Objectives

- Divide and conquer algorithms
  - Recurrence relations
  - Counting inversions

Review

- What approach are we learning to solve problems (as of Wednesday)?
- What is the recurrence relation for merge sort?
  - What is a recurrence relation in general?

Merge Sort's Recurrence Relation

- $T(n)$ = number of comparisons to mergesort an input of size *n*
- Goal: put an *upperbound* on $T(n)$:

O(n)

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

Solve T(n) to come up with explicit bound

Approaches to Solving Recurrences

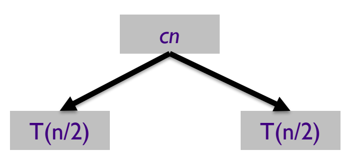
- Unroll recursion
  - Look for patterns in runtime at each level
  - Sum up running times over all levels
- Substitute guess solution into recurrence
  - Check that it works
  - Induction on n

Unrolling Recurrence: T(n)

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

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

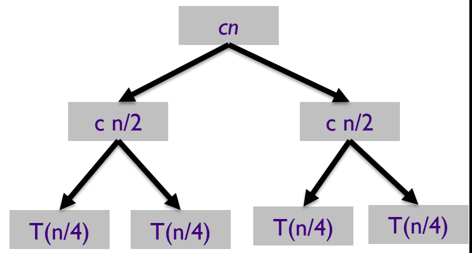
- First level: 2 T(n/2) + *cn*



How does the next level break down?

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

- Next level:



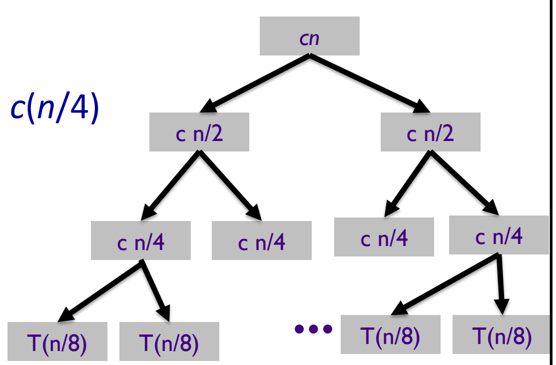
Each one is 2 T(n/4) + $c(n/2)$

Next level?

Unrolling Recurrence

- Next level:

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

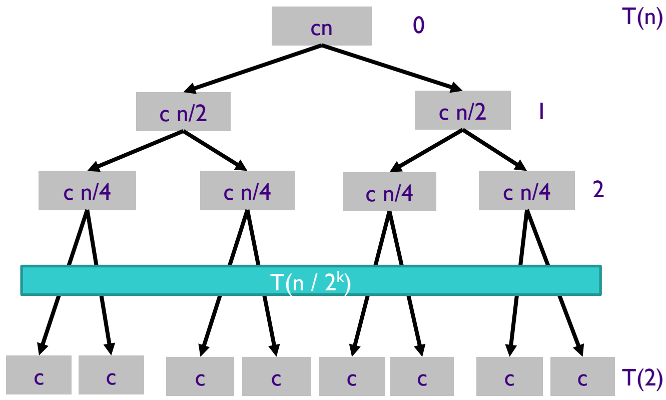


And so on…

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

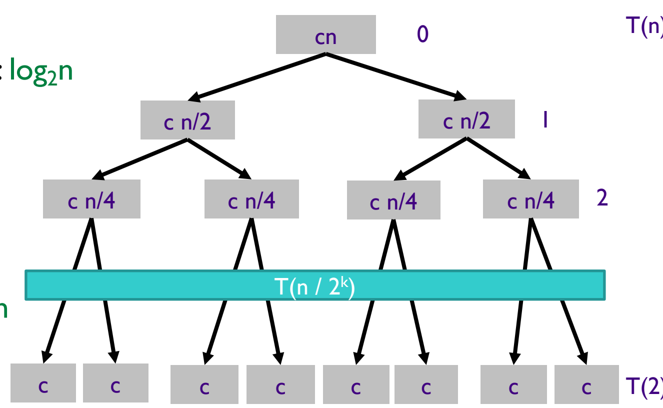
- How many levels are there (assuming *n* is a power of 2)?
- How much does each level cost, in terms of the level?
- What is the total run time?

