

CS5800 Algorithms

Module 3. More Divide and Conquer

1

Master Method For Solving Recurrences (CLRS 4.5)

- “Cookbook” method for solving recurrences of the form

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

- Proof is presented in CLRS 4.6, left as optional (not covered in class)
- Intuitive understanding: We compare $f(n)$ with $n^{\log_b a}$
 - If $f(n)$ is smaller (polynomially & asymptotically), then $T(n) = \Theta(n^{\log_b a})$.
 - If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \log n)$
 - If $f(n)$ is bigger with some more conditions (see Theorem 4.1), then $T(n) = \Theta(f(n))$

2

Master Method (Revisit)

$$T(n) = 2T\left(\frac{n}{2}\right) + n \lg n$$

Which case?

$$f(n) = n \lg n, n^{\log_b a} = n^{\log_2 2} = n$$

So, $f(n)$ is asymptotically bigger than $n^{\log_b a}$. Is it Case 3?

- Try to prove by the substitution method.

3

$$\text{Solve } T(n) = 2T\left(\frac{n}{2}\right) + n \lg n$$

4

Master Method (Revisit)

- $T(n) = aT\left(\frac{n}{b}\right) + f(n)$
 - Case 1: If there exists a constant $\epsilon > 0$, such that $f(n) = O(n^{\log_b a - \epsilon})$ (polynomially & asymptotically smaller), then $T(n) = \Theta(n^{\log_b a})$.
 - Case 2: If there exists a constant $k \geq 0$, such that $f(n) = \Theta(n^{\log_b a} \lg^k n)$, then $T(n) = \Theta(n^{\log_b a} \lg^{k+1} n)$
 - Case 3: If there exists a constant $\epsilon > 0$, such that $f(n) = \Omega(n^{\log_b a + \epsilon})$, and if $f(n)$ additionally satisfies the *regularity condition*, $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n (polynomially & asymptotically bigger), then $T(n) = \Theta(f(n))$.
(See Theorem 4.1 and Exercise 4.5-5 for the regularity condition – out of this course.)

5

Divide-And-Conquer With Searching

Make Every Algorithm Recursive For The Sake Of Algorithm Analysis Only

6

Binary Search (Exercise 2.3-5)

- Searching a “sorted” array for a value
 - Like looking up a dictionary for a word, or a phone book for a name
- Everyone is expected to write the binary search code (pseudocode or actual language) fluently both recursively and iteratively
 - Remember the “divide-and-conquer” nature
 - And also the “elimination” nature: You can eliminate half of the array once you compare with the middle value
 - What differences & benefits are there in each approach (recursive & iterative)?
 - Keep this question in mind when you experiment the code in the next slides

7

<http://pythontutor.com/>

```
def binary_search(array, value):
    return binary_search_recursive(array, 0, len(array) - 1, value)

def binary_search_recursive(array, left, right, value):
    if left > right:
        return -1

    mid = (left + right) / 2
    if value == array[mid]:
        return mid
    if value > array[mid]:
        return binary_search_recursive(array, mid + 1, right, value)

    return binary_search_recursive(array, left, mid - 1, value)

print(binary_search([0,11,22,33,44,55,66,77,88], 77))
```

8

Iterative version

```
def binary_search_iterative(array, value):
    left = 0
    right = len(array) - 1

    while left <= right:
        mid = (left + right) // 2
        if value == array[mid]:
            return mid
        if value > array[mid]:
            left = mid + 1
        else:
            right = mid - 1
    return -1

print(binary_search_iterative([0,11,22,33,44,55,66,77,88], 77))
```

9

Q: Given an array [0,11,22,33,44,55,66,77,88] and a searched value 77, what is the correct sequence of (left, right) pairs in the binary search? Assume 0 is the starting index.

- a. (1, 8), (4, 8), (6, 8), (7, 7)
- b. (0, 8), (0, 4), (0, 2), (0, 1), (0, 0), (0, -1)
- c. (0, 8), (5, 8), (7, 8)
- d. (0, 8), (4, 8), (6, 9), (7, 7)

10

Q: Given an array [0,11,22,33,44,55,66,77,88] and a searched value 30, what is the correct sequence of (left, right) pairs in the binary search? Assume 0 is the starting index.

