

# Chapter 7

## Recurrences, Closest Pair and Selection

OLD CS 473: Fundamental Algorithms, Spring 2015  
February 10, 2015

### 7.1 Recurrences

#### 7.1.0.1 Solving Recurrences

Two general methods:

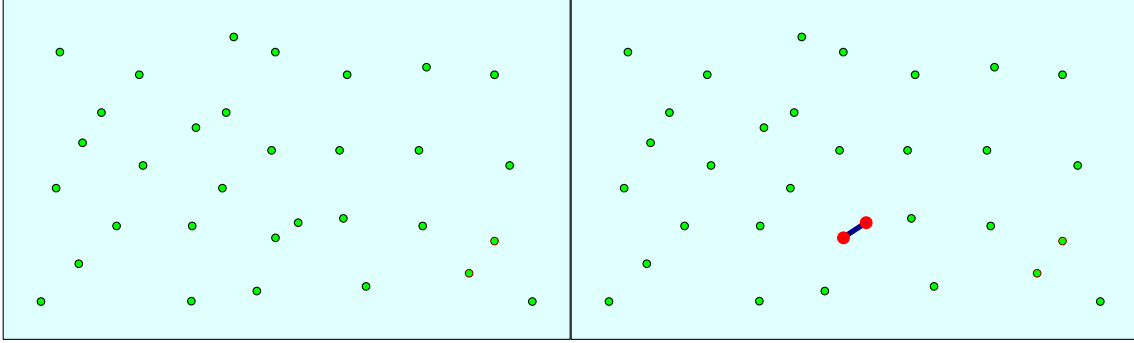
- (A) Recursion tree method: need to do sums
  - (A) elementary methods, geometric series
  - (B) integration
- (B) Guess and Verify
  - (A) guessing involves intuition, experience and trial & error
  - (B) verification is via induction

#### 7.1.0.2 Recurrence: Example I

- (A) Consider  $T(n) = 2T(n/2) + n/\lg n$ .
- (B) Construct recursion tree, and observe pattern.
- (C)  $i$ th level has  $n_i = 2^i$  nodes.
- (D) problem size at node of level  $i$  is  $n/2^i$ .
- (E) work at node of level  $i$  is  $w_i = \frac{n}{2^i} / \lg \frac{n}{2^i}$ .
- (F) Total work at  $i$ th level is  $n_i \cdot w_i = 2^i \cdot \frac{n}{2^i} / \lg \frac{n}{2^i} = n / \lg \frac{n}{2^i}$
- (G) Summing over all levels  $T(n) = \sum_{i=0}^{\lg n-1} n_i \cdot w_i = \sum_{i=0}^{\lg n-1} \frac{n}{\lg \frac{n}{2^i}} = n \sum_{i=0}^{\lg n-1} \frac{1}{\lg n - i} = n \sum_{j=1}^{\lg n} \frac{1}{j} = nH_{\lg n} = \Theta(n \log \log n)$

#### 7.1.0.3 Recurrence: Example II

- (A) Consider...



- (B) What is the depth of recursion?  $\sqrt{n}, \sqrt{\sqrt{n}}, \sqrt{\sqrt{\sqrt{n}}}, \dots, O(1)$ .
- (C) Number of levels:  $n^{2^{-L}} = 2$  means  $L = \log \log n$ .
- (D) Number of children at each level is 1, work at each node is 1
- (E) Thus,  $T(n) = \sum_{i=0}^L 1 = \Theta(L) = \Theta(\log \log n)$ .

#### 7.1.0.4 Recurrence: Example III

- (A) Consider  $T(n) = \sqrt{n}T(\sqrt{n}) + n$ .
- (B) Using recursion trees: number of levels  $L = \log \log n$
- (C) Work at each level? Root is  $n$ , next level is  $\sqrt{n} \times \sqrt{n} = n$ , so on. Can check that each level is  $n$ .
- (D) Thus,  $T(n) = \Theta(n \log \log n)$

#### 7.1.0.5 Recurrence: Example IV

- (A) Consider  $T(n) = T(n/4) + T(3n/4) + n$ .
- (B) Using recursion tree, we observe the tree has leaves at different levels (a *lop-sided* tree).
- (C) Total work in any level is at most  $n$ . Total work in any level without leaves is exactly  $n$ .
- (D) Highest leaf is at level  $\log_4 n$  and lowest leaf is at level  $\log_{4/3} n$
- (E) Thus,  $n \log_4 n \leq T(n) \leq n \log_{4/3} n$ , which means  $T(n) = \Theta(n \log n)$

## 7.2 Closest Pair

### 7.2.1 The Problem

#### 7.2.1.1 Closest Pair - the problem

**Input** Given a set  $S$  of  $n$  points on the plane

**Goal** Find  $p, q \in S$  such that  $d(p, q)$  is minimum

### 7.2.1.2 Applications

- (A) Basic primitive used in graphics, vision, molecular modelling
- (B) Ideas used in solving nearest neighbor, Voronoi diagrams, Euclidean MST

## 7.2.2 Algorithmic Solution

### 7.2.2.1 Algorithm: Brute Force

- (A) Compute distance between every pair of points and find minimum.
- (B) Takes  $O(n^2)$  time.
- (C) Can we do better?

