CS3000: Algorithms & Data
Paul Hand

Lecture 3:

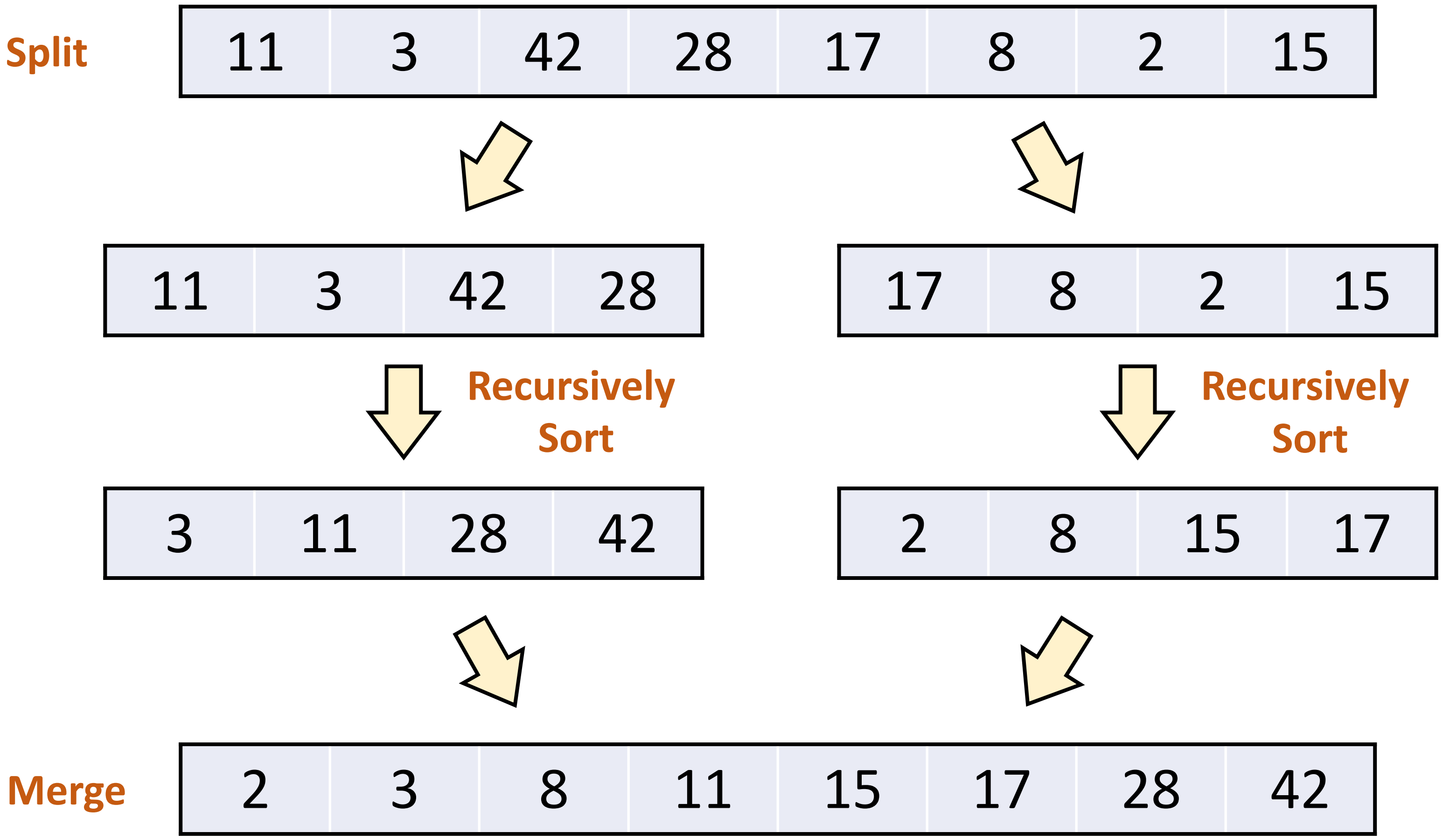
- Asymptotic Analysis
- Divide and Conquer: Mergesort

Jan 16, 2019

Divide and Conquer Algorithms

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

Divide and Conquer: Mergesort



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 $H(1), \dots, H(n)$ 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 $\Rightarrow$ 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

