### Divide and Conquer 2

Lecture 3

Timothy Kim GWU CSCI 6212

#### Practice Question

 Given the following recurrence, find a f(n) such that T(n) becomes O(log n)

$$T(n) = 2T(\frac{n}{2}) + f(n)$$

#### Quick Review

- Binary Search Tree
  - Operations and their running time
- Merge Sort
  - Analysis via recurrence tree
- Master Method
  - · Proof and implication it has on divide and conquer algorithm design.

# Today

- Quick Sort
  - Description
  - Proof of correctness
  - Analysis
  - Random Pivot Selection
  - Deterministic Pivot Selection
- Matrix Multiplication (if time provides)

## Sorting Problem

- Input: array of n numbers
- Output: array of same n numbers, placed in increasing order
- · Assumption: all numbers are unique

#### Quick Sort

- Used a lot in practice
- O(n log n) on average
- In place sort (minimal memory needed)
- My favorite algorithm (beautiful analysis)

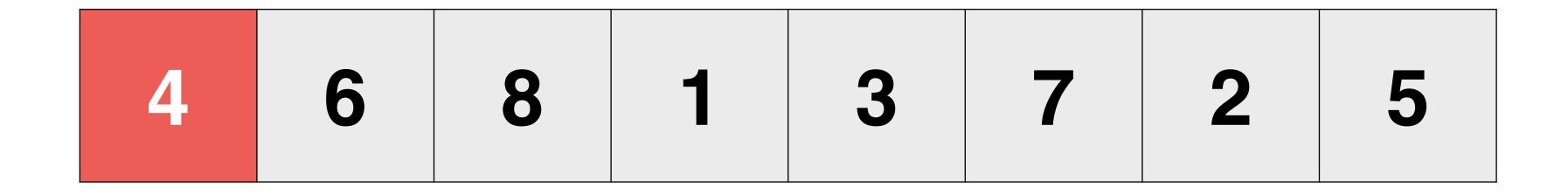
# Partitioning

Pick a pivot

4	6	8	1	3	7	2	5
---	---	---	---	---	---	---	---

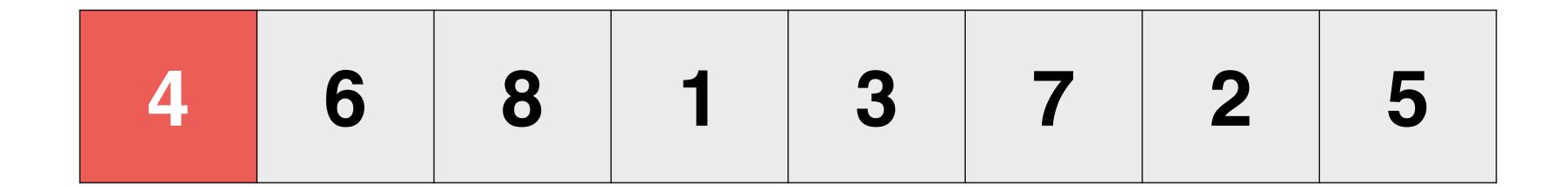
# Partitioning

Pick a pivot

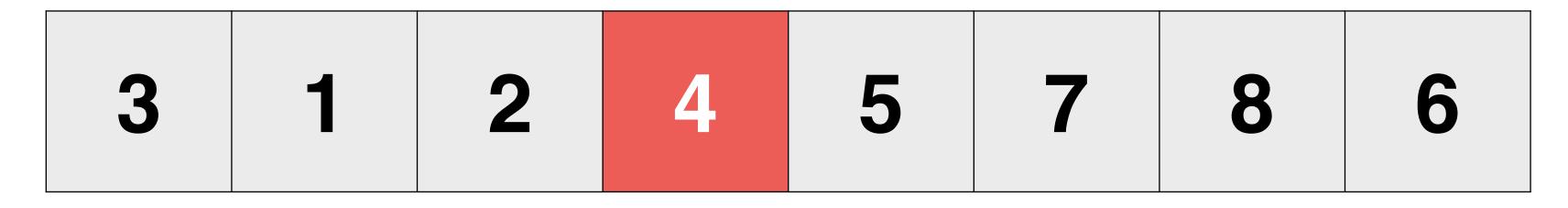


## Partitioning

Pick a pivot



Rearrange



less than four

greater than four

#### Partition Subroutine

- Places pivot in this correct location
- Running time?
- Reduces the problem size (divide and conquer)

