CS3000: Algorithms & Data
Paul Hand

Lecture 3:

- Asymptotic Analysis
- Divide and Conquer: Mergesort

Jan 16, 2019

Asymptotic Analysis

Asymptotic Order Of Growth

- **“Big-Oh” Notation:** $f(n) = O\big(g(n)\big)$ if there exists $c \in (0, \infty)$ and $n_0 \in \mathbb{N}$ such that $f(n) \leq c \cdot g(n)$ for every $n \geq n_0$.
  - Asymptotic version of $f(n) \leq g(n)$
  - Roughly equivalent to $\lim_{n\to\infty} \frac{f(n)}{g(n)} < \infty$

Asymptotic Order Of Growth

- **"Big-Oh" Notation:** $f(n) = O\big(g(n)\big)$ if there exists $c \in (0, \infty)$ and $n_0 \in \mathbb{N}$ such that $f(n) \leq c \cdot g(n)$ for every $n \geq n_0$.
  - Asymptotic version of $f(n) \leq g(n)$
  - Roughly equivalent to $\lim_{n \to \infty} \frac{f(n)}{g(n)} < \infty$

- Activity: Which of these statements are true?
  - $3n^2 + n = O(n^2)$
  - $n^3 = O(n^2)$
  - $10n^4 = O(n^5)$
  - $\log_2 n = O(\log_{16} n)$
  - $n \log_2(n^2) = O(n \log_2 n)$

Big-Oh Rules

- **Constant factors can be ignored**
  - $\forall C > 0 \quad Cn = O(n)$
- **Smaller exponents are Big-Oh of larger exponents**
  - $\forall a > b \quad n^b = O(n^a)$
- **Any logarithm is Big-Oh of any polynomial**
  - $\forall a, \varepsilon > 0 \quad \log_2^a n = O(n^\varepsilon)$
- **Any polynomial is Big-Oh of any exponential**
  - $\forall\ a > 0, b > 1 \quad n^a = O(b^n)$
- **Lower order terms can be dropped**
  - $n^2 + n^{3/2} + n = O(n^2)$

Asymptotic Order Of Growth

- **"Big-Omega" Notation:** $f(n) = \Omega(g(n))$ if there exists $c \in (0, \infty)$ and $n_0 \in \mathbb{N}$ s.t. $f(n) \geq c \cdot g(n)$ for every $n \geq n_0$.
  - Asymptotic version of $f(n) \geq g(n)$
  - Roughly equivalent to $\lim_{n\to\infty} \frac{f(n)}{g(n)} > 0$
- **"Big-Theta" Notation:** $f(n) = \Theta(g(n))$ if there exists $c_1 \leq c_2 \in (0, \infty)$ and $n_0 \in \mathbb{N}$ such that $c_2 \cdot g(n) \geq f(n) \geq c_1 \cdot g(n)$ for every $n \geq n_0$.
  - Asymptotic version of $f(n) = g(n)$
  - Roughly equivalent to $\lim_{n\to\infty} \frac{f(n)}{g(n)} \in (0, \infty)$

Asymptotic Running Times

- **We usually write running time as a Big-Theta**
  - Exact time per operation doesn't appear
  - Constant factors do not appear
  - Lower order terms do not appear

- **Examples:**
  - $30 \log_2 n + 45 = \Theta(\log n)$
  - $Cn \log_2 2n = \Theta(n \log n)$
  - $\sum_{i=1}^{n} i = \Theta(n^2)$

Asymptotic Order Of Growth

- **"Little-Oh" Notation:** $f(n) = o\big(g(n)\big)$ if for every $c > 0$ there exists $n_0 \in \mathbb{N}$ s.t. $f(n) < c \cdot g(n)$ for every $n \geq n_0$.
  - Asymptotic version of $f(n) < g(n)$
  - Roughly equivalent to $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$
- **"Little-Omega" Notation:** $f(n) = \omega\big(g(n)\big)$ if for every $c > 0$ there exists $n_0 \in \mathbb{N}$ such that $f(n) > c \cdot g(n)$ for every $n \geq n_0$.
  - Asymptotic version of $f(n) > g(n)$
  - Roughly equivalent to $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$

Activity

- Fill in the blank with the strongest statement that applies $(O, \Omega, \Theta, o, \omega)$:
  - $15\, n \log_2 n$ = ____ $(\log_2 \sqrt{n})$
  - $n^2 =$_____ $(5\, n^2 + n)$
  - $100n =$ _____ $(5\, n^2 + n)$
  - $3^{\log_2 n} = 2^{\log_3 n}$

Sorting – Insertion Sort and Mergesort

Divide and Conquer Algorithms

- Split your problem into smaller subproblems
- Recursively solve each subproblem
- Combine the solutions to the subprobelms