- (1, 8), (4, 8), (6, 8), (7, 7)
- (0, 8), (0, 3), (2, 3), (3, 3), (4, 3)
- (0, 8), (0, 3), (2, 3), (3, 3), (3, 2)
- (0, 8), (0, 4), (0, 2), (0, 1), (0, 0), (0, -1)

11

Analysis of Binary Search

- Best case: $T(n) = \Theta(1)$
 - Fixed # operations (1 mid calc op, 1 if, 1 return) when the mid entry is a hit
- Worst case
 - $T(n) = T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + \Theta(1)$
 - $T(0) = \Theta(1)$
- Easier to derive the recurrence from recursive code
- Fewer # steps with iterative code

```
def binary_search_iterative(array, value):
```

```
    left = 0
    right = len(array)-1
    while left <= right:
        mid = (left + right) / 2
        if value == array[mid]:
            return mid
        if value > array[mid]:
            left = mid + 1
        else: # value < array[mid]
            right = mid - 1
    return -1 # No match
```

```
def binary_search_recursive(array, left, right, value):
```

```
    if left > right:
        return -1
    mid = (left + right) / 2
    if value == array[mid]:
        return mid
    if value > array[mid]:
        return binary_search_recursive(array, mid+1, right, value)
    # value < array[mid]
    return binary_search_recursive(array, left, mid-1, value)
```

12

Recursive vs. Iterative Binary Search

- Easier to write recursive code and derive recurrence from it
- Call stack overhead in recursive code : $O(\log n)$ space complexity
- Harder to write iterative code and analyze it
- Faster execution (constant factor speed up) with iterative code, no call stack overhead ($O(1)$ space complexity)
- Quite common to start out writing recursive implementation, then translate it to iterative code for optimization
- Possible variation: What if we need to return the index of the first match when there are multiple matches?

13

Sidebar: Binary Divide-And-Conquer For Searching Unsorted Array

- With an unsorted array, we can't eliminate the problem size by half, like with a sorted array and binary search
- Had to do sequential search, eliminating problem size by one at a time
- Can we do the binary divide-and-conquer with unsorted array as well?
 - Yes, we can. Can you code that?
 - But we won't get any benefit, as our recurrence will be:
 - $T(n) = 2T\left(\frac{n}{2}\right) + \Theta(1) \rightarrow T(n) = \Theta(n)$, not $\Theta(\log n)$
- Can we derive general solution on $T(n) = aT\left(\frac{n}{b}\right) + \Theta(n^c)$?

14

Maximum Subarray Problem (CLRS 4.1)

- Given an example array $A=[13,-3,-25,20,-3,-16,-23,18,20,-7,12,-5,-22,15,-4,7]$,
 - What is the subarray whose sum is the maximum of all subarray sums?
 - In this example, it's $[18,20,-7,12]$ with sum 43.
 - You can confirm this yourself by whatever means
 - It doesn't make much difference between finding the max subarray sum (43) and finding the subarray itself ($[18,20,-7,12]$). Think about why.
 - Interesting only when there are negative numbers in the array.
 - Read CLRS 4.1 for a motivating application of this problem
 - Stock trading to maximize gain when daily change amounts are known.
 - Not a real stock trading technique!

15

Naïve, Brute-Force Solutions

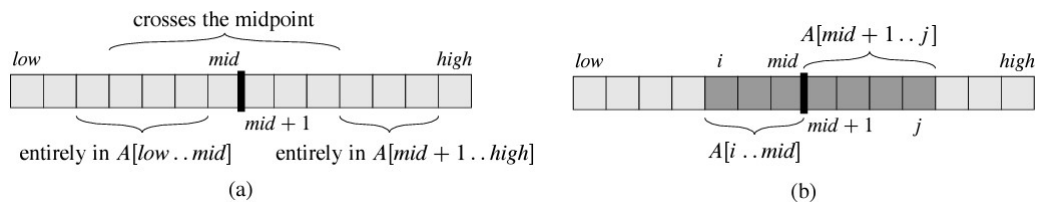
- Evaluate sums of all $A[i..j]$ for any possible i & j , and find the max.
 - $\text{max_subarray_sum} = -\text{infinity}$ (or $A[1]$)
 - for $i=1$ to n (assuming 1-starting array indexing)
 - for $j=i$ to n
 - $\text{subarray_sum}=0$
 - for $k=i$ to j
 - $\text{subarray_sum} += A[k]$
 - If $\text{subarray_sum} > \text{max_subarray_sum}$,
 - $\text{max_subarray_sum} = \text{subarray_sum}$
 - return max_subarray_sum
 - $\Theta(n^3)$: Really naïve, repeating same summation many times
 - $\Theta(n^2)$: By separating out summations, storing them in a 2D array, then doing comparisons, we can achieve this improvement.

