Algorithms, Spring '25

Recursion
(cont)

Recap

Recursion

- If you can solve directly (usually because input is small), do it!
- Otherwise, reduce to simple (usually smaller) instances of the same problem.

Result:

Recursion Fairy

- Helps to solidify that "black box" mentality, so you don't keep unpacking the next level.

(She's also called the "induction hypothesis".)

Merge Sort:

Divide & conquer recurrences
+ proof of correctness

```
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]
```

Figure 1.6. Mergesort

<u>First:</u> correctness, in 2 parts.

<u>Part 1</u>: Merge works

<u>Setup</u>: Given $A[1..n]$ and an index m with $1 \le m \le n$ where $A[1..m]$ & $A[m+1..n]$ are sorted, MERGE correctly sorts $A[1..n]$ by end.

<u>How?</u>

$$A[\ \underset{\substack{\uparrow\\ i}}{\overline{0}}\ \cdots\cdots\ \overline{m}\ \ \underset{\substack{\uparrow\\ j}}{\overline{m+1}}\ \cdots\cdots\ \overline{n}\ ]$$

and $k \leftarrow 0$ to n

<u>So:</u> at iteration k, show we correctly copy k^{th} sorted element.

Backwards induction: consider what is left to sort, ie $n-k$.

<u>Spps $k=n$:</u>

IH:
Now, let $k < n$, & suppose works for any value greater than k:

IS: 4 cases:

```
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]
```

Mergesort: runtime

Quicksort:

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

Solving: worst case!

Note: "Median of three"

- Somewhat better can still be good!

Remember, while $O(n^2)$ worst case, this is the best sorting algorithm in practice.

Issues to consider: (at least outside of 3100)

Recursion Trees:

Let's start with an example.

$$T(n) = 3T\left(\frac{n}{7}\right) + n^2$$

How can I "visualize" the time spent?

Recursion trees (cont)

Next part: how to generalize?

$$T(n) = r\,T\left(\frac{n}{c}\right) + f(n)$$

What it means:

```
Algorithm(n):
    // code
    for i ← 1 to r
        Algorithm(n/c)
    // more code
```

Solving:

Master Theorem:

Combining the three cases above gives us the following "master theorem".

Theorem 1 *The recurrence*

$$\begin{aligned} T(n) &= aT(n/b) + cn^k \\ T(1) &= c, \end{aligned}$$

where a, b, c, and k are all constants, solves to:

$$\begin{aligned} T(n) &\in \Theta(n^k) \text{ if } a < b^k \\ T(n) &\in \Theta(n^k \log n) \text{ if } a = b^k \\ T(n) &\in \Theta(n^{\log_b a}) \text{ if } a > b^k \end{aligned}$$

THEOREM 2 **MASTER THEOREM** Let f be an increasing function that satisfies the recurrence relation

$$f(n) = af(n/b) + cn^d$$

whenever $n = b^k$, where k is a positive integer, $a \geq 1$, b is an integer greater than 1, and c and d are real numbers with c positive and d nonnegative. Then

$$f(n) \text{ is } \begin{cases} O(n^d) & \text{if } a < b^d, \\ O(n^d \log n) & \text{if } a = b^d, \\ O(n^{\log_b a}) & \text{if } a > b^d. \end{cases}$$

Other examples

Medians: find "middle" element.

Two were covered:

```
QuickSelect(A[1..n], k):
  if n = 1
      return A[1]
  else
      Choose a pivot element A[p]
      r ← Partition(A[1..n], p)
      if k < r
          return QuickSelect(A[1..r − 1], k)
      else if k > r
          return QuickSelect(A[r + 1..n], k − r)
      else
          return A[r]
```

Figure 1.12. Quickselect, or one-armed quicksort

Q: How do we know which side has the k^{th} element?