Algorithms

More Recursion

Recap

- HW0 due

  A few thoughts: - goal of HW
  - induction
  - recursion

- HW1 posted: Due next Friday
  (may work in groups)

- HW2 will be 1st orally graded HW

- Next week: Finally back to normal,
  I hope!

  Perusall due by 8am on Monday

Recursion trees: Master theorem

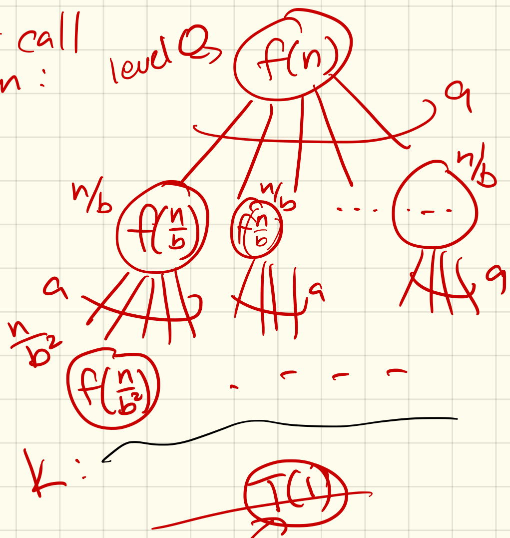
One way to tackle recurrences.
(There are others!)

Big idea: $T(k) = aT\left(\frac{k}{b}\right) + f(k)$

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

Sum total amount of "work":

Root - first call on fcn:



depth

total in tree: $= \sum_{i=0}^{depth} (\text{work on level } i)$

$$\text{total in tree} := \sum_{i=0}^{depth} (\text{work on level } i)$$

$$= \sum_{i=0}^{depth} (\#\text{ nodes in level } i)(\text{work in each node})$$

$$= \sum_{i=0}^{depth} (a^i)\left(f\left(\frac{n}{b^i}\right)\right)$$

series, value can be bounded using identities

depth: d is when

$$\frac{n}{b^d} = 1 \Rightarrow n = b^d$$

$$\log_b n = \log_b(b^d) = d$$

$$d = O(\log n)$$

You saw the merge sort recurrence:

M(n)?

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

$M(\frac{n}{2})$

$M(\frac{n}{2})$

$O(n)$

```
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]
```

Figure 1.6. Mergesort

$$M(n) = 1 + M(\tfrac{n}{2}) + M(\tfrac{n}{2}) + 8 \cdot n$$
$$= 2M(\tfrac{n}{2}) + O(n)$$

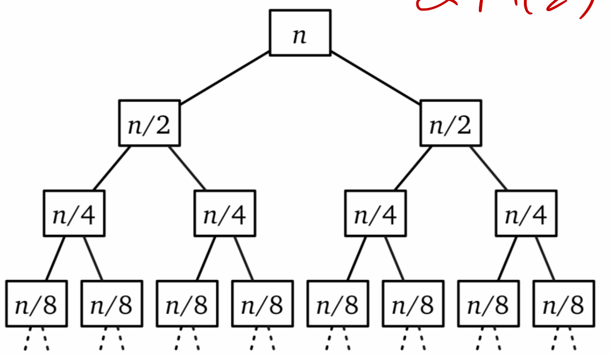


Figure 1.10. The recursion tree for mergesort

Summation: $\displaystyle\sum_{i=0}^{\log_2 n} \cancel{2^i} \cdot \frac{n}{\cancel{2^i}} = n\left[\sum_{i=0}^{\log n} 1\right]$

$= n \log n$

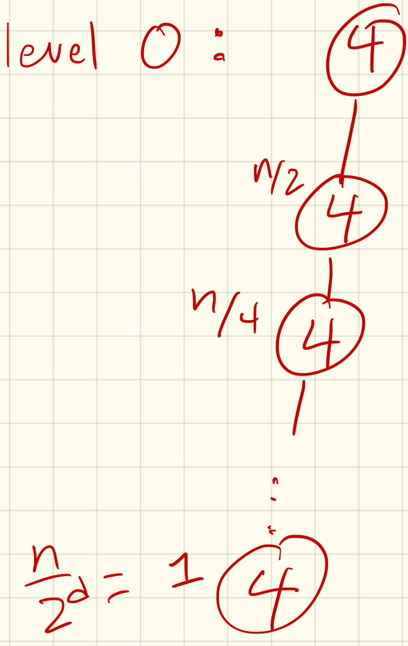
Another: Binary search:

$$B(n) \leq 4 + B\left(\frac{n}{2}\right)$$

$$\leq 1 \cdot B\left(\frac{n}{2}\right) + \cancel{4}$$

$$\Theta(1)$$

Tree:



$$\sum_{i=0}^{\log_2 n} 1 \cdot 4 = \underbrace{(4 + 4 + \dots + 4)}_{\log n}$$

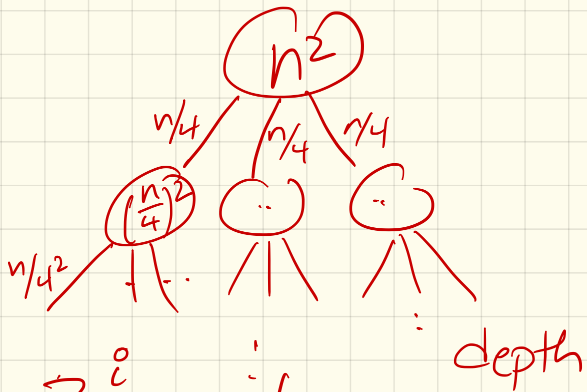
$$= O(\log n)$$

Example:

$$T(n) = 3T\left(\frac{n}{4}\right) + n^2$$

$f(n) = n^2$

$f\left(\frac{n}{4}\right) = \left(\frac{n}{4}\right)^2$



level i : 3^i nodes

each holding $\left(\frac{n}{4^i}\right)^2$

$$T(n) = \sum_{i=0}^{\log_4 n} (3^i)\left(\frac{n}{4^i}\right)^2$$

$$= \sum_{i=0}^{\log n} n^2 \cdot 3^i \cdot \left(\frac{1}{4^i}\right)^2$$

$$= n^2 \left[\sum_{i=0}^{\log n} \left(\frac{3}{16}\right)^i\right] = O(n^2)$$

Ignoring floors & ceilings (& constants):

In practice, even if you don't understand this, the point is you can do it!

The Why: domain transformation

Idea: Exact solution is impossible.

But - upper & lower bound the (messy) summation.

Upper bound:

$$T(n) = T(\lceil \tfrac{n}{2} \rceil) + T(\lfloor \tfrac{n}{2} \rfloor) + n$$

First:

$$T(n) \le S(n)$$

$$T(n) = S(n-2)$$

define $S(n)$, so it's close to $T(n)$ & "master"-like:

$$S(n) = T(n+\alpha)$$

and $\underbrace{S(n) \le 2S(\tfrac{n}{2}) + O(n)}_{\text{"master"-like}}$

How??

Next One: Multiplication

In general, we say this is $O(1)$ time $\longrightarrow$ <u>lies</u>!

In reality:

```
       31415962
     × 27182818
      251327696
      31415962
    251327696
    62831924
  251327696
  31415962
219911734
62831924
853974377340916
```

How to formalize?

nested for loops

Runtime? (2 n-bit #s)

$O(n^2)$

Better: A trick:

$$(10^m a + b)(10^m c + d)$$
$$= 10^{2m} ac + 10^m (bc + ad) + bd$$

Example $\left.\begin{matrix} 963{,}245 \\ 624{,}197 \end{matrix}\right\}$ + $m = 3$:

Make this an algorithm:

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

Runtime:

Hrm - not better after all...

Another trick!

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

Huh?

Recall:

$$(10^m a + b)(10^m c + d)$$
$$= 10^{2m} ac + 10^m (bc + ad) + bd$$

Conclusion:

New & improved pseudocode:

```
FASTMULTIPLY(x, y, n):
    if n = 1
        return x · y
    else
        m ← ⌈n/2⌉
        a ← ⌊x/10^m⌋;  b ← x mod 10^m
        d ← ⌊y/10^m⌋;  c ← y mod 10^m
        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
```

Analysis:

Some comments

- In practice, done in base 2, not 10.

- Actually, this can break down even more!

  If we apply another recursive layer, can get $O(n \log n)$ eventually.

  (Ever heard of Fast Fourier transforms?)