16

Can We Do Any Better?

- How about binary divide-and-conquer, like earlier?

- Max subarray sum of $A[\text{low}..\text{high}]$ is the maximum of:
 - Max subarray sum of $A[\text{low}..\text{mid}] \leftarrow$ Recursively computed
 - Max subarray sum of $A[\text{mid}+1..\text{high}] \leftarrow$ Recursively computed
 - Max of sums of subarrays straddling mid
 - Easier than original problem because this problem is constrained.



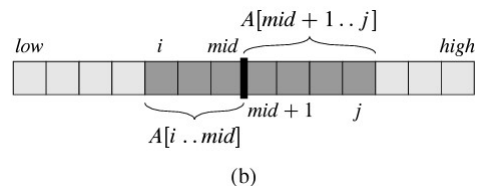
CLRS Fig. 4.4

17

FIND-MAX-CROSSING-SUBARRAY ($A, \text{low}, \text{mid}, \text{high}$)

```

1  left-sum =  $-\infty$ 
2  sum = 0
3  for i = mid downto low
4      sum = sum + A[i]
5      if sum > left-sum
6          left-sum = sum
7          max-left = i
8  right-sum =  $-\infty$ 
9  sum = 0
10 for j = mid + 1 to high
11     sum = sum + A[j]
12     if sum > right-sum
13         right-sum = sum
14         max-right = j
15 return (max-left, max-right, left-sum + right-sum)
    
```



CLRS pp. 71

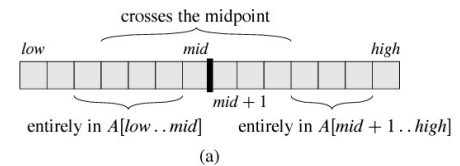
18

FIND-MAXIMUM-SUBARRAY ($A, low, high$)

```

1  if high == low
2      return (low, high, A[low])           // base case: only one element
3  else mid = ⌊(low + high)/2⌋
4      (left-low, left-high, left-sum) =
        FIND-MAXIMUM-SUBARRAY ( $A, low, mid$ )
5      (right-low, right-high, right-sum) =
        FIND-MAXIMUM-SUBARRAY ( $A, mid + 1, high$ )
6      (cross-low, cross-high, cross-sum) =
        FIND-MAX-CROSSING-SUBARRAY ( $A, low, mid, high$ )
7      if left-sum ≥ right-sum and left-sum ≥ cross-sum
8          return (left-low, left-high, left-sum)
9      elseif right-sum ≥ left-sum and right-sum ≥ cross-sum
10         return (right-low, right-high, right-sum)
11     else return (cross-low, cross-high, cross-sum)

```



CLRS pp. 72

19

Analysis of Divide-And-Conquer Max Subarray

- $T(1) = \Theta(1)$: Base case. Recursive case is:
- $T(n) = T\left(\frac{n}{2}\right) + T\left(\frac{n}{2}\right) + T_{crossing}(n) + \Theta(1)$ where:
 - $T_{crossing}(n) = \Theta(n)$
- $T(n) = 2T\left(\frac{n}{2}\right) + \Theta(n)$: Exactly the same as merge sort
- $T(n) = \Theta(n \log n)$
- Achieved $\Theta(n^2)$ to $\Theta(n \log n)$ improvement
 - Actually we can do better and achieve linear time ($\Theta(n)$)
 - Exercise 4.1-5 in pp. 75.
 - It's not a divide-and-conquer solution, though. It's rather a clever intuition.