## Partition (naive)

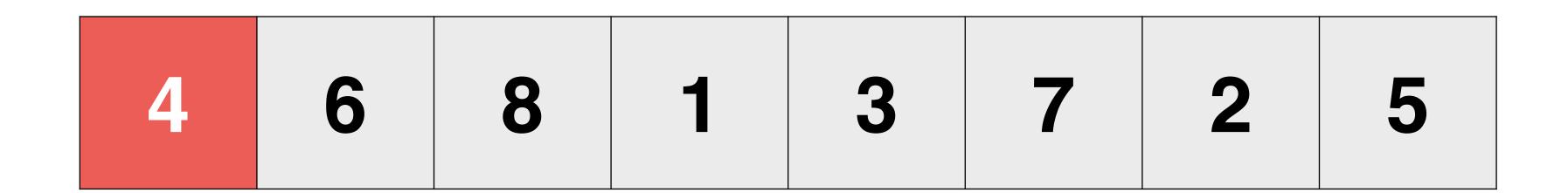
```
1. Partition(A, pivot):

    Running Time?

   L = \lceil \rceil
                                       • O(n)
G = \begin{bmatrix} 1 \end{bmatrix}
4. for x in A:
                                     Space?
5. if x < pivot:
                                       • O(n)
          L = L + X
7. else if x > pivot:
      G = G + X
8.
    return L + pivot + G
```

# Partition (in place) Example

1. Partition(A, begin, end): • Assumption: pivot is in the first position



## Partition (in place)

```
    Partition(A, begin, end):
    pivotIndex = ChoosePivot(A, begin, end)
    pivot = A[pivotIndex]
    swap A[begin] with A[pivotIndex]
    //Rest for Homework
```

## Quick Sort (naive)

```
    QuickSort(A):
    p = ChoosePivot(A)
    S1, S2 = Partition(A, p)
    return QuickSort(S1) + p + QuickSort(S2)
```

# Quick Sort (in place)

```
    QuickSort(A, begin, end):
    if begin < end:</li>
    p = Partition(A, begin, end)
    QuickSort(A, begin, p - 1)
    QuickSort(A, p + 1, end)
```

// Call
 QuickSort(A, 1, |A|)

#### Quick Sort Proof of Correctness

- QuickSort correctly sorts all possible input array of length n
  - (No matter how pivot is chosen)
- Proof by induction

### Proof by Induction

- · Given P(n) (a statement parameterized by n)
- Claim: P(n) holds true for all possible value of n > 0
  - 1. Base Case: First prove P(1) is true
  - 2. Inductive Hypothesis:

Assume P(n) is true for all values of n leading up to k  $P(1) \dots P(k)$ 

3. Inductive Case: Prove P(k + 1) is true.

### Proof by Induction of QS

- P(n) = QuickSort correctly sorts all possible input array of length n
- Claim: P(n) is true for all values of n > 0
- Base Case:
  - P(1) = QuickSort correctly sorts input array of length 1
- Inductive Hypothesis:
  - Assume  $P(1) \sim P(k)$  holds true
- Inductive Case:
  - Prove P(k + 1) holds true

#### Inductive Case

- Prove P(k + 1) is true:
  - Note, QuickSort partitions input array around a pivot.

$$S_1 = \{ \text{ elements} 
$$S_2 = \{ \text{ elements} > p \}$$$$

- Pivot is in the right place
- $k_1 = \text{size of } S_1 < k + 1$ 
  - Thus  $P(k_1)$  must be true, that is  $S_1$  is sorted (by Inductive Hypothesis)
- $k_2 = size of S_2 < k + 1$ 
  - Thus P(k<sub>2</sub>) must be true, that is S<sub>2</sub> is sorted (by Inductive Hypothesis)
- · QED

## Quick Sort Analysis

```
    QS(A):
    p = ChoosePivot(A)
    S1, S2 = Partition(A, p)
    return QS(S1) + p + QS(S2)

T(n)

??

O(n)

T(??) + T(??)
```

- The recurrence is T(n) = T(l) + T(m) + O(n) + f(n)
- Master Method doesn't work!
- Size of S1 and S2 depends on the ChoosePivot subroutine
- Let's think about best case and worst case

## Worst Case pivot selection

- Recall QS's running time: T(n) = T(l) + T(m) + O(n) + f(n)
- What if every time we selected a pivot, the division looked like this:

$$l = 0$$
  
 $m = n-1$   
 $T(m) = (n-1) + (n-2) + ... + 1 = \frac{n(n+1)}{2} - n = O(n^2)$ 

- · That means, pivot was selected in sorted order.
- Finally  $T(n) = O(n^2)$
- Can you think of a pivot selection algorithm that produces this case?

