Objectives

- Divide and conquer problems
- Recurrence relations

Aside: Oscars

- **Patrick Reynold's Status:** "James Franco gave 'Six Degrees of Kevin Bacon -- look it up on the Internet' a shout-out during the Oscars, and my site tipped over. Warn me next time!"
  - "Fixed it! Just needed more Apache processes to allow more users in at once."
  - "For the record, my traffic before the shout-out was about 30 hits per minute. After, 150. After I fixed it, 250 -- and still going strong."

DIVIDE AND CONQUER ALGORITHMS

Divide-and-Conquer

Divide et impera.
Veni, vidi, vici.
- Julius Caesar

- Divide-and-conquer process
  - **Break up** problem into **several parts**
  - Solve each part **recursively**
  - **Combine** solutions to sub-problems into overall solution
- Most common usage:
  - Break up problem of size n into two equal parts of size ½n
  - Solve two parts recursively
  - Combine two solutions into overall solution

Discussion

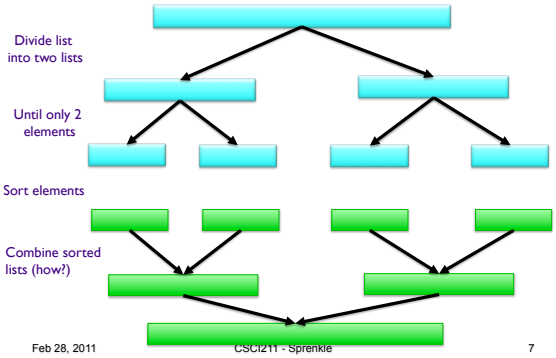
- What is a well-known divide and conquer algorithm?

Merge Sort

Merge Sort

- How does Merge Sort work?
- When do we stop?

Merge Sort



RECURRENCE RELATIONS

Analyzing Merge Sort

General Template
- Break up problem of size n into two equal parts of size ½n
- Solve two parts recursively
- Combine two solutions into overall solution

- **Def.** T(n) = number of comparisons to mergesort an input of size *n*
- Want to say a bit more about what T(n) is
  - Break it down more…

What can we say about the running time w.r.t. to the different parts of the above template?

Analyzing Merge Sort

General Template
- Break up problem of size n into two equal parts of size ½n **O(1)**
- Solve two parts recursively **T(n/2) + T(n/2)**
- Combine two solutions into overall solution **O(n)**

- **Def.** T(n) = number of comparisons to mergesort an input of size *n*
- Want to say a bit more about what T(n) is
  - Break it down more…

What is the base case? Its running time?

Merge Sort's Recurrence Relation

- Put an *upperbound* on T(n):

O(n)

For some constant c,
$T(n) \leq 2\,T(n/2) + cn$ when $n > 2$,
$T(2) \leq c$

Solve T(n) to come up with explicit bound

Approaches to Solving Recurrences

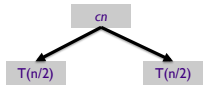
1. Unroll recursion
   - Look for patterns in runtime at each level
   - Sum up running times over all levels
2. Substitute guess solution into recurrence
   - Check that it works
   - Induction on n

Unrolling Recurrence: T(n)

T(n) = 2 T(n/2) + *cn*

Unrolling Recurrence: 2 T(n/2) + cn

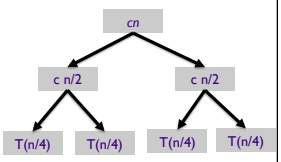
- First level: 2 T(n/2) + *cn*



How does the next level break down?

Unrolling Recurrence: 2 T(n/2) + cn

- Next level:



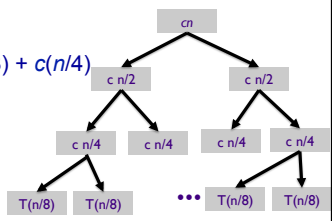
Each one is 2 T(n/4) + *c*(*n*/2)

Next level?

Unrolling Recurrence

- Next level:

Each one is 2 T(n/8) + *c*(*n*/4)

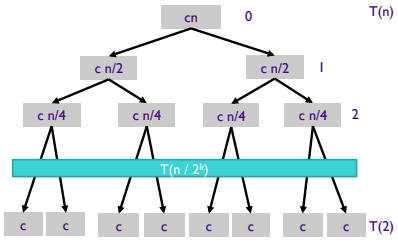


And so on…

What does the final level look like?

Unrolling Recurrence

- How much does each level cost, in terms of the level?
- How many levels are there (assuming *n* is a power of 2)?
- What is the total run time?



Unrolling Recurrence

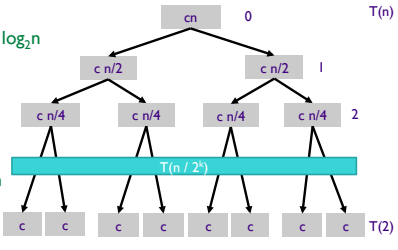
- How many levels are there (assuming *n* is a power of 2)?
- How much does each level cost, in terms of the level?
- What is the total run time?

