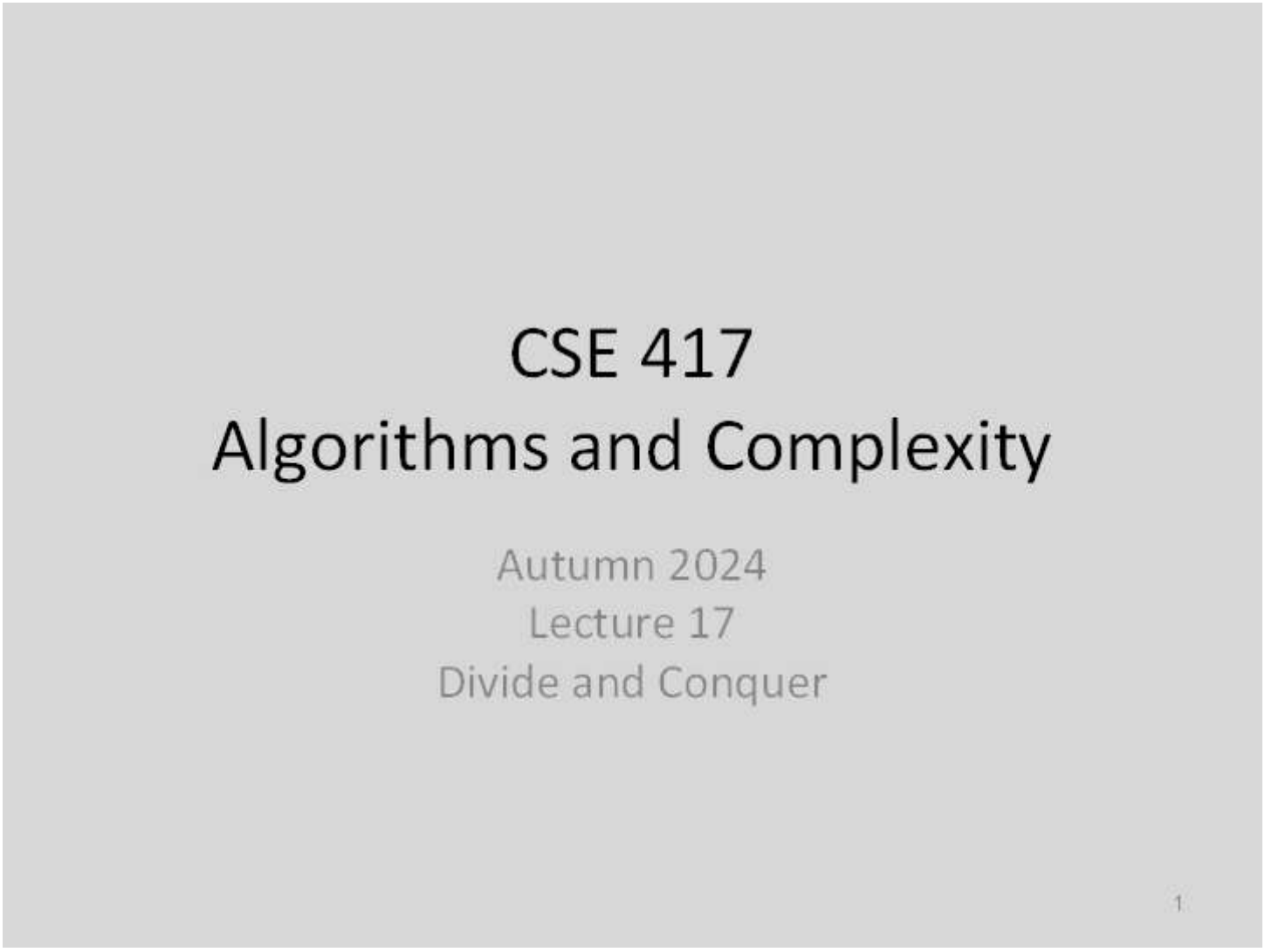
Lecture17


CSE 417
Algorithms and Complexity
Autumn 2024
Lecture 17
Divide and Conquer
1

Announcements

- HW 6 available, due Friday
- Schedule
  - Today – sorting and divide and conquer
  - Wednesday – more divide and conquer
  - Friday – dynamic programming