#### Naive Pivot Selection

```
    ChoosePivot(A):
    return A[1]
```

### Best case pivot selection

- Recall QS's running time: T(n) = T(l) + T(m) + O(n) + f(n)
- What if l=m and f(n)=O(n) ?
- Then our running time becomes  $T(n) = 2T(\frac{n}{2}) + O(n)$
- Using master method:  $T(n) = 2T(\frac{n}{2}) + O(n) = O(n \log n)$
- But how?

#### Random Pivot Selection

```
    ChoosePivot(A):
    r = random(1, |A|)
    return A[r]
```

#### Random QuickSort Analysis

#### · Claim:

 For all input array of length n, the average running time of QuickSort with random pivot selection is O(n log n)

## Probability Ideas

- Sample Spaces
- Events
- Random Variables
- Expected Values
- Linearity of Expectation

# Sample Space

- $\Omega$  = All possible outcomes of a randomness (often finite)
- Given  $i \in \Omega$ ,  $P(i) \ge 0$
- Constraint  $\sum_{i \in \Omega} P(i) = 1$
- Example: Rolling 2 dice
  - $\Omega = \{ (1,1), (2,1), (3,1), \dots (5,6), (6,6) \}$
  - Where P(i) = 1/36 for all  $i \in \Omega$

#### Event

- Event is a subset of sample space.
  - $S \subseteq \Omega$

$$P(S) = \sum_{i \in S} P(i) = 1$$

- Example: set of outcomes for which the sum of two dice is 7
  - $S = \{(1,6), (2,5), (3,4), (4,3), (5,2), (6,1)\}$
  - P(S) = ?

#### Random Variables

- Random variable is a function that maps sample space to a real-value.
  - ullet  $X:\Omega o\mathbb{R}$
- Example:
  - Sum of two dice

## Expected Value

- Let X be a random variable
- Expected value or expectation of X is just a average value of X

$$E[X] = \sum_{i \in \Omega} X(i) \cdot P(i)$$

- Example: X = Sum of two dice
  - What is E[X]?

## Linearity of Expectation

#### • Claim:

Given  $X_1, \ldots, X_n$  random variables over  $\Omega$ :

$$E[\sum_{i=1}^{n} X_i] = \sum_{i=1}^{n} E[X_i]$$

• Dice Example: if  $X_1$ ,  $X_2$  = two dice

$$E[X_i] = 1/6 (1+2+3+4+5+6) = 3.5$$
  
 $E[X_1+X_2] = E[X_1]+E[X_2] = 3.5+3.5=7$ 

## Proof Setup

- Fix an input array A of length n
- Pivot Sequence  $\sigma$  = list of random pivots chosen by running QS with A
- Sample Space:  $\Omega$  = set of all possible pivot sequences
- Random Variable:

$$\sigma \in \Omega$$

 $C(\sigma)$  = number of comparison made by QS given  $\sigma$ 

· Why do we care about number of comparisons?

#### Lemma

- Running time of QuickSort dominated by comparisons
- More formally

$$\exists c \mid \forall \sigma \in \Omega, T(\sigma) \leq c \cdot C(\sigma)$$

 Most of the work is done during partition, all it does is just compares elements and swaps.

#### New Goal

 Average Running time of Randomized QuickSort is determined by expected value of the random variable C, number of comparisons done by QuickSort.

$$T(n) = E[C] = O(n \log n)$$

But what is the value of C?

## Random Variable Decomposition

We'll look at a smaller random variable

```
A = \text{fixed input array}
\Omega = \text{set of all possible pivot sequence}
\sigma \in \Omega
z_i = i^{\text{th}} \text{ smallest element of } A
X_{ij}(\sigma) = \text{number of times } z_i, z_j \text{ get compared } \mid i < j
```

 Given any two element in A, how many times can they be compared to each other?

$$X_{ij}(\sigma) = [0,1] = \text{indicator random variable}$$

## Random Variable Decomposition

$$C(\sigma)$$
 = number of comparisons between input elements  $X_{ij}(\sigma)$  = number of comparisons between  $z_i$  and  $z_j$ 

$$\forall \sigma, C(\sigma) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij}(\sigma)$$

$$E[C] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} E[X_{ij}]$$

note: 
$$E[X_{ij}] = 0 \cdot P(X_{ij} = 0) + 1 \cdot P(X_{ij} = 1) = P(X_{ij} = 1)$$

$$E[C] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} P(X_{ij} = 1, z_i \text{ and } z_j \text{ get compared})$$