Number of levels: $\log_2 n$

2^k problems
Size: $n/2^k$

Each level takes
$2^k * c * (n/2^k) = cn$

⇒ O(n log n)



Alternative: Proof by Induction

- Claim. If T(n) satisfies this recurrence, then T(n) = n $\log_2$ n.
  - Recall: T(n) = 2 T(n/2) + *cn*
- Pf. (by induction on n)
  - Base case: n = 1
  - Inductive hypothesis: T(n) ≤ cn $\log_2$ n
  - Goal: show that T(2n) = 2cn $\log_2$ (2n)

Why doubling n?

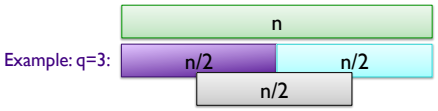
Proof by Induction

- Claim. If T(n) satisfies this recurrence, then T(n) = n $\log_2$ n.
  - Recall: T(n) = 2 T(n/2) + *cn*
- Pf. (by induction on n)
  - Inductive hypothesis: T(n) ≤ cn $\log_2$ n
  - Goal: show that T(2n) = 2cn $\log_2$ (2n)

T(2n) = 2T(n) + c2n
= 2cn $\log_2$n + 2cn Replace w/ induction hypothesis
= 2cn ($\log_2$(2n)-1) + 2cn
= 2cn $\log_2$(2n) - 2cn + 2cn
= 2cn $\log_2$(2n) ✔

Another Example

- Instead of recursively solving 2 problems, solve *q* problems
  - Size of problems is still n/2
- Combining solutions is still O(n)



What is the recurrence relation?

Another Example

- Instead of recursively solving 2 problems, solve *q* problems
  - Size of problems is still n/2
- Combining solutions is still O(n)
- Recurrence relation:
  - For some constant *c*,
    T(n) ≤ *q* T(n/2) + *cn* when n > 2
    T(2) ≤ *c*

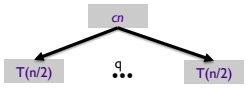
Intuition about running time?

Unrolling Recurrence, q > 2

T(n) ≤ *q* T(n/2) + cn

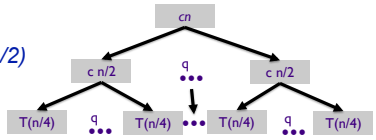
Unrolling Recurrence, q > 2

- First level:
  q T(n/2) + *cn*



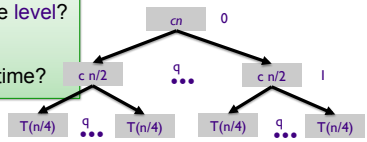
Unrolling Recurrence, q > 2

- Next level:
  q T(n/4) + *c(n/2)*



Unrolling Recurrence, q > 2

How much does each level cost, in terms of the level?
Number of levels?
What is the total run time?



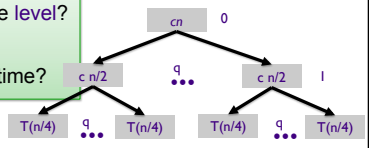
q^k problems at level *k*
Size: $n/2^k$

Number of levels: $\log_2 n$

Each level takes $q^k * c * (n/2^k) = (q/2)^j$ cn
→ Total work per level is *increasing* as level increases

Unrolling Recurrence, q > 2

How much does each level cost, in terms of the level?
Number of levels?
What is the total run time?



$T(n) \leq \Sigma_{j=0,\log n} (q/2)^j$ cn

Geometric series: × → $O(n^{\log_2 q})$

Summary

- Use recurrences to analyze the run time of divide and conquer algorithms
  - Number of sub problems
  - Size of sub problems
  - Number of times divided (number of levels)
  - Cost of merging problems

Know Your Recurrence Relations

What algorithm has this recurrence relation?
What is that algorithm's running time?

Recurrence	Algorithm	Running Time
T(n) = T(n/2) + O(1)		
T(n) = T(n-1) + O(1)		
T(n) = 2 T(n/2) + O(1)		
T(n) = T(n-1) + O(n)		
T(n) = 2 T(n/2) + O(n)		

Know Your Recurrence Relations

What algorithm has this recurrence relation?
What is that algorithm's running time?

Recurrence	Algorithm	Running Time
T(n) = T(n/2) + O(1)	Binary Search	O(log n)
T(n) = T(n-1) + O(1)	Sequential/ Linear Search	O(n)
T(n) = 2 T(n/2) + O(1)	Binary Tree Traversal	O(n)
T(n) = T(n-1) + O(n)	Selection Sort	$O(n^2)$
T(n) = 2 T(n/2) + O(n)	Merge Sort	O(n log n)

Exam 1 Results

- Average: 86%
- Median: 84%
- Standard Deviation: 3 points
  - 9 points difference between high and low score
- Common issues
  - Not explaining how processing original input into some form (include in runtime analysis)
  - Misunderstanding the problem
    - Solving a problem I didn't ask for
  - Not explaining run times
    - Include all parts

Assignments

- Start reading Chapter 5
- Wiki due Wednesday
  - Read 4.5-4.8
- PS5 due Friday
  - SSA but still turn it in