$T(1) = c$

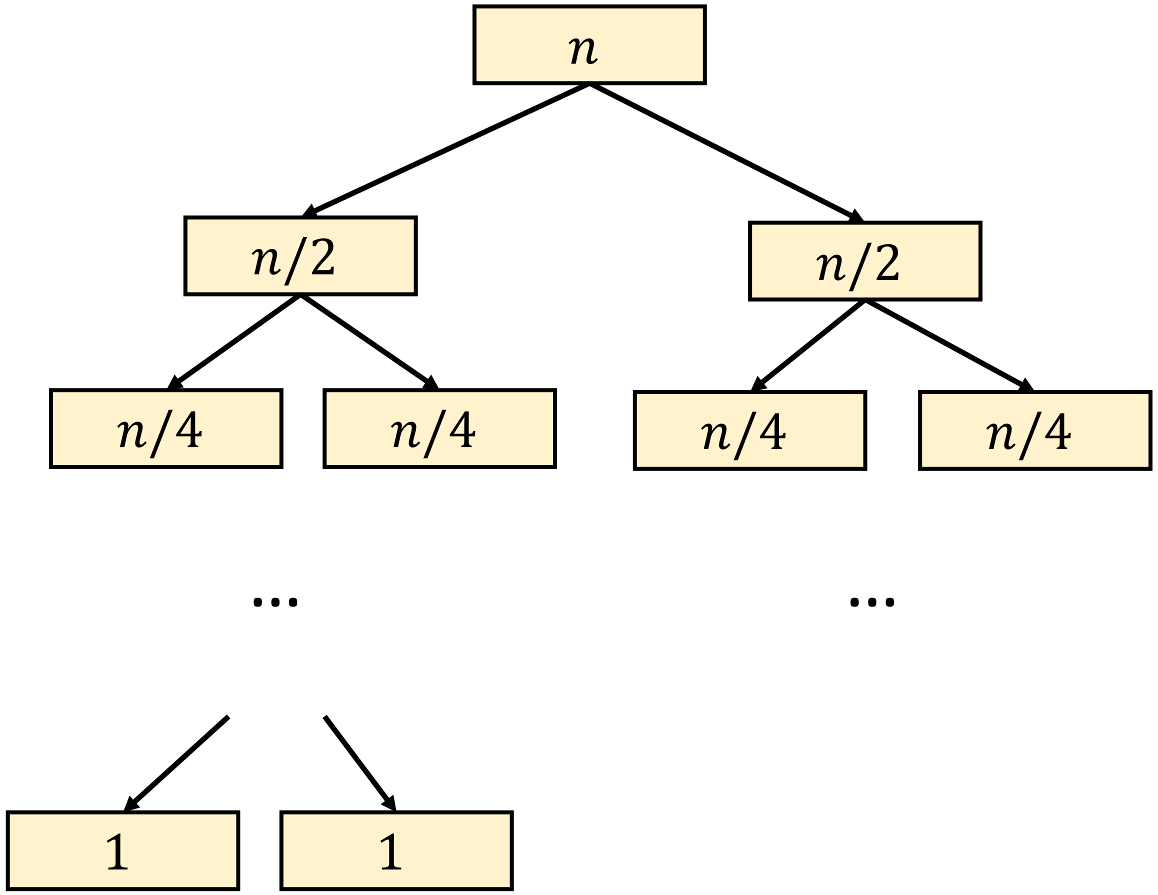
$T(n) = 2\,T(n/2) + Cn$

So what is $T(n)$?

Find Runtime using Recursion Trees and Summation

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

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	$2^i \cdot \left(\frac{Cn}{2^i}\right) = Cn$
$\log_2 n$	$2^{\log_2 n} \cdot C = Cn$

Total Work

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
    - Can prove using recursion tree and summing work at each level
    - Can prove using induction
    - Can use "Master Theorem" for recurrences

Solving Recurrences:
“The Master Theorem”

Activity

- Suppose $r > 0$.
- For large ℓ , how does $S = \sum_{i=0}^{\ell} r^i$ behave?
- What tools do we have to help you? What can you claim?

The “Master Theorem”

- Generic divide-and-conquer algorithm:
  - Split into $\boldsymbol{a}$ pieces of size $\frac{\boldsymbol{n}}{\boldsymbol{b}}$ and merge in time $O\left(n^{\boldsymbol{d}}\right)$
- Recipe for recurrences of the form:
  - $T(n) = \boldsymbol{a} \cdot T(n/\boldsymbol{b}) + Cn^{\boldsymbol{d}}$

Geometric Series

- Claim: $S = \sum_{i=0}^{\ell} r^i = \frac{r^{\ell+1} - 1}{r - 1}$
- Why?