20

Strassen's Matrix Multiplication Algorithm (CLRS 4.2)

- Multiplying 2 $n \times n$ matrices
 - Study Appendix D if you are not familiar with matrices and operations
- Simple straightforward algorithm, giving $\Theta(n^3)$

SQUARE-MATRIX-MULTIPLY(A, B)

```

1   $n = A.rows$ 
2  let  $C$  be a new  $n \times n$  matrix
3  for  $i = 1$  to  $n$ 
4      for  $j = 1$  to  $n$ 
5           $c_{ij} = 0$ 
6          for  $k = 1$  to  $n$ 
7               $c_{ij} = c_{ij} + a_{ik} \cdot b_{kj}$ 
8  return  $C$ 
```

CLRS pp.75

21

Simple Divide-And-Conquer Matrix Mult.

- Partition each of A , B , and C into 4 quarters:

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}, \quad B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}, \quad C = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}, \quad (4.9)$$

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \cdot \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}. \quad (4.10)$$

$$C_{11} = A_{11} \cdot B_{11} + A_{12} \cdot B_{21}, \quad (4.11)$$

$$C_{12} = A_{11} \cdot B_{12} + A_{12} \cdot B_{22}, \quad (4.12)$$

$$C_{21} = A_{21} \cdot B_{11} + A_{22} \cdot B_{21}, \quad (4.13)$$

$$C_{22} = A_{21} \cdot B_{12} + A_{22} \cdot B_{22}. \quad (4.14)$$

- Can we reduce # multiplications some way?

22

Strassen's Improvement

- Not sure how Strassen found this, but he observed that, by letting:

$$\begin{aligned}
 S_1 &= B_{12} - B_{22}, & P_1 &= A_{11} \cdot S_1 = A_{11} \cdot B_{12} - A_{11} \cdot B_{22}, \\
 S_2 &= A_{11} + A_{12}, & P_2 &= S_2 \cdot B_{22} = A_{11} \cdot B_{22} + A_{12} \cdot B_{22}, \\
 S_3 &= A_{21} + A_{22}, & P_3 &= S_3 \cdot B_{11} = A_{21} \cdot B_{11} + A_{22} \cdot B_{11}, \\
 S_4 &= B_{21} - B_{11}, & P_4 &= A_{22} \cdot S_4 = A_{22} \cdot B_{21} - A_{22} \cdot B_{11}, \\
 S_5 &= A_{11} + A_{22}, & P_5 &= S_5 \cdot S_6 = A_{11} \cdot B_{11} + A_{11} \cdot B_{22} + A_{22} \cdot B_{11} + A_{22} \cdot B_{22}, \\
 S_6 &= B_{11} + B_{22}, & P_6 &= S_7 \cdot S_8 = A_{12} \cdot B_{21} + A_{12} \cdot B_{22} - A_{22} \cdot B_{21} - A_{22} \cdot B_{22}, \\
 S_7 &= A_{12} - A_{22}, & P_7 &= S_9 \cdot S_{10} = A_{11} \cdot B_{11} + A_{11} \cdot B_{12} - A_{21} \cdot B_{11} - A_{21} \cdot B_{12}, \\
 S_8 &= B_{21} + B_{22}, \\
 S_9 &= A_{11} - A_{21}, \\
 S_{10} &= B_{11} + B_{12}.
 \end{aligned}$$

23

- Submatrix C_{ij} can be represented as follows:

$$C_{11} = P_5 + P_4 - P_2 + P_6, \quad C_{12} = P_1 + P_2, \quad C_{21} = P_3 + P_4, \quad C_{22} = P_5 + P_1 - P_3 - P_7,$$

- Algebraic proofs shown in CLRS pp. 81
- We've now got $7 \frac{n}{2} \times \frac{n}{2}$ multiplications and 18 additions

- 18 additions are still $\Theta(n^2)$.
- Thus new recurrence is

$$T(n) = 7T\left(\frac{n}{2}\right) + \Theta(n^2)$$

- Solution to this recurrence (using master theorem, which will be presented later) is $T(n) = \Theta(n^{\log_2 7}) \cong \Theta(n^{2.81})$

24

Quicksort

Fastest Sorting Algorithm On Average, How To Prove That

25

Recall Merge Sort

- “Divide”: easy. Just divide the input array into two sub-arrays.
- “Conquer”: hard. It takes $O(n)$ time complexity.

