COT 5407: Introduction to Algorithms

Giri NARASIMHAN

www.cs.fiu.edu/~giri/teach/5407S19.html

Solving Recurrences using Master Theorem

Master Theorem:

Let a,b >= 1 be constants, let f(n) be a function, and let

$$T(n) = aT(n/b) + f(n)$$

1. If f(n) = O($n^{\log_b a-e}$) for some constant e>0, then
   - T(n) = Theta($n^{\log_b a}$)
2. If f(n) = Theta($n^{\log_b a}$), then
   - T(n) = Theta($n^{\log_b a}$ log n)
3. If f(n) = Omega($n^{\log_b a+e}$) for some constant e>0, then
   - T(n) = Theta(f(n))

Solving Recurrences by Substitution

- Guess the form of the solution
- (Using mathematical induction) find the constants and show that the solution works

Example

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

Guess (#1) $T(n) = O(n)$

Need $T(n) <= cn$ for some constant $c>0$

Assume $T(n/2) <= cn/2$ Inductive hypothesis

Thus $T(n) <= 2cn/2 + n = (c+1) n$

Our guess was wrong!!

Solving Recurrences by Substitution: 2

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

Guess (#2) $T(n) = O(n^2)$

Need $T(n) <= cn^2$ for some constant $c>0$

Assume $T(n/2) <= cn^2/4$ Inductive hypothesis

Thus $T(n) <= 2cn^2/4 + n = cn^2/2+ n$

Works for all n as long as $c>=2$!!

But there is a lot of "slack"