Divide and Conquer Algorithms

- **Examples:**
  - Mergesort: sorting a list
  - Binary Search: search in a sorted list
  - Karatsuba’s Algorithm: integer multiplication
  - Closest pair of points
  - Fast Fourier Transform
  - …
- **Key Tools:**
  - Correctness: proof by induction
  - Running Time Analysis: recurrences
  - Asymptotic Analysis

Sorting

11	3	42	28	17	8	2	15

$A[1]$ $A[n]$

Given a list of n numbers,
put them in ascending order

2	3	8	11	15	17	28	42

A Simple Algorithm

11	3	42	28	17	8	2	15

A Simple Algorithm: Insertion Sort

Find the maximum

11	3	42	28	17	8	2	15

Swap it into place, repeat on the rest

11	3	15	28	17	8	2	42

11	3	15	2	17	8	28	42

Repeat $n - 1$ times.

2	3	8	11	15	17	28	42

A Simple Algorithm: Insertion Sort

Find the maximum

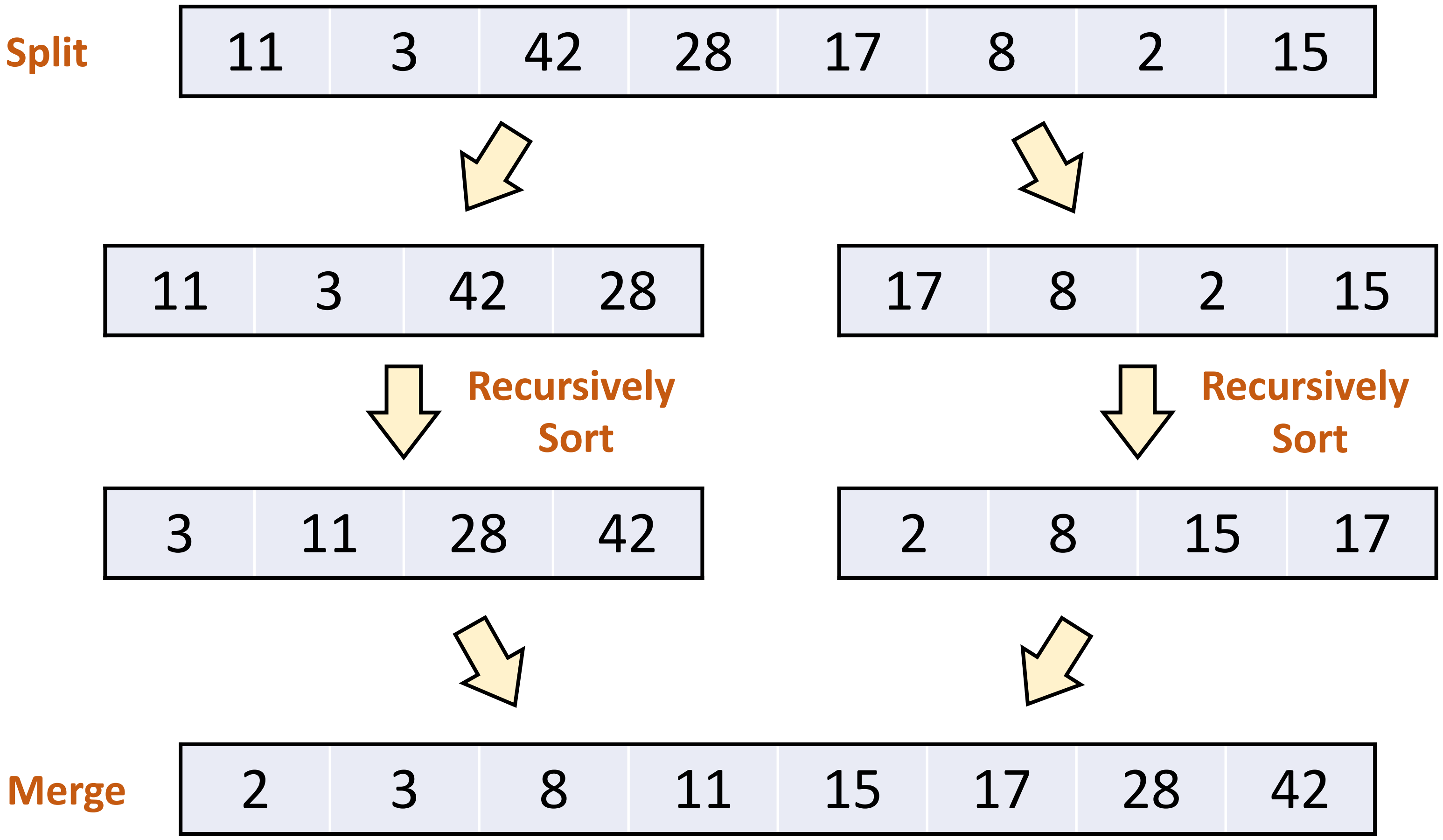
11	3	42	28	17	8	2	15

Swap it into place, repeat on the rest

11	3	15	28	17	8	2	42

Running Time:

Divide and Conquer: Mergesort



Divide and Conquer: Mergesort

- **Key Idea:** If $\boldsymbol{L}, \boldsymbol{R}$ are sorted lists of length n, then we can merge them into a sorted list $\boldsymbol{A}$ of length $2n$ in time Cn
  - Merging two sorted lists is faster than sorting from scratch

3	11	28	42	$\boldsymbol{L}$
2	8	15	17	$\boldsymbol{R}$

								$\boldsymbol{A}$

Merging two sorted lists

```
Merge(L,R):
  Let n ← len(L) + len(R)
  Let A be an array of length n
  j ← 1, k ← 1,

  For i = 1,…,n:
    If (j > len(L)):          // L is empty
      A[i] ← R[k], k ← k+1
    ElseIf (k > len(R)):      // R is empty
      A[i] ← L[j], j ← j+1
    ElseIf (L[j] <= R[k]):    // L is smallest
      A[i] ← L[j], j ← j+1
    Else:                     // R is smallest
      A[i] ← R[k], k ← k+1

  Return A
```

Merging two sorted lists

```
Merge(L,R):
  Let n ← len(L) + len(R)
  Let A be an array of length n
  j ← 1, k ← 1,

  For i = 1,…,n:
    If (j > len(L)):            // L is empty
      A[i] ← R[k], k ← k+1
    ElseIf (k > len(R)):        // R is empty
      A[i] ← L[j], j ← j+1
    ElseIf (L[j] <= R[k]):      // L is smallest
      A[i] ← L[j], j ← j+1
    Else:                       // R is smallest
      A[i] ← R[k], k ← k+1

  Return A
```

- **Prove:** If L and R are sorted from smallest to largest, then A is sorted from smallest to largest.

MergeSort Algorithm

```
MergeSort(A):
  If (len(A) = 1): Return A    // Base Case

  Let m ← ⌈len(A)/2⌉            // Split
  Let L ← A[1:m], R ← A[m+1:n]

  Let L ← MergeSort(L)          // Recurse
  Let R ← MergeSort(R)

  Let A ← Merge(L,R)            // Merge

  Return A
```

Correctness of Mergesort

- **Claim:** The algorithm **Mergesort** is correct

```
MergeSort(A):
  If (len(A) = 1): Return A          // Base Case

  Let m ← ⌊len(A)/2⌋                 // Split
  Let L ← A[1:m], R ← A[m+1:n]

  Let L ← MergeSort(L)               // Recurse
  Let R ← MergeSort(R)

  Let A ← Merge(L,R)                 // Merge

  Return A
```

$\forall n \in \mathbb{N}$ $\forall$ list A with n numbers Mergesort returns A in sorted order

Inductive Hypothesis: $H(n) = \forall$ A of size n MergeSort is correct

Base Case: $H(1)$ is true, obviously

Inductive Step: Assume $H(1), \ldots, H(n)$ are all true. We'll prove $H(n+1)$.

Correctness of Mergesort

- **Claim:** The algorithm **`Mergesort`** is correct

Inductive Step:

Assume: MergeSort is Correct for all A of size $\leq n$

Want to show: MergeSort is correct for all A of Size $n+1$

Consider an A of size $n+1$.

① $\lceil \frac{n+1}{2} \rceil$ & $n - \lceil \frac{n+1}{2} \rceil \leq n$

② L, R both correctly sorted by inductive hypothesis (

③ L, R sorted ⇒ A sorted.

```
MergeSort(A):
  If (len(A) = 1): Return A      // Base Case

  Let m ← ⌈len(A)/2⌉             // Split
  Let L ← A[1:m], R ← A[m+1:n]

  Let L ← MergeSort(L)           // Recurse
  Let R ← MergeSort(R)

  Let A ← Merge(L,R)             // Merge

  Return A
```

Running Time of Mergesort

```
MergeSort(A):
  If (n = 1): Return A

  Let m ← ⌈n/2⌉
  Let   L ← A[1:m]
        R ← A[m+1:n]

  Let L ← MergeSort(L)
  Let R ← MergeSort(R)
  Let A ← Merge(L,R)

  Return A
```