2

Divide and Conquer for Sorting

- Merge Sort

```
Array MSort(Array a, int n){
   if (n ≤ 1)  return a;
   return Merge(MSort(a[0 .. n/2], n/2),  MSort(a[n/2+1 .. n-1], n/2);
}
```

- $T(n) = 2T(n/2) + n; T(1) = 1;$

3

Quicksort [Tony Hoare, 1959]

QuickSort(S):

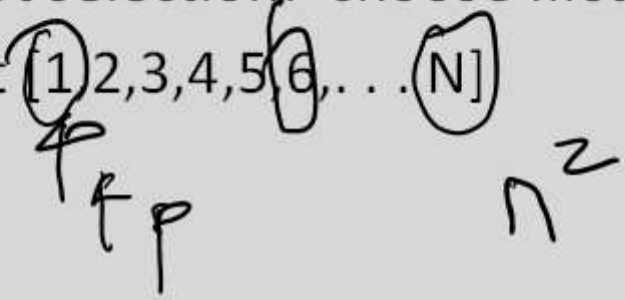
1. Pick an element v in **S**. This is the ***pivot*** value.
2. Partition **S**-{v} into two disjoint subsets, $\mathbf{S}_1$ and $\mathbf{S}_2$ such that:
   - elements in $\mathbf{S}_1$ are all $< v$
   - elements in $\mathbf{S}_2$ are all $> v$
3. Return concatenation of QuickSort($\mathbf{S}_1$), v, QuickSort($\mathbf{S}_2$)

Recursion ends if Quicksort( ) receives an array of length 0 or 1.

4

Quicksort – worst case

- Pivot selection: choose first element
- Sort [1,2,3,4,5,6,. . .,N]



5

Quicksort - pragmatics

- Pivot selection rules
  - Median of first, middle, and last
  - Choose random element
- In place implementation
- Algorithm engineering for partitioning
- Recursion cutoff for small problems

6

Average case analysis for Quicksort

- All inputs equally likely
  - Or random elements used for pivot
  - Or input is randomly shuffled
- QS(n) = average number of comparisons for Quicksort on input of size n.

7

Building a recurrence

Pivot chosen at random. The chance of having i elements less than the pivot is 1/n.

$$T(n) = (n-1) + \frac{1}{n}\sum_{i=0}^{n-1}(T(i) + T(n-i-1)).$$

$$T(n) = (n-1) + \frac{2}{n}\sum_{i=1}^{n-1} T(i)$$

Solution: $T(N) \approx 2\, n \ln n$

8

Computing the Median

- Given n numbers, find the number of rank n/2
- One approach is sorting
  - Sort the elements, and choose the middle one
  - Can you do better?

9

Problem generalization

- *Selection*, given n numbers and an integer k, find the k-th largest

10

Select(A, k)

```
Select(A, k){
        Choose element x from A
        S1 = {y in A | y < x}
        S2 = {y in A | y > x}
        S3 = {y in A | y = x}
        if (|S2| >= k)
                return Select(S2, k)
        else if (|S2| + |S3| >= k)
                return x
        else
                return Select(S1, k - |S2| - |S3|)
}
```

11

Randomized Selection

- Choose the element at random
- Analysis can show that the algorithm has expected run time O(n)

12

$|x|<1 \quad \sum_{i=0}^{\infty} x^i = \frac{1}{1-x}$

Deterministic Selection

$\frac{1}{1-\frac{3}{4}}$

- What is the run time of select if we can guarantee that choose finds an x such that $|S_1| < 3n/4$ and $|S_2| < 3n/4$ in O(n) time