## 7.2.3 Special Case

### 7.2.3.1 Closest Pair: 1-d case

**Input** Given a set  $S$  of  $n$  points on a line

**Goal** Find  $p, q \in S$  such that  $d(p, q)$  is minimum

Algorithm

- (A) Sort points based on coordinate
- (B) Compute the distance between successive points, keeping track of the closest pair.

Running time  $O(n \log n)$

- (A) Can we do this in better running time?
- (B) Can reduce Distinct Elements Problem (see lecture 1) to this problem in  $O(n)$  time. Do you see how?

### 7.2.3.2 Generalizing 1-d case

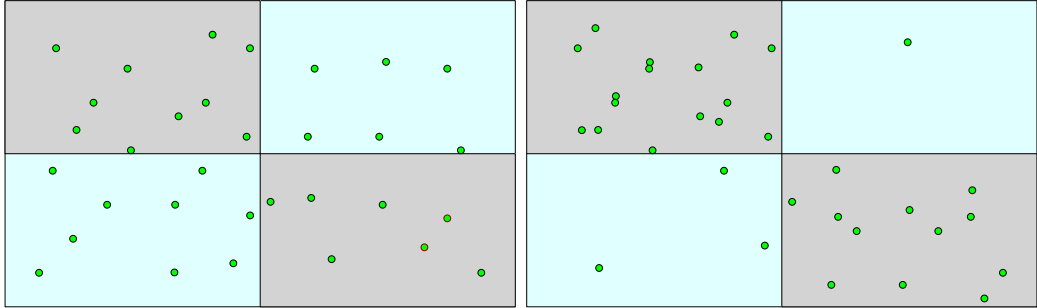
- (A) Can we generalize 1-d algorithm to 2-d?
- (B) Sort according to  $x$  or  $y$ -coordinate??
- (C) No easy generalization.

## 7.2.4 Divide and Conquer

### 7.2.4.1 First Attempt

Divide and Conquer I

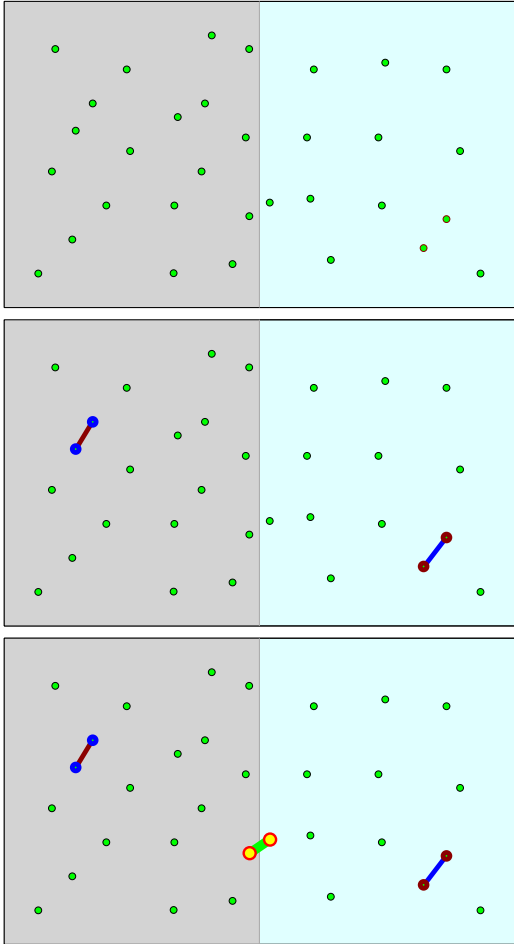
- (A) Partition into 4 quadrants of roughly equal size.
- (B) Find closest pair in each quadrant recursively.
- (C) Combine solutions.
- (D) But... How to partition the points in a balanced way?

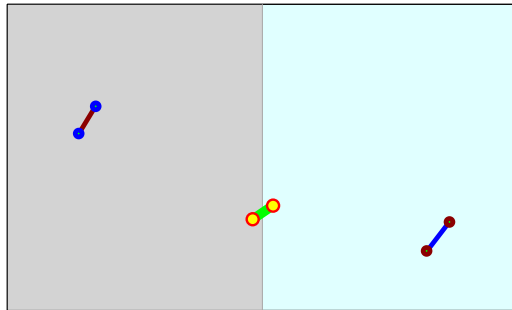
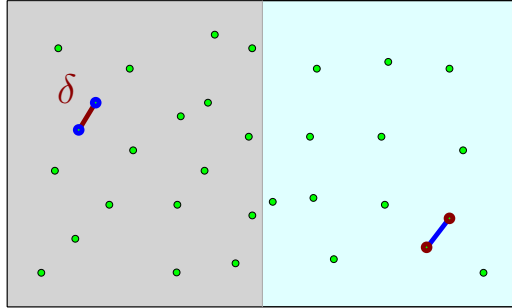


### 7.2.4.2 New Algorithm

Divide and Conquer II

- (A) Divide the set of points into two equal parts via vertical line.
- (B) Find closest pair in each half recursively.
- (C) Find closest pair with one point in each half
- (D) Return the best pair among the above 3 solutions





## 7.2.5 Towards a fast solution

### 7.2.5.1 New Algorithm

Divide and Conquer II