⇒ What if divide the input array into 3 sub-arrays?

⇒ “Divide”: $O(1)$

⇒ “Conquer”: still $O(n)$

⇒ $T(n) = 3T\left(\frac{n}{3}\right) + O(n)$

⇒ Overall, the same time complexity as Merge Sort

⇒ What if divide the input array into \sqrt{n} sub-arrays?

⇒ “Divide”: $O(\sqrt{n})$

⇒ “Conquer”: $O(n\sqrt{n})$

⇒ $T(n) = \sqrt{n}T(\sqrt{n}) + O(n\sqrt{n})$

⇒ Overall, it is worse than Merge Sort

⇒ If we can make the “conquer” part in $O(n)$, the time complexity becomes $O(n \log \log n)$. But, can we?

26

Quicksort: Different Kind Of Divide & Conquer

- So far, “divide” was straightforward, and “conquer” was involved.
- In sorting, can we make “conquer” part easy (almost nothing), by doing more on “divide” part?
 - CLRS 7.1 “Divide”: **Partition** (rearrange) the array $A[p..r]$ into two (either one of the two may be empty) subarrays $A[p..q-1]$ and $A[q+1..r]$ such that:
 - $A[i] \leq A[q]$ for any $p \leq i < q$, and
 - $A[j] > A[q]$ for any $q < j \leq r$.
 - CLRS 7.1 “Conquer”: Then conquering becomes straightforward:
 - Sort $A[p..q-1]$ recursively
 - Sort $A[q+1..r]$ recursively

QUICKSORT(A, p, r)

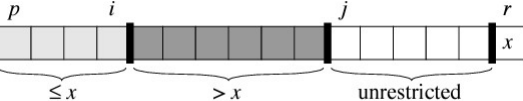
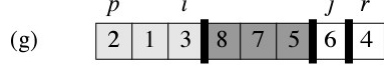
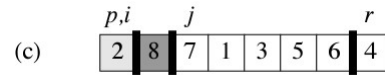
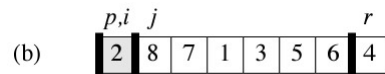
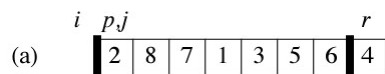
```

1  if  $p < r$ 
2     $q = \text{PARTITION}(A, p, r)$ 
3    QUICKSORT( $A, p, q - 1$ )
4    QUICKSORT( $A, q + 1, r$ )
    
```

To sort an entire array A , the initial call is QUICKSORT($A, 1, A.length$).

27

PARTITION(A, p, r) Illustration



PARTITION(A, p, r)

```

1   $x = A[r]$ 
2   $i = p - 1$ 
3  for  $j = p$  to  $r - 1$ 
4    if  $A[j] \leq x$ 
5       $i = i + 1$ 
6      exchange  $A[i]$  with  $A[j]$ 
7  exchange  $A[i + 1]$  with  $A[r]$ 
8  return  $i + 1$ 
    
```

28

PARTITION() Can Be Recursive As Well

- Maybe more overhead, but maybe easier to understand

i	p	j						r
	2	8	7	1	3	5	6	4

- If $A[p] \leq A[r]$, return $\text{PARTITION}(A, p + 1, r)$
- Otherwise, 3-way swap between $A[p]$, $A[r]$, and $A[r - 1]$
 - $A[p]$ to $A[r]$, $A[r]$ to $A[r - 1]$, $A[r - 1]$ to $A[p]$, then return $\text{PARTITION}(A, p, r - 1)$
 - Definitely more swaps (so more overhead), but still correct (and same asymptotic notation) with easier derivation of the recurrence relation
 - $T(n) = T(n - 1) + \Theta(1) \rightarrow T(n) = \Theta(n)$
- Base case: If $p = r$, return r .