$T(n) = T(\frac{3}{4}n) + n = T(\frac{1}{16}n) + \frac{3}{4}n + n$

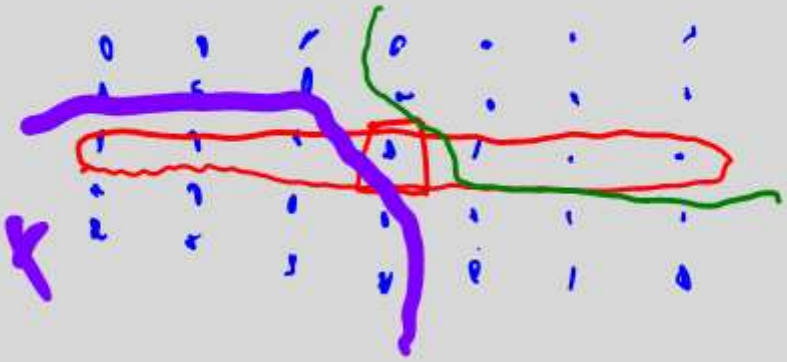
$= \sum_{i=1}^{k} \left(\frac{3}{4}\right)^i \leq n \sum_{i=1}^{\infty} \left(\frac{3}{4}\right)^i =$

13

BFPRT Algorithm

- A very clever choose algorithm . . .

Split into n/5 sets of size 5
M be the set of medians of these sets
Let x be the median of M



BFPRT runtime

$|S_1| < 3n/4$, $|S_2| < 3n/4$

Split into n/5 sets of size 5 $O(n)$

M be the set of medians of these sets $O(n)$

x be the median of M $T(\frac{n}{5})$

Construct S_1 and S_2 $O(n)$

Recursive call in S_1 or S_2 $T(\frac{3}{4}n)$

$T(n) \leq T(\frac{3}{4}n) + T(\frac{n}{5}) + cn$

15

BFPRT Recurrence

$$T(n) \leq T(3n/4) + T(n/5) + c\,n$$

Suppose $T(k) \leq 20ck$ for $k < n$

$$T(n) \leq 20c\frac{3n}{4} + 20c\frac{n}{5} + cn$$

$$= 15cn + 4cn + cn$$

$$= 20cn$$

Prove that $T(n) \leq 20\,c\,n$

16

Inversion Problem

- Let $a_1, \ldots a_n$ be a permutation of 1 . . n
- (a_i, a_j) is an inversion if $i < j$ and $a_i > a_j$

4, 6, 1, 7, 3, 2, 5

- Problem: given a permutation, count the number of inversions
- This can be done easily in $O(n^2)$ time
  - Can we do better?

17

Application

- Counting inversions can be use to measure how close ranked preferences are
  - People rank 20 movies, based on their rankings you cluster people who like that same type of movie

