Objectives

- More on recurrence relations
- Divide and conquer algorithms
  - Counting inversions

Review

- Describe the template for divide and conquer solutions
- What is the recurrence relation for Merge Sort?
- What is a recurrence relation?
- How can you compute D&C running times?
  - 2 ways to solve

Review: 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
- Define a **recurrence relation** that describes the running time

Review: Recurrence Relations

- Use recurrences to analyze/determine 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
- How to solve
  - Unrolling
  - Substitution

Analyzing Merge Sort

```
MergeSort( L[n] ):
     if n == 1:
          return L
     if n == 2:
          compare the two entries in L,
          swap if necessary
          return L
     A = MergeSort(L[1…n/2])
     B = MergeSort(L[n/2+1…])
     M = Merge(A, B)
     return M
```

What is the recurrence relation?

Analyzing Merge Sort

```
MergeSort( L[n] ):
     if n == 1:
          return L                    Base cases
     if n == 2:
          compare the two entries in L,
          swap if necessary
          return L
     A = MergeSort(L[1…n/2])     T(n/2)
     B = MergeSort(L[n/2+1…])    T(n/2)
     M = Merge(A, B)             O(n)
     return M
```

What is the recurrence relation?

T(n) = 2T(n/2) + O(n)

Analyzing Binary Search

```
BinarySearch( L[n], key ):
     if n == 1 and L[n] == key:
          return n
     else:
          return NOT_FOUND
     mid = n/2
     if L[mid] == key:
          return mid
     if L[mid] < key:
          return BinarySearch(L[mid+1:], key)
     else:
          return BinarySearch(L[:mid], key)
```

What is the recurrence relation?

Analyzing Binary Search

```
BinarySearch( L[n], key ):
     if n == 1 and L[n] == key:
          return n
     else:
          return NOT_FOUND
     mid = n/2
     if L[mid] == key:
          return mid
     if L[mid] < key:
          return BinarySearch(L[mid+1:], key)
     else:
          return BinarySearch(L[:mid], key)
```

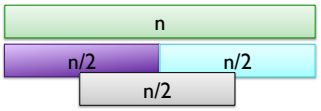
What is the recurrence relation?

T(n) = T(n/2) + c

Another Recurrence Relation

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

Example: q=3:



What is the recurrence relation?

Another Recurrence Relation

- 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*,

    $T(n) \leq q\ T(n/2) + cn$ when $n > 2$

    $T(2) \leq c$

Intuition about running time?

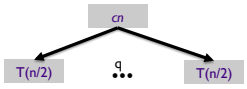
Unrolling Recurrence, q > 2

$T(n) \leq q\ T(n/2) + cn$

Unrolling Recurrence, q > 2

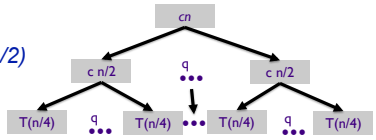
- First level:

  $q\ T(n/2) + cn$



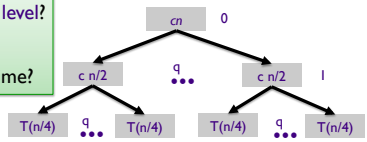
Unrolling Recurrence, q > 2

- 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?



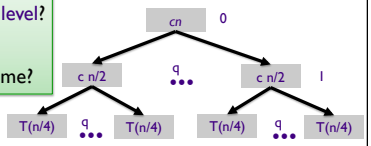
q^k problems at level *k*
Size: $n/2^k$

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:
(constant ratio between successive terms)
Multiplying previous term by (q/2)

$O(n^{\log_2 q})$

Unrolling the Recurrence

- Generalize: What are the steps?

Summary

- Use recurrences to analyze the run time of divide and conquer algorithms
- Need to figure out
  - 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)

Looking Ahead

- Wiki: 4.8, 5.1, 5.2
- Problem Set 6 – due Friday – SSA day