## What is the probability?

claim: 
$$\forall i < j, P(X_{ij} = 1) = \frac{2}{j - i + 1}$$

fix  $z_i, z_j \mid i < j$ consider  $S = \{z_i, z_{i+1}, ..., z_{j-1}, z_j\}$ 

- If pivot choice is not in S, then S get passed down the recursive call
- If pivot is chosen from S,
  - 1. If  $z_i$  or  $z_j$  gets chosen, then  $z_i$  and  $z_j$  gets compared
  - 2. If  $z_{i+1}, \ldots$ , or  $z_{j-1}$  gets chosen, then  $z_i$  and  $z_j$  never gets compared
- Since pivots are chosen uniformly at random, all elements in S is equally likely

$$P(X_{ij} = 1) = \frac{\text{choices that lead to } z_i \text{ and } z_j \text{ gets compared}}{\text{total number of choices}} = \frac{2}{j - i + 1}$$

#### New Goal

$$E[C] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} P(X_{ij} = 1, z_i \text{ and } z_j \text{ get compared})$$

$$P(X_{ij} = 1) = \frac{2}{j - i + 1}$$

$$E[C] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j-i+1} \stackrel{?}{=} O(n \log n)$$

#### Proof

$$E[C] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j-i+1} \stackrel{?}{=} O(n \log n)$$

$$= 2 \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{1}{j-i+1} \le 2 \cdot n \cdot \sum_{k=2}^{n} \frac{1}{k}$$

For each fixed i

$$\sum_{i=i+1}^{n} \frac{1}{j-i+1} \le \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{n}$$

# Let's finish the proof

$$E[C] \le 2n \sum_{k=2}^{n} \frac{1}{k}$$

$$\leq 2n \int_{2}^{n} \frac{1}{x} dx$$

$$\leq 2n \int_{0}^{\pi} \frac{1}{x} dx = 2n \cdot \ln x |_{1}^{n} = 2n \cdot (\ln n - \ln 1) = O(n \log n)$$

### Deterministic QuickSort

- ChoosePivot subroutine must be updated without random call.
- What is the best pivot selection?
- How can we pick the median?

## Naive Median Selection

```
    Median(A):
    S = sort(A)
    middle = |S|/2
    return S[middle]
```

What's wrong with this?

## Randomized Selection

1. Select(A, n, i): · Homework

#### Deterministic Selection

```
1. Select(A, n, i):
2. split A in to group of 5
3. sort each group
4. C = array of 3rd element in each group (median array)
5. p = Select(C, n/5, n/10) (median of median)
6. Partition(A, p)
7. j = location of p after partition
8. if (i = j) return p
     if (j < i) return Select(A[1...j-1], j-1, i)
9.
               return Select(A[j+1, n], n-j, i-j)
    else
10.
```

### Running Time of Deterministic Selection

```
1. Select(A, n, i):
     split A in to group of 5
                                                 2. O(n)
3. sort each group
                                                  3. O(n)
4. C = array of 3rd element in each group
                                                 4. O(n)
5. p = Select(C, n/5, n/10)
                                                  5. T(n/5)
                                                  6. O(n)
6.
     Partition(A, p)
                                                 7. 0(1)
     j = location of p after partition
     if (i = j) return p
8.
     if (j < i) return Select(first section)</pre>
                                                 9.0(?)
               return Select(second section)
     else
```

# Running time of Selection

$$T(n) = O(n) + T(n/5) + T(x)$$

- Claim:  $x \le n \frac{7}{10}$
- In other words, our median of median will cause a partition of our array to be at best 30:70 split.
- Visual proof of the claim

## Visual Proof Idea

## Finish the proof

$$T(n) \le O(n) + T(\frac{n}{5}) + T(\frac{7n}{10}) \stackrel{?}{=} O(n)$$

- Let k be a constant > 1, then  $T(n) \le kn$
- Proof by induction
  - Base Case: T(1) = 1 = O(n)
  - Inductive Hypothesis:  $T(k) \le kn \ \forall \ k < n$
  - Inductive Case:  $T(n) \le cn + T(\frac{n}{5}) + T(\frac{7n}{10})$

$$\leq cn + k\frac{n}{5} + k\frac{7n}{10}$$

$$= n(c + \frac{9k}{10})$$

$$=kn$$