18

Counting Inversions

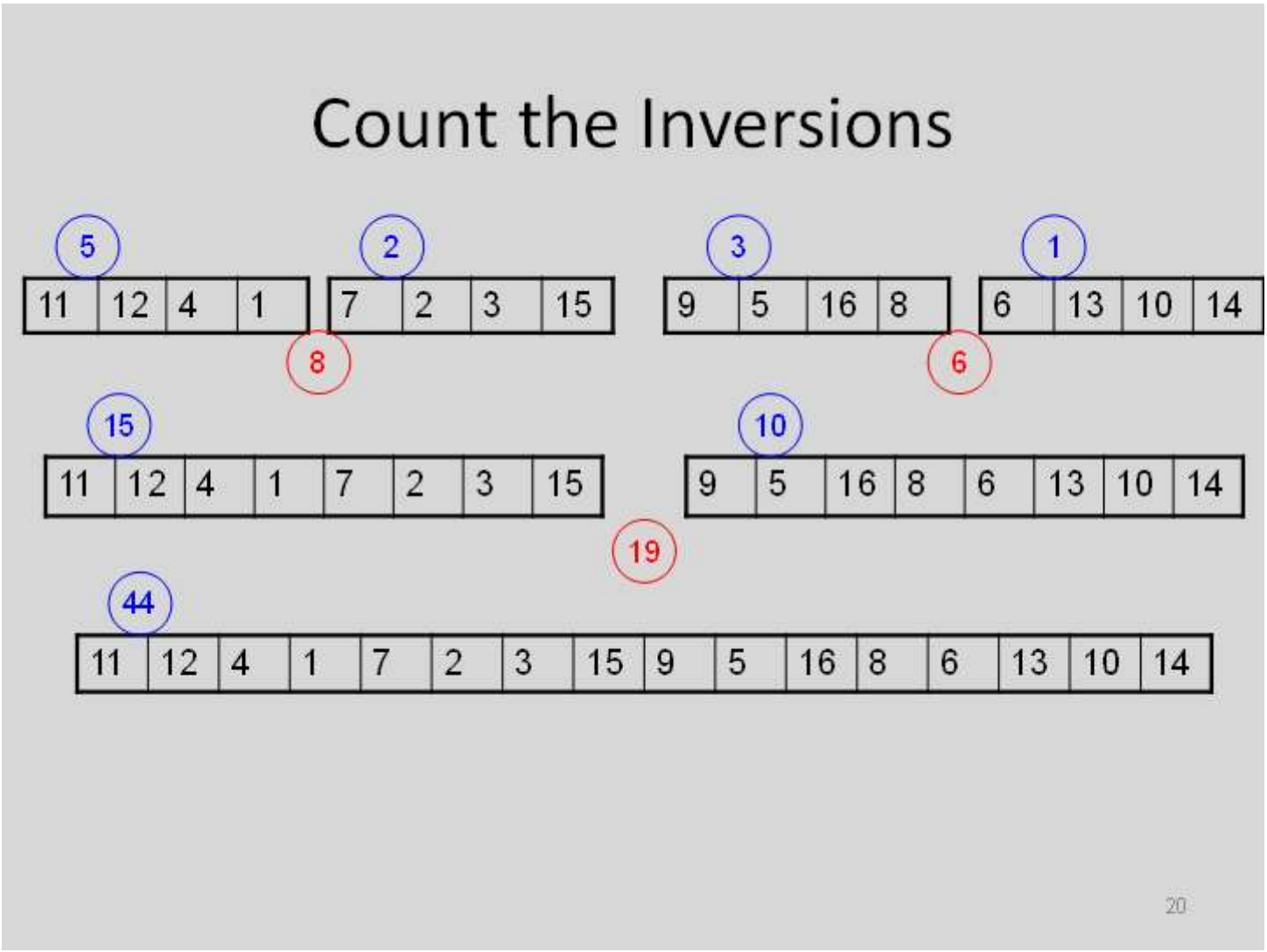
11	12	4	1	7	2	3	15	9	5	16	8	6	13	10	14

Count inversions on lower half

Count inversions on upper half

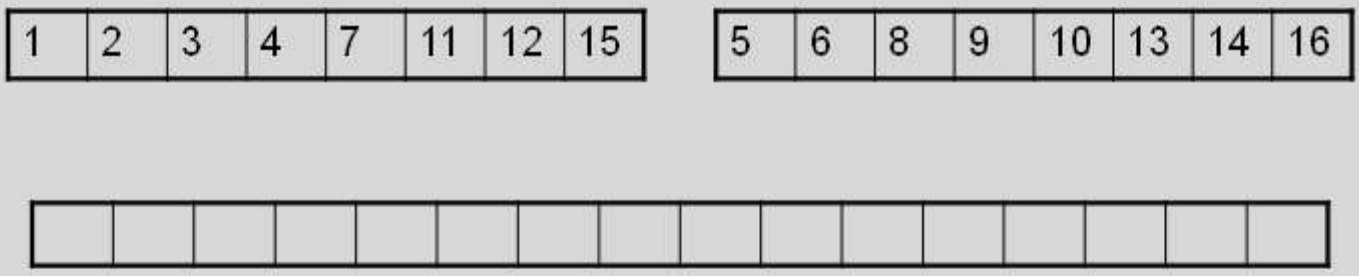
Count the inversions between the halves

19


Count the Inversions
5
11 12 4 1
2
7 2 3 15
8
3
9 5 16 8
1
6 13 10 14
6
15
11 12 4 1 7 2 3 15
10
9 5 16 8 6 13 10 14
19
44
11 12 4 1 7 2 3 15 9 5 16 8 6 13 10 14
20

Problem – how do we count inversions between sub problems in O(n) time?