Solving Recurrences by Substitution: 3

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

Guess (#3) $T(n) = O(n\log n)$

Need $T(n) <= cn\log n$ for some constant $c>0$

Assume $T(n/2) <= c(n/2)(\log(n/2))$ Inductive hypothesis

Thus $T(n) <= 2\, c(n/2)(\log(n/2)) + n$

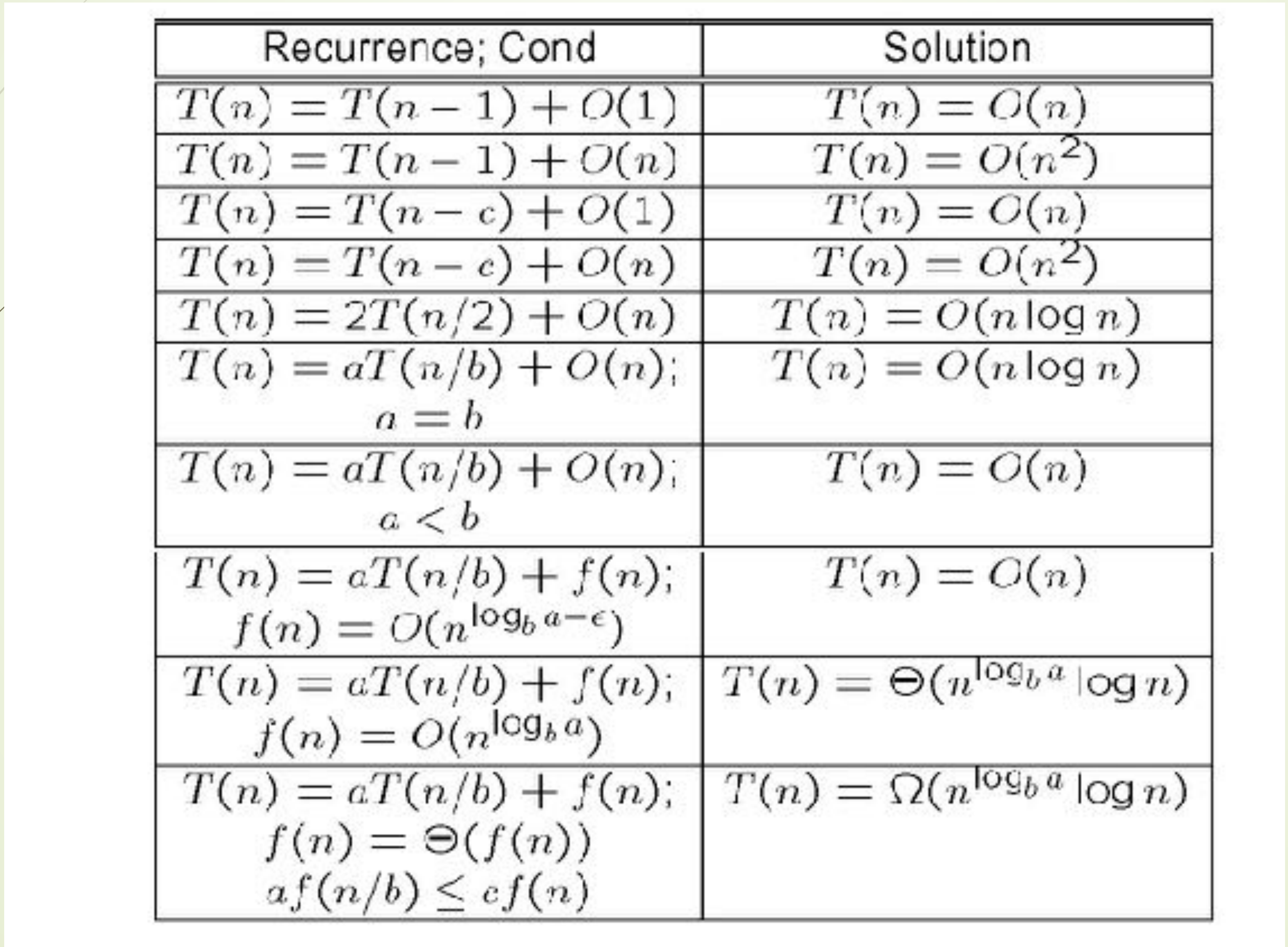
$<= cn\log n - cn + n <= cn\log n$

Works for all n as long as $c>=1$!!

This is the correct guess. WHY?

Show $T(n) >= c'n\log n$ for some constant $c'>0$

Solving Recurrence Relations



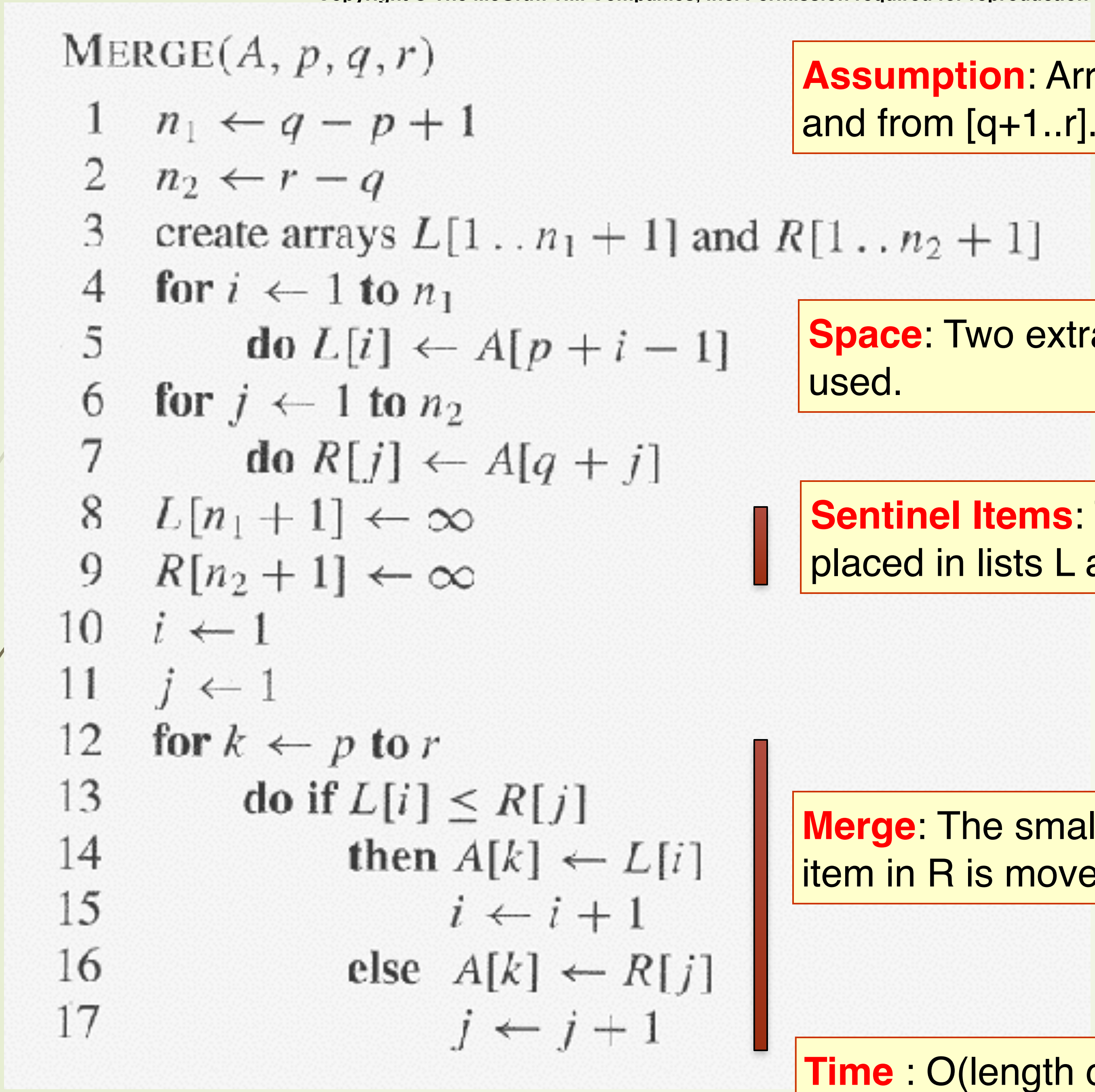
Recurrence; Cond	Solution
$T(n) = T(n-1) + O(1)$	$T(n) = O(n)$
$T(n) = T(n-1) + O(n)$	$T(n) = O(n^2)$
$T(n) = T(n-c) + O(1)$	$T(n) = O(n)$
$T(n) = T(n-c) + O(n)$	$T(n) = O(n^2)$
$T(n) = 2T(n/2) + O(n)$	$T(n) = O(n \log n)$
$T(n) = aT(n/b) + O(n)$; $a = b$	$T(n) = O(n \log n)$
$T(n) = aT(n/b) + O(n)$; $a < b$	$T(n) = O(n)$
$T(n) = aT(n/b) + f(n)$; $f(n) = O(n^{\log_b a - \epsilon})$	$T(n) = O(n)$
$T(n) = aT(n/b) + f(n)$; $f(n) = O(n^{\log_b a})$	$T(n) = \Theta(n^{\log_b a} \log n)$
$T(n) = aT(n/b) + f(n)$; $f(n) = \Theta(f(n))$ $af(n/b) \leq cf(n)$	$T(n) = \Omega(n^{\log_b a} \log n)$