$T(n)$ = time to sort list of size n

$T(1) = c$

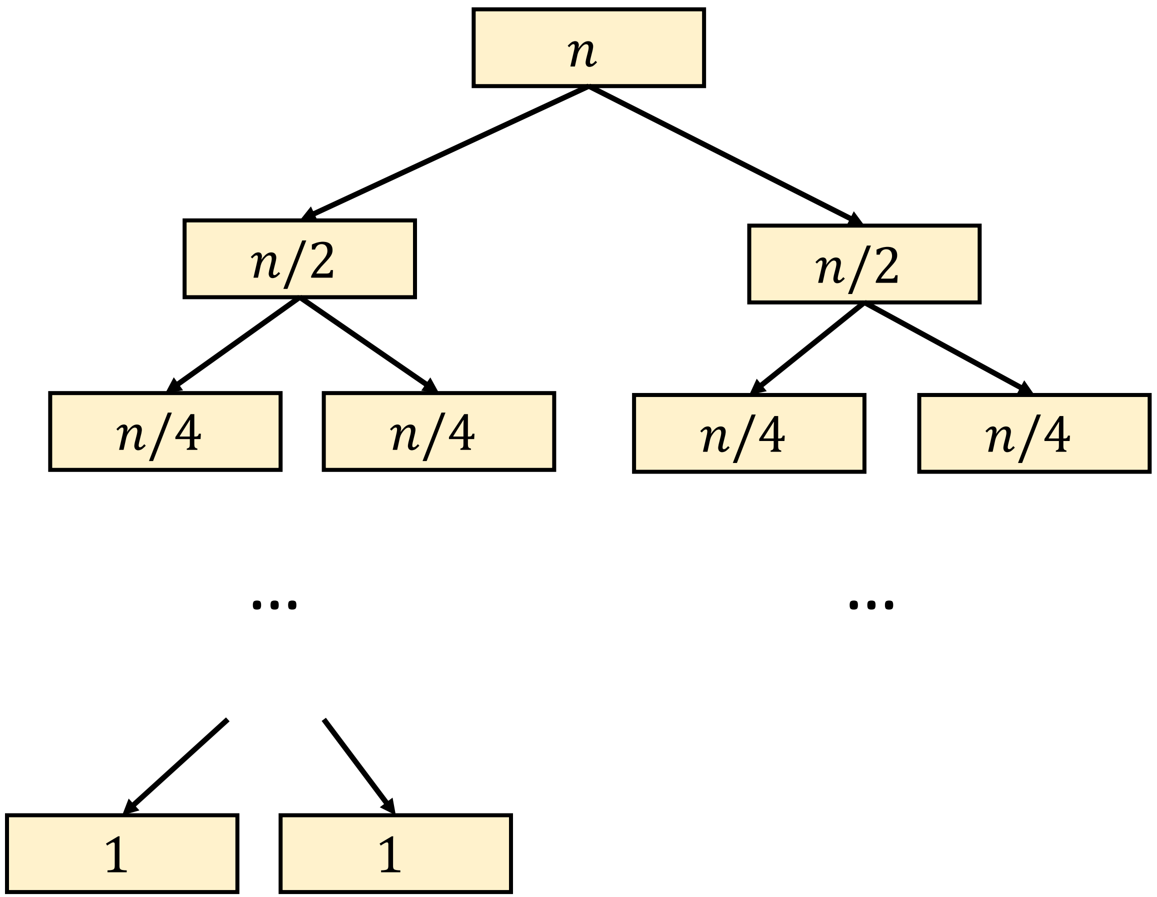
$T(n) = 2\,T(n/2) + cn$

So what is $T(n)$?

Recursion Trees

$$T(n) = 2 \cdot T(n/2) + Cn$$
$$T(1) = C$$

Input Size at this Level



Level	Work at this Level
0	Cn
1	$2 \cdot \left(\frac{Cn}{2}\right) = Cn$
2	$4 \cdot \left(\frac{Cn}{4}\right) = Cn$
i	$2^i \cdot \left(\frac{Cn}{2^i}\right) = Cn$
$\log_2 n$	$2^{\log_2 n} \cdot C = Cn$

Proof by Induction

$T(n) = 2 \cdot T(n/2) + Cn$
$T(1) = C$

- **Claim:** $T(n) = Cn \log_2 2n$

Mergesort Summary

- Sort a list of n numbers in $\Theta(n \log_2 n)$ time
  - Can actually sort anything that allows comparisons
  - No comparison based algorithm can be (much) faster
- Divide-and-conquer
  - Break the list into two halves, sort each one and merge
  - Key Fact: Merging sorted lists is easier than sorting
- Proof of correctness
  - Proof by induction
- Analysis of running time
  - Recurrences