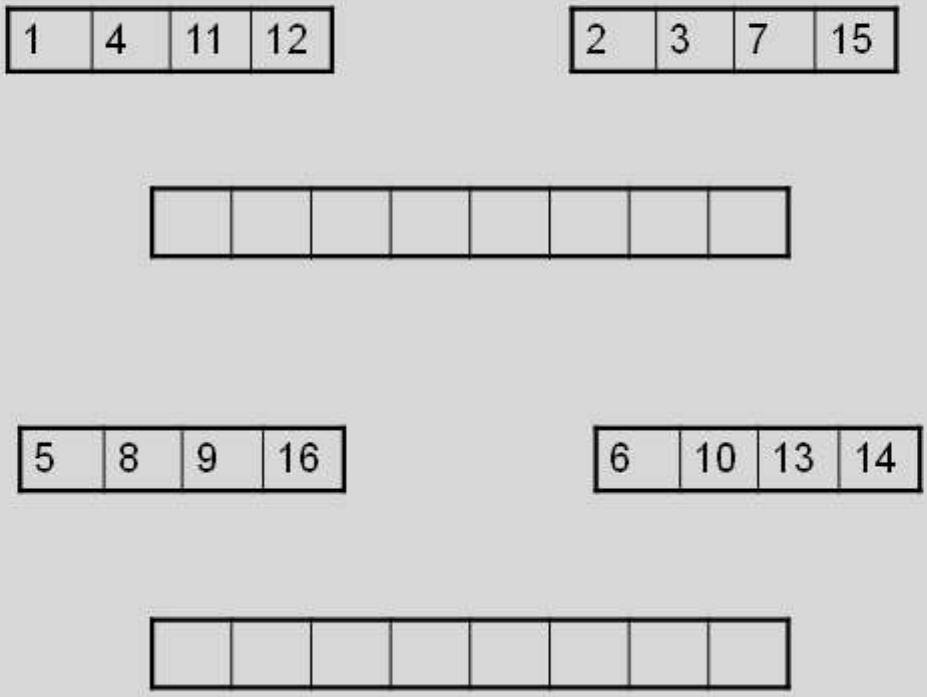
- Solution – Count inversions while merging



Standard merge algorithm – add to inversion count when an element is moved from the upper array to the solution

21

Use the merge algorithm to count inversions

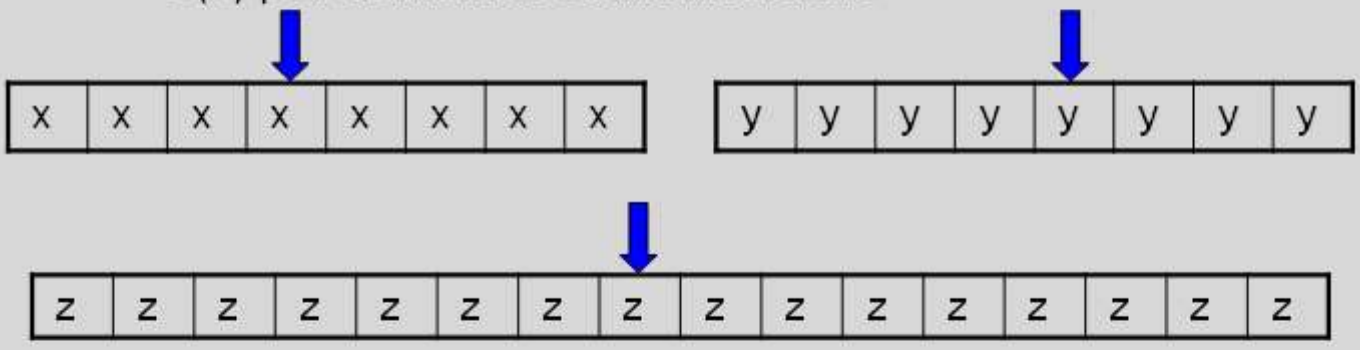


Indicate the number of inversions for each element detected when merging

22

Inversions

- Counting inversions between two sorted lists
  - O(1) per element to count inversions



- Algorithm summary
  - Satisfies the "Standard recurrence"
  - $T(n) = 2\,T(n/2) + cn$

23