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 the recurrence $T(n) \le 2\,T(n/2) + cn$, then $T(n) \le cn \log_2 n$.
- Pf. (by induction on n)
  - Base case: $n = 2$
  - Inductive hypothesis: $T(n) \le cn \log_2 n$
  - Goal: show that $T(2n) \le 2cn \log_2 (2n)$

Why doubling n?

Proof by Induction

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

$$\begin{aligned} T(2n) &\le 2T(n) + c2n \\ &\le 2cn \log_2 n + 2cn \\ &\le 2cn (\log_2(2n) - 1) + 2cn \\ &\le 2cn \log_2(2n) - 2cn + 2cn \\ &\le 2cn \log_2(2n) \checkmark \end{aligned}$$

- Recurrence relation
- Replace T(n) w/ induction hypothesis
- Log rules: what is the difference between $\log_2(2n)$ and $\log_2(n)$?

Another Recurrence Relation: Binary Search

- How does binary search work?
- What is its recurrence relation?

Analyzing Binary Search

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

What is the recurrence relation?

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

Unroll the Recurrence

- $T(n) = T(n/2) + c$
- Which makes the runtime?

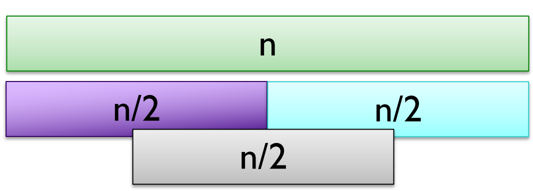
Unroll the Recurrence

- T(n) = T(n/2) + c
  - Constant work at each level
  - Number of levels: log n
- Which makes the runtime? O(log n)

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?

March 8, 2019 CSCI211 - Sprenkle 19

Unrolling Recurrence, q > 2

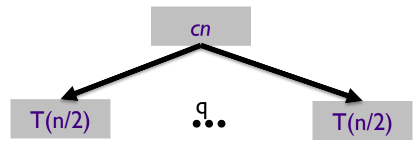
$T(n) \leq q\,T(n/2) + cn$

March 8, 2019 CSCI211 - Sprenkle 20

Unrolling Recurrence, q > 2

- First level:

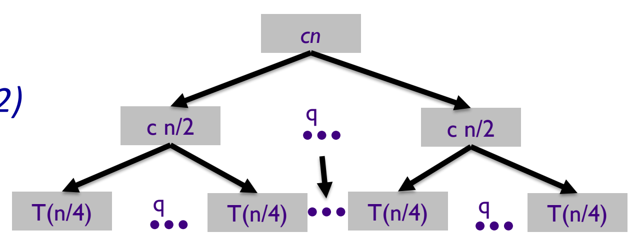
  q T(n/2) + cn



Unrolling Recurrence, q > 2

- Next level:

  q T(n/4) + $c(n/2)$

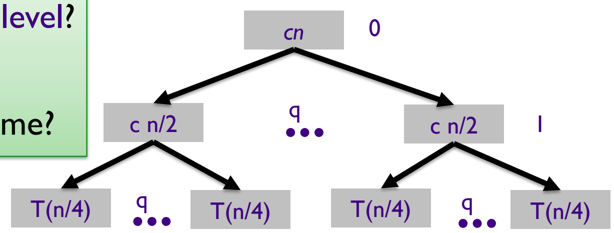


Unrolling Recurrence, q > 2

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: $\log_2 n$

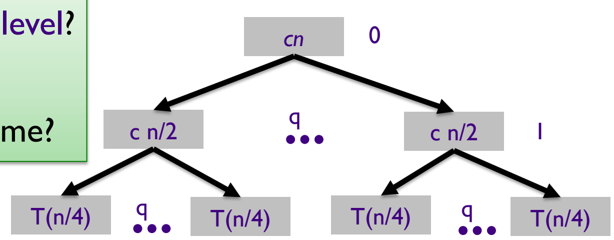
Each level takes $q^k * c * (n/2^k) = (q/2)^j\ cn$
→ Total work per level is *increasing* as level increases

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 runtime 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)		

Looking Ahead

- Problem Set 7 – due next Friday
- Wiki – 4.8, 5-5.3