29

Given the input array $A = [13, 19, 9, 5, 12]$, what is the correct array when the textbook's $\text{PARTITION}()$ pseudocode is applied to the array A ?

- a) $[19, 13, 12, 9, 5]$
- b) $[13, 9, 12, 5, 19]$
- c) $[9, 5, 12, 19, 13]$
- d) $[5, 9, 12, 13, 19]$

30

Quicksort Analysis

QUICKSORT(A, p, r)

```

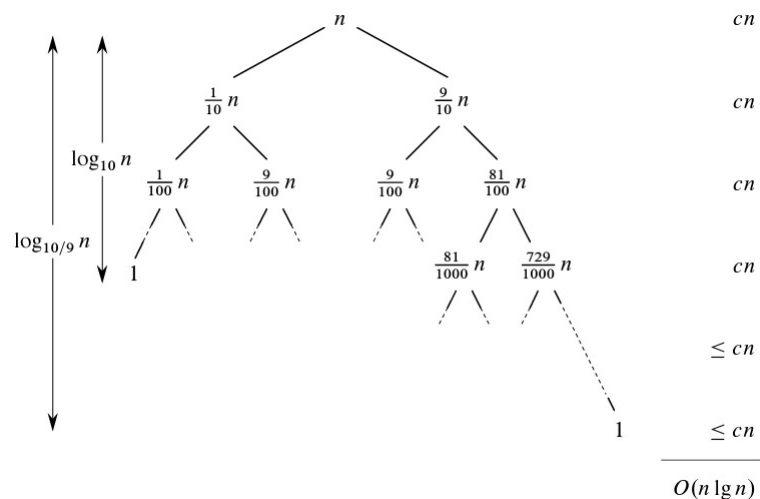
1  if  $p < r$ 
2     $q = \text{PARTITION}(A, p, r)$ 
3    QUICKSORT( $A, p, q - 1$ )
4    QUICKSORT( $A, q + 1, r$ )
    
```

- From the QUICKSORT() pseudocode,
 $T_{\text{qsort}}(n) = T_{\text{partition}}(n) + T_{\text{qsort}}(n_1) + T_{\text{qsort}}(n_2)$, where $n_1 + n_2 + 1 = n$ and $n_1 \geq 0, n_2 \geq 0$
- It's obvious that $T_{\text{partition}}(n) = \Theta(n)$.
- So, $T(n) = T(n_1) + T(n_2) + \Theta(n)$ where $n_1 + n_2 + 1 = n$
- Worst case: $n_1 = 0$ or $n_2 = 0$ all the time (bad split/partition) \rightarrow
 - $T(n) = T(n - 1) + \Theta(n) \rightarrow T(n) = \Theta(n^2)$
 - When would this happen? What's the insertion/bubble sort performance in that case?
- Best case: $n_1 \cong n_2$ as much as possible (even split) \rightarrow
 - $T(n) = 2T\left(\frac{n}{2}\right) + \Theta(n) \rightarrow T(n) = \Theta(n \lg n)$

31

Balanced Splits, Even Skewed (CLRS Fig. 7.4)

- 9-to-1 splits all the time
- In fact, doesn't matter what x & y in x -to- y splits, as long as x & y are fixed.



32

Randomized Quicksort (CLRS Section 7.3)

- To avoid worst case as much as possible,
 - Pick the pivot from a random index, not from a fixed one at the end.
 - Still rely on the original PARTITION() after swapping the randomly picked pivot with the original fixed pivot.

RANDOMIZED-PARTITION(A, p, r)

```
1  $i = \text{RANDOM}(p, r)$   
2 exchange  $A[r]$  with  $A[i]$   
3 return PARTITION( $A, p, r$ )
```

RANDOMIZED-QUICKSORT(A, p, r)

```
1 if  $p < r$   
2    $q = \text{RANDOMIZED-PARTITION}(A, p, r)$   
3   RANDOMIZED-QUICKSORT( $A, p, q - 1$ )  
4   RANDOMIZED-QUICKSORT( $A, q + 1, r$ )
```

33

Formal Proofs of RANDOMIZED-QUICKSORT Time Complexities (CLRS Section 7.4)

- Lots of algebraic derivations. We won't focus on those.
- Also random probabilistic analysis and derivations for randomized case. We won't focus on those either.
- Just read through CLRS Section 7.4 and see how they go.