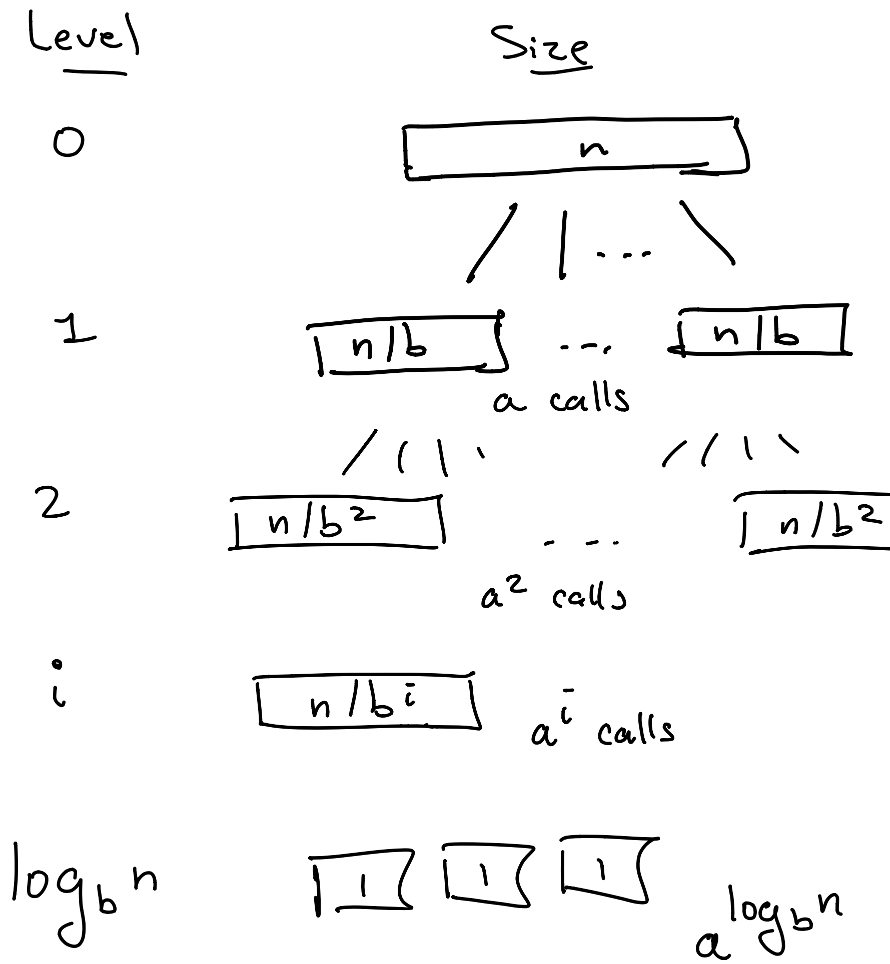
$$S \quad = 1 + r + r^2 + \cdots + r^\ell$$

$$rS = \qquad r + r^2 + \cdots + r^\ell + r^{\ell+1}$$

- Solution $S = \frac{1-r^{\ell+1}}{1-r} = \frac{r^{\ell+1}-1}{r-1}$

Recursion Tree

- $T(n) = \boldsymbol{a}T(n/\boldsymbol{b}) + n^{\boldsymbol{d}}$



Work

n^d

$a \times \left(\frac{n}{b}\right)^d = \left(\frac{a}{b^d}\right) \cdot n^d$

$a^2 \times \left(\frac{n}{b^2}\right)^d = \left(\frac{a}{b^d}\right)^2 \cdot n^d$

$\left(\frac{a}{b^d}\right)^i \cdot n^d$

$a^{\log_b n} = \left(\frac{a}{b^d}\right)^{\log_b n} \cdot n^d$

Recursion Tree

- $T(n) = \boldsymbol{a}T(n/\boldsymbol{b}) + n^{\boldsymbol{d}}$
- $\left(\frac{\boldsymbol{a}}{\boldsymbol{b}^{d}}\right) > 1$

Recursion Tree

- $T(n) = \boldsymbol{a}T(n/\boldsymbol{b}) + n^{\boldsymbol{d}}$
- $\left(\frac{\boldsymbol{a}}{\boldsymbol{b}^{\boldsymbol{d}}}\right) = 1$

Recursion Tree

- $T(n) = \boldsymbol{a}T(n/\boldsymbol{b}) + n^{\boldsymbol{d}}$
- $\left(\frac{\boldsymbol{a}}{\boldsymbol{b}^{d}}\right) < 1$

The "Master Theorem"

- Recipe for recurrences of the form:
  - $T(n) = \boldsymbol{a} \cdot T(n/\boldsymbol{b}) + Cn^{\boldsymbol{d}}$
- Three cases:
  - $\left(\frac{a}{b^d}\right) > 1 : \; T(n) = \Theta\left(n^{\log_{\boldsymbol{b}} \boldsymbol{a}}\right)$
  - $\left(\frac{a}{b^d}\right) = 1 : \; T(n) = \Theta\left(n^{\boldsymbol{d}} \log n\right)$
  - $\left(\frac{a}{b^d}\right) < 1 : \; T(n) = \Theta\left(n^{\boldsymbol{d}}\right)$

Practice

- Use the Master Theorem to Solve:
  - $T(n) = 16 \cdot T\left(\frac{n}{4}\right) + n^2$
  - $T(n) = 21 \cdot T\left(\frac{n}{5}\right) + n^2$
  - $T(n) = 2 \cdot T\left(\frac{n}{2}\right) + 1$
  - $T(n) = 1 \cdot T\left(\frac{n}{2}\right) + 1$

- Recipe for recurrences of the form:
  - $T(n) = \boldsymbol{a} \cdot T(n/\boldsymbol{b}) + Cn^{\boldsymbol{d}}$
- Three cases:
  - $\left(\frac{\boldsymbol{a}}{\boldsymbol{b^d}}\right) > 1 : \; T(n) = \Theta\left(n^{\log_b \boldsymbol{a}}\right)$
  - $\left(\frac{\boldsymbol{a}}{\boldsymbol{b^d}}\right) = 1 : \; T(n) = \Theta\left(n^{\boldsymbol{d}} \log n\right)$
  - $\left(\frac{\boldsymbol{a}}{\boldsymbol{b^d}}\right) < 1 : \; T(n) = \Theta\left(n^{\boldsymbol{d}}\right)$