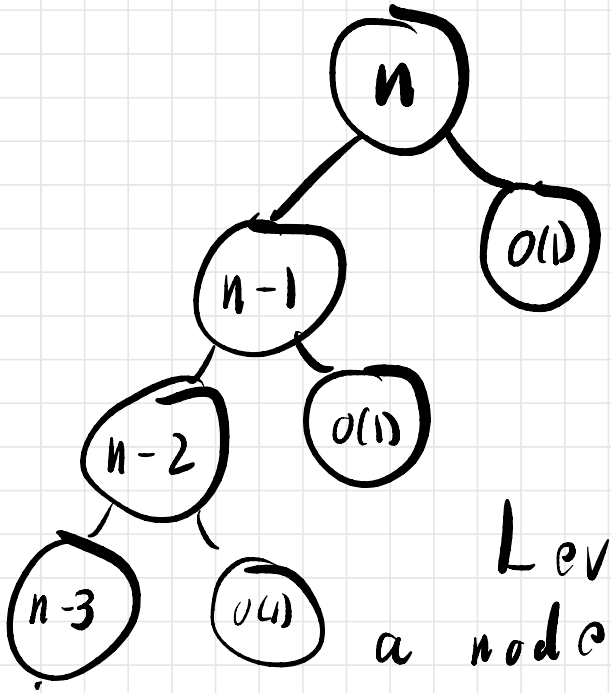
```
Partition(A[1..n], p):
  swap A[p] ↔ A[n]
  ℓ ← 0                       ⟨⟨#items < pivot⟩⟩
  for i ← 1 to n−1
      if A[i] < A[n]
          ℓ ← ℓ + 1
          swap A[ℓ] ↔ A[i]
  swap A[n] ↔ A[ℓ+1]
  return ℓ + 1
```

$T(n)$: worst case time for QuickSort(A[1..n])

$$T(n) = \Theta(n) + \max_{1 \le r \le n} (T(r-1) + T(n-r))$$

What if r = problem size each call?

(ignore the c because we're aiming for a big-Oh anyway)



Level i has a node of value $n-i$.

$$T(n) \geq \sum_i n-i = \Omega(n^2)$$

At most n levels summing to $\leq n$ in all cases,

so $T(n) \leq O(n^2)$

$T(n) = \Theta(n^2)$ ↖ (worst-case!)

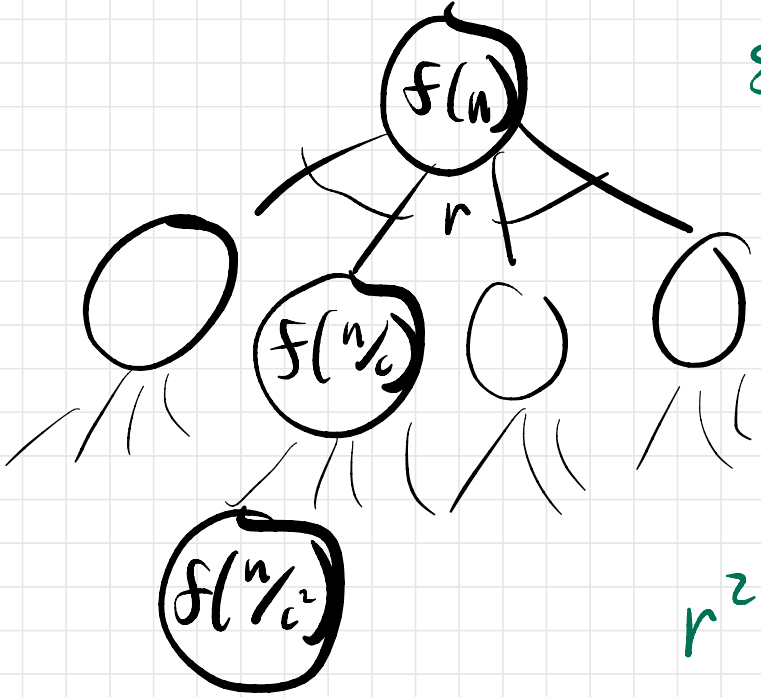
A Common Pattern

An algorithm does some $f(n)$ work outside calls.