34

Brief Summary

- Worst-case: $T(n) = \max_{0 \leq q \leq n-1} (T(q) + T(n - q - 1)) + \Theta(n)$
 - We can show $T(n) \leq cn^2$ for some c and large enough n , showing $T(n) = O(n^2)$ (This is done in textbook)
 - Can also show $T(n) \geq cn^2$ for some c and large enough n , showing $T(n) = \Omega(n^2)$ (Exercise 7.4-1)
 - Therefore, worst-case $T(n) = \Theta(n^2)$.
- Average-case (expected running time) of RANDOMIZED-QUICKSORT()
 - Probabilities of possible cases, number of comparisons becoming random variable, derive the expected average of the random variable.
 - $E[X] = \dots = O(n \log n)$
- In practice, Quick Sort is usually faster than Merge Sort.

35

Median Finding Algorithm

No Need To Sort Entire Array

36

Medians and Order Statistics

- Given a set A of n elements,
 - Minimum (first in the ordered sequence), maximum (last), median (mid)
 - If n is even, there could be 2 medians. For simplicity, we mean the lower median.
- General i -th order statistic: The i -th smallest element of A
 - Minimum: A 's 1st order statistic, maximum: A 's n -th order statistic
 - Median: A 's $\lfloor (n + 1)/2 \rfloor$ -th order statistic
- Algorithm to find the i -th order statistic of A for any given A and i
 - Simple (Naïve): Sort A , return $A[i]$: $O(n \lg n)$
 - Do we really need to sort the entire array? Aren't we doing more than necessary?
 - Finding minimum: Just by scanning the entire array, you can find i -th element in $O(n)$ (In fact, this is the optimal solution – why? Read the textbook).
 - But what about finding i -th order (general algorithm) ?

37

Average Linear Time Selection Algorithm

- CLRS 9.2 RANDOMIZED-SELECT() (pp. 216)

```
RANDOMIZED-SELECT( $A, p, r, i$ )
1  if  $p == r$ 
2    return  $A[p]$ 
3   $q = \text{RANDOMIZED-PARTITION}(A, p, r)$ 
4   $k = q - p + 1$ 
5  if  $i == k$            // the pivot value is the answer
6    return  $A[q]$ 
7  elseif  $i < k$ 
8    return RANDOMIZED-SELECT( $A, p, q - 1, i$ )
9  else return RANDOMIZED-SELECT( $A, q + 1, r, i - k$ )
```

38

Time Complexity Analysis of RAND-SEL

- $T_{select}(n) = T_{partition}(n) + T_{select}(n')$, where $0 \leq n' < n$
- Worst case: $T(n) = T(n-1) + O(n)$
 - $\Theta(n^2)$, just like quicksort
- Average (expected) case: $T(n) = T(n/k) + O(n)$
 - Requires random variable analysis, just like in quicksort
 - Assume probabilities, find expected running time, show it's at most $O(n)$
 - Details in CLRS 9.2
 - We don't need to do this all the time.
 - I'd say intuition is more important than formal proof.
 - Think about balanced splits-case (e.g., 2:3), and derive running time, confirm it's $O(n)$.
- Can we achieve $O(n)$ in worst case as well?
 - Surprisingly, yes.
 - Time complexity analysis of SELECT() is more interesting and involved.

39

Selection algorithm in worst-case linear time

(Main Idea)

- Divide the input array into $n/5$ groups.
- Find the median for each group (note each group consists of at most 5 elements) – $O(1)$
- Find the median of $n/5$ medians (This is done recursively) – $T(n/5)$
- $3n/10 - 6$ ($= 3(n/10 - 2)$) is already less than the median of the medians.
- If the median of the medians is the median, return.
- Otherwise, recursively call among $7n/10 + 6$ elements. – $T(7n/10 + 6)$
- Overall running time:
 - $T(n) = T(n/5) + T(7n/10 + 6) + O(n) \Rightarrow O(n)$

40