Sorting Algorithms

- SelectionSort
- InsertionSort
- BubbleSort
- ShakerSort
- MergeSort
- HeapSort
- QuickSort
- Bucket & Radix Sort
- Counting Sort

Sorting Algorithms

- Number of Comparisons
- Number of Data Movements
- Additional Space Requirements



```
MERGE$(A, p, q, r)$
  $n_1 \leftarrow q - p + 1$
  $n_2 \leftarrow r - q$
  create arrays $L[1 \,..\, n_1 + 1]$ and $R[1 \,..\, n_2 + 1]$
  for $i \leftarrow 1$ to $n_1$
      do $L[i] \leftarrow A[p + i - 1]$
  for $j \leftarrow 1$ to $n_2$
      do $R[j] \leftarrow A[q + j]$
  $L[n_1 + 1] \leftarrow \infty$
  $R[n_2 + 1] \leftarrow \infty$
  $i \leftarrow 1$
  $j \leftarrow 1$
  for $k \leftarrow p$ to $r$
      do if $L[i] \leq R[j]$
            then $A[k] \leftarrow L[i]$
                 $i \leftarrow i + 1$
            else $A[k] \leftarrow R[j]$
                 $j \leftarrow j + 1$
```

Assumption: Array A is sorted from [p..q] and from [q+1..r].

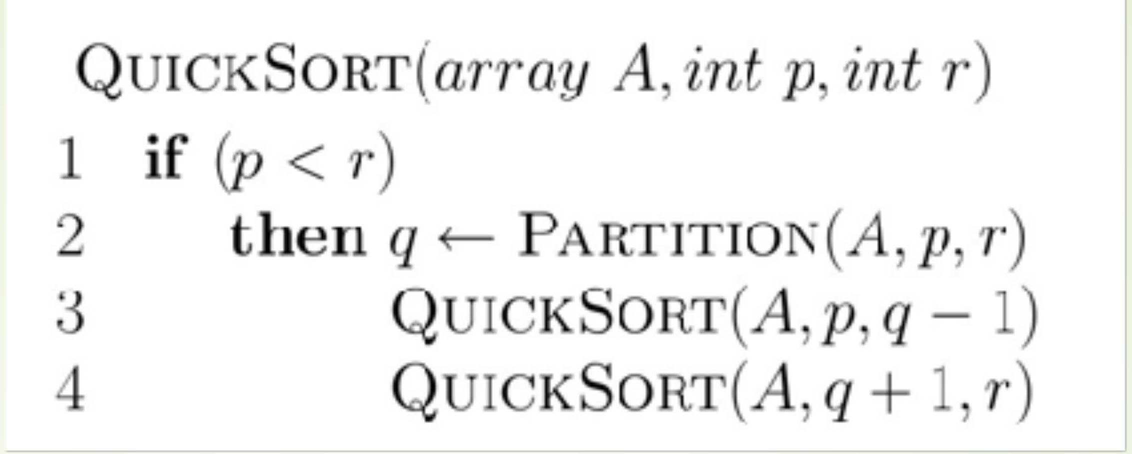
Space: Two extra arrays L and R are used.