It does r recursive call for some constant r.

Subproblems have size n/c for some constant c.

$$T(n) = r \cdot T(n/c) + f(n)$$



$c = f(n)$

$r \cdot f(n/c)$

$r^2 \cdot f(n/c^2)$

$O(1)$ ○ ○ ○ ○ ○

At level i:

Value of a node

$$f(n/c^i)$$

nodes r^i

Sum is $r^i \cdot f(n/c^i)$

L: depth of tree (max depth over all nodes)

$L = \log_c n$

$\Rightarrow$ # leaves $r^L = r^{\log_c n}$

Useful fact from calculus:

geometric series

$$a + as + as^2 + \cdots as^{x-1} = \frac{a(1-s^x)}{1-s}$$

$\neq 1$

If s is a constant $\neq 1$ & z is the largest term, series is in $\Theta(z)$

Three common cases:

$$T(n) = \sum_{i=0}^{L} r^i f(n/c^i)$$

if series is

geometric + decreasing:

$$T(n) = \Theta(f(n))$$

all terms equal:

$$T(n) = L \cdot f(n) = \Theta(f(n) \log_c n)$$
$$= \Theta(f(n) \log n)$$

geometric + increasing:

$$T(n) = \Theta(\#\text{leaves}) = \Theta(r^{\log_c n})$$
$$= \Theta(n^{\log_c r})$$

"Master Method" (CLRS)

Multiplication

Want to multiply x & y (pos. ints). Each has $\leq n$ digits.

Let a, b, c, d s.t.

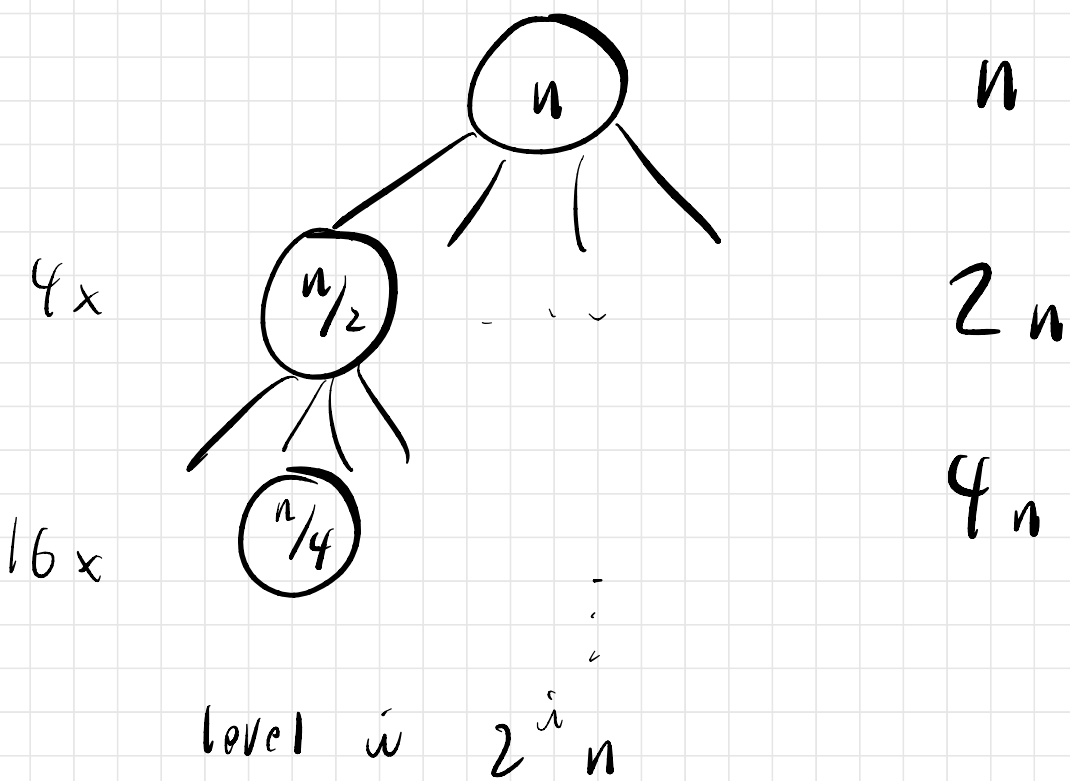
$$x = 10^m a + b \qquad m = \lceil n/2 \rceil$$