Sentinel Items: Two sentinel items placed in lists L and R.

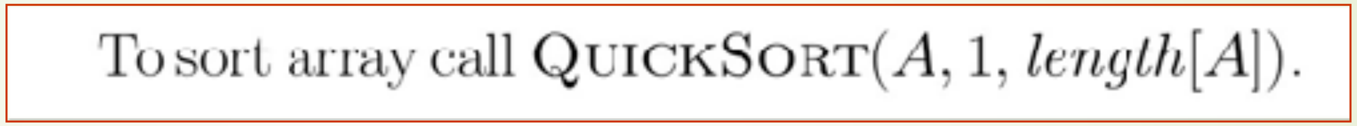
Merge: The smaller of the item in L and item in R is moved to next location in A

Time : O(length of lists)

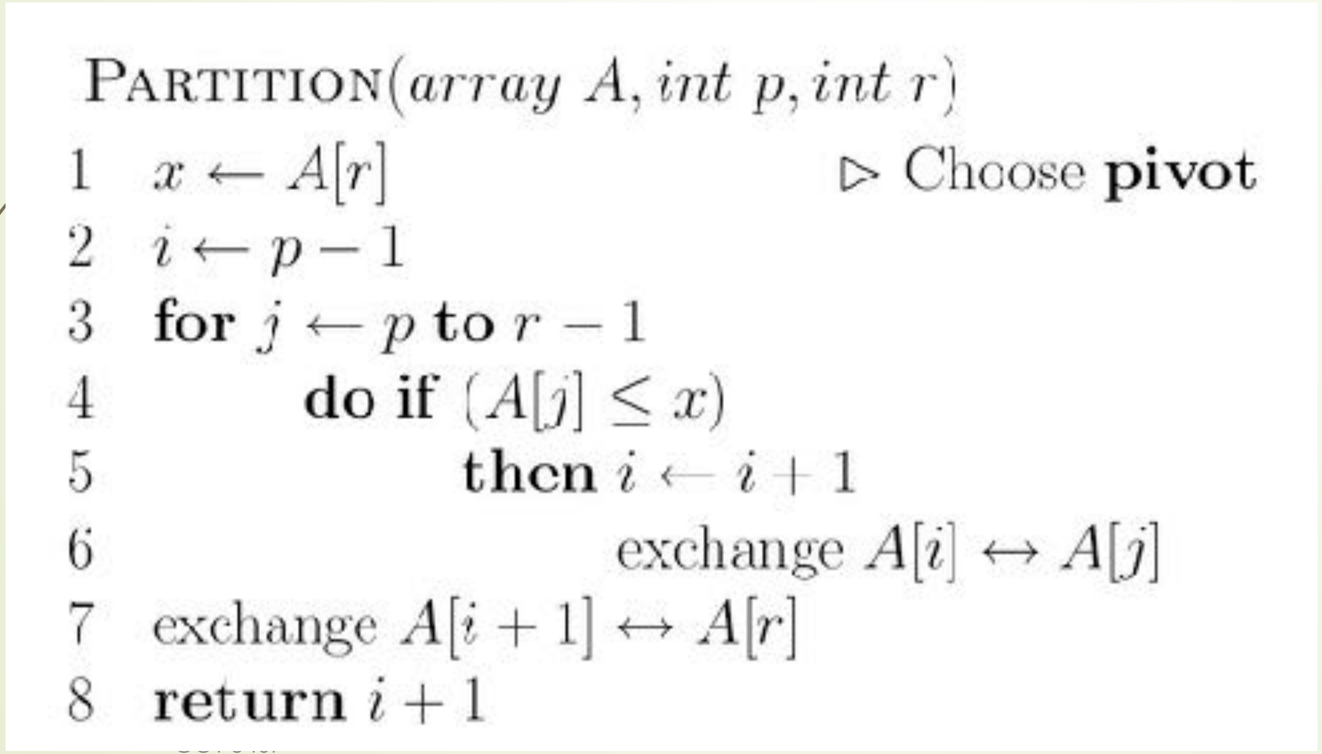
QuickSort



```
QUICKSORT(array $A$, int $p$, int $r$)
if ($p < r$)
    then $q \leftarrow$ PARTITION($A, p, r$)
        QUICKSORT($A, p, q-1$)
        QUICKSORT($A, q+1, r$)
```