$$y = 10^m c + d$$

$$\begin{aligned} xy &= (10^m a + b)(10^m c + d) \\ &= 10^{2m} ac + 10^m (bc + ad) \\ &\quad + bd \end{aligned}$$

```
SplitMultiply(x, y, n):
  if n = 1
      return x · y
  else
      m ← ⌈n/2⌉
      a ← ⌊x/10^m⌋;  b ← x mod 10^m        ⟨⟨x = 10^m a + b⟩⟩
      c ← ⌊y/10^m⌋;  d ← y mod 10^m        ⟨⟨y = 10^m c + d⟩⟩
      e ← SplitMultiply(a, c, m)
      f ← SplitMultiply(b, d, m)
      g ← SplitMultiply(b, c, m)
      h ← SplitMultiply(a, d, m)
      return 10^(2m) e + 10^m (g + h) + f
```

$$T(n) = 4 \cdot T(n/2) + \Theta(n)$$



$$T(n) = \Theta(n^{\log_2 4}) = \Theta(n^2)$$

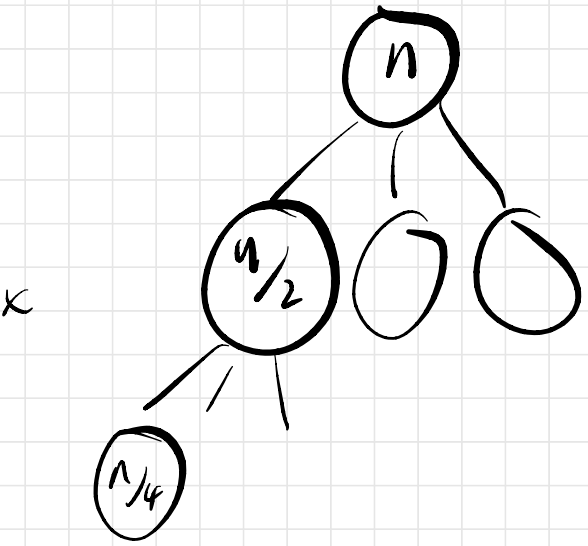
karatsuba

$$bc + ad = ac + bd - ac + bc + ad - bd$$

$$= ac + bd - (a-b)(c-d)$$

```
FastMultiply(x, y, n):
  if n = 1
      return x · y
  else
      m ← ⌈n/2⌉
      a ← ⌊x/10^m⌋;  b ← x mod 10^m        ⟨⟨x = 10^m a + b⟩⟩
      c ← ⌊y/10^m⌋;  d ← y mod 10^m        ⟨⟨y = 10^m c + d⟩⟩
      e ← FastMultiply(a, c, m)
      f ← FastMultiply(b, d, m)
      g ← FastMultiply(a − b, c − d, m)
      return 10^(2m) e + 10^m (e + f − g) + f
```

$$T'(n) = 3 \cdot T(n/2) + \Theta(n)$$



n

$\frac{3}{2} \cdot n$

$(3/2)^2 \cdot n$

level i $\quad (3/2)^i \cdot n$

$$T(n) = \Theta(n^{\log_2 3}) \quad \leftarrow \text{better on homework}$$
$$= O(n^{1.585})$$

Best: $O(n \log n)$

Harvey & van der Hoeven found 2021