To sort array call QUICKSORT(A, 1, $length[A]$).



```
PARTITION(array $A$, int $p$, int $r$)
$x \leftarrow A[r]$                    ▷ Choose **pivot**
$i \leftarrow p-1$
for $j \leftarrow p$ to $r-1$
    do if ($A[j] \leq x$)
        then $i \leftarrow i+1$
            exchange $A[i] \leftrightarrow A[j]$
exchange $A[i+1] \leftrightarrow A[r]$
return $i+1$
```

Page 146, CLRS

Lower Bounds

- It's possible to prove lower bounds for many comparison-based problems.
- For comparison-based problems, for inputs of length N, if there are P(N) possible solutions, then
  - any algorithm needs $\boxed{\log_2(P(N))}$ to solve the problem.
- Binary Search on a list of N items has at least N + 1 possible solutions. Hence lower bound is
  - $\log_2(N+1)$.
- Sorting a list of N items has at least N! possible solutions. Hence lower bound is
  - $\log_2(N!) = O(N \log N)$
- Thus, MergeSort is an optimal algorithm.
  - Because its worst-case time complexity equals lower bound!

k-Selection; Median

- Select the k-th smallest item in list
- Naïve Solution
  - Sort;
  - pick the k-th smallest item in sorted list.

    O(n log n) time complexity
- Idea: Modify Partition from QuickSort
  - How?
- Randomized solution: Average case O(n)
- Improved Solution: worst case O(n)

More Dynamic Operations

	Search	Insert	Delete	Comments
Unsorted Arrays	O(N)	O(1)	O(N)	
Sorted Arrays	O(log N)	O(N)	O(N)	
Unsorted Linked Lists	O(N)	O(1)	O(N)	
Sorted Linked Lists	O(N)	O(N)	O(N)	
Binary Search Trees	O(H)	O(H)	O(H)	H = O(N)
Balanced BSTs	O(log N)	O(log N)	O(log N)	As H = O(log N)

	Se/In/De	Rank	Select	Comments
Balanced BSTs	O(log N)	O(N)	O(N)	
Augmented BBSTs	O(log N)	O(log N)	O(log N)	

OS-Rank

```
OS-RANK(x,y)
// Different from text (recursive version)
// Find the rank of x in the subtree rooted at y
1   r = size[left[y]] + 1
2   if x = y then return r
3   else if ( key[x] < key[y] ) then
4       return OS-RANK(x,left[y])
5   else return r + OS-RANK(x,right[y] )
```

Time Complexity O(log n)

OS-Select

```
OS-SELECT(x,i) //page 304
// Select the node with rank i
// in the subtree rooted at x
1.  r = size[left[x]]+1
2.  if i = r then
3.          return x
4.  elseif  i < r then
5.          return OS-SELECT (left[x], i)
6.  else    return OS-SELECT (right[x], i-r)
```

Time Complexity O(log n)

How to augment data structures

1. choose an underlying data structure
2. determine additional information to be maintained in the underlying data structure,
3. develop new operations,
4. verify that the additional information can be maintained for the modifying operations on the underlying data structure.

Augmenting RB-Trees

Theorem 14.1, page 309

Let f be a field that augments a red-black tree T with n nodes, and f(x) can be computed using only the information in nodes x, left[x], and right[x], including f[left[x]] and f[right[x]].

Then, we can maintain f(x) during insertion and deletion without asymptotically affecting the O(log n) performance of these operations.

For example,

size[x] = size[left[x]] + size[right[x]] + 1

rank[x] = ?

Augmenting information for RB-Trees

- Parent
- Height
- Any associative function on all previous values or all succeeding values.
- Next
- Previous

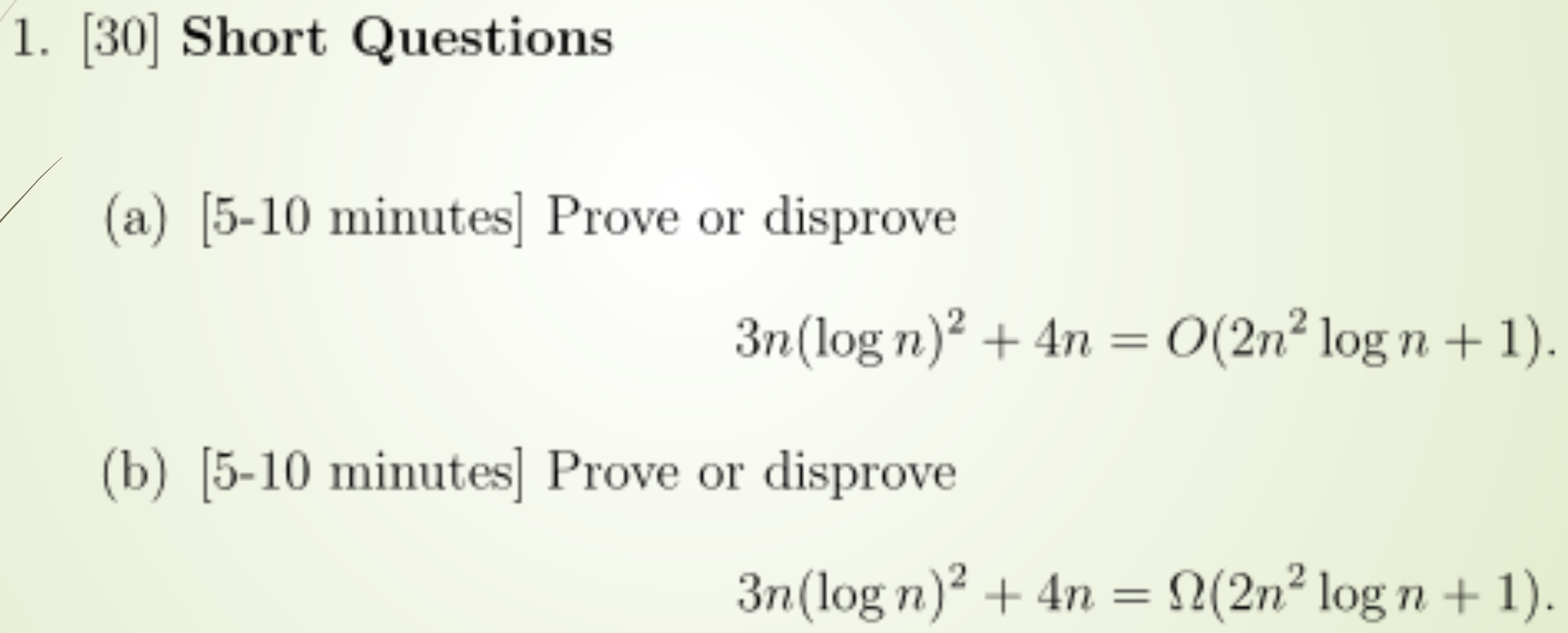
Approach to DP Problems

- Write down a recursive solution
- Use recursive solution to identify list of **subproblems** to solve (there must be overlapping subproblems for effective DP)
- Decide a data structure to store solutions to subproblems (**MEMOIZATION**)
- Write down **Recurrence relation** for solutions of subproblems
- Identify a **hierarchy/order** for subproblems
- Write down non-recursive solution/algorithm

1-d, 2-d, 3-d Dynamic Programming

- Classification based on the dimension of the table used to store solutions to subproblems.
- 1-dimensional DP
  - Activity Problem
- 2-dimensional DP
  - LCS Problem
  - 0-1 Knapsack Problem
  - Matrix-chain multiplication
- 3-dimensional DP
  - All-pairs shortest paths problem

1. Recurrence Relations



1. [30] **Short Questions**

   (a) [5-10 minutes] Prove or disprove

   $$3n(\log n)^2 + 4n = O(2n^2 \log n + 1).$$

   (b) [5-10 minutes] Prove or disprove

   $$3n(\log n)^2 + 4n = \Omega(2n^2 \log n + 1).$$

1. More Recurrence Relations

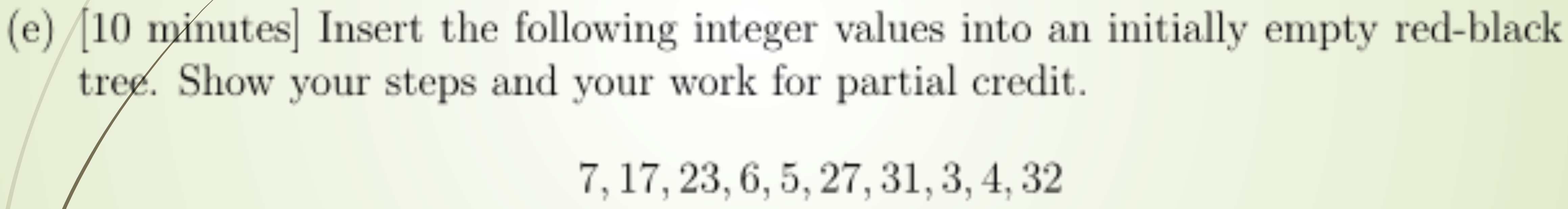
(c) [5-10 minutes] Solve the following recurrence relation using any of the 3 methods we have discussed in class:

$$T(n) = \frac{10}{9}T(4n/5) + O(n)$$

(d) [5-10 minutes] Solve the following recurrence relation using any of the 3 methods we have discussed in class:

$$T(n) = T(4n/7) + O(1)$$

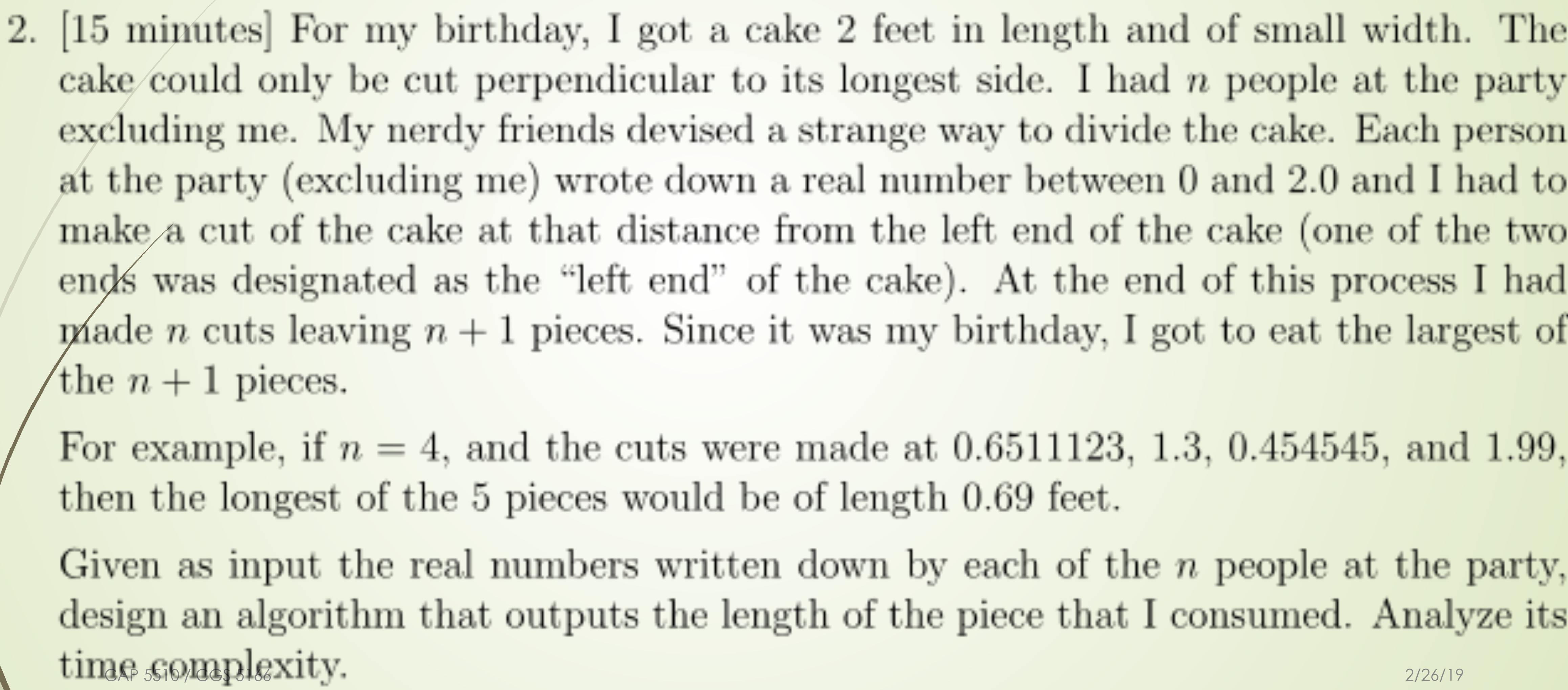
RB-Trees



(e) [10 minutes] Insert the following integer values into an initially empty red-black tree. Show your steps and your work for partial credit.

$$7, 17, 23, 6, 5, 27, 31, 3, 4, 32$$

Have my cake and eat it too …

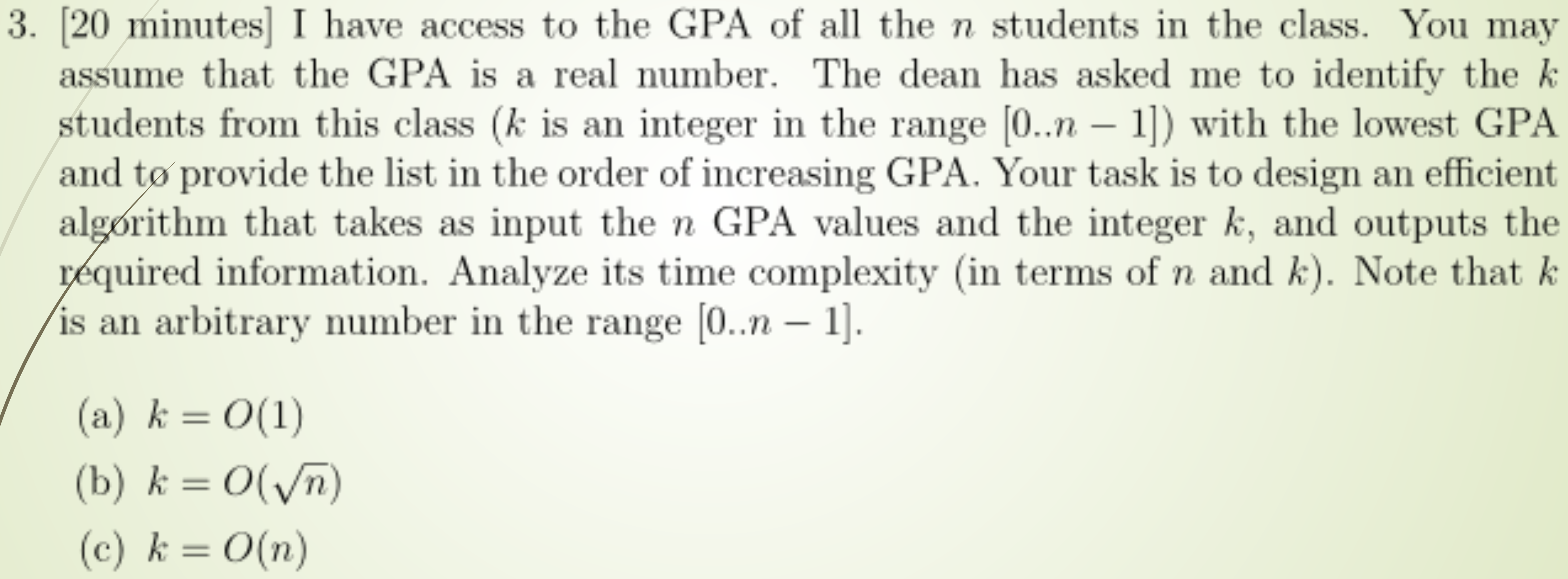


2. [15 minutes] For my birthday, I got a cake 2 feet in length and of small width. The cake could only be cut perpendicular to its longest side. I had n people at the party excluding me. My nerdy friends devised a strange way to divide the cake. Each person at the party (excluding me) wrote down a real number between 0 and 2.0 and I had to make a cut of the cake at that distance from the left end of the cake (one of the two ends was designated as the "left end" of the cake). At the end of this process I had made n cuts leaving $n + 1$ pieces. Since it was my birthday, I got to eat the largest of the $n + 1$ pieces.

For example, if $n = 4$, and the cuts were made at 0.6511123, 1.3, 0.454545, and 1.99, then the longest of the 5 pieces would be of length 0.69 feet.

Given as input the real numbers written down by each of the n people at the party, design an algorithm that outputs the length of the piece that I consumed. Analyze its time complexity.

Finding k poor students



3. [20 minutes] I have access to the GPA of all the n students in the class. You may assume that the GPA is a real number. The dean has asked me to identify the k students from this class (k is an integer in the range $[0..n-1]$) with the lowest GPA and to provide the list in the order of increasing GPA. Your task is to design an efficient algorithm that takes as input the n GPA values and the integer k, and outputs the required information. Analyze its time complexity (in terms of n and k). Note that k is an arbitrary number in the range $[0..n-1]$.

   (a) $k = O(1)$

   (b) $k = O(\sqrt{n})$

   (c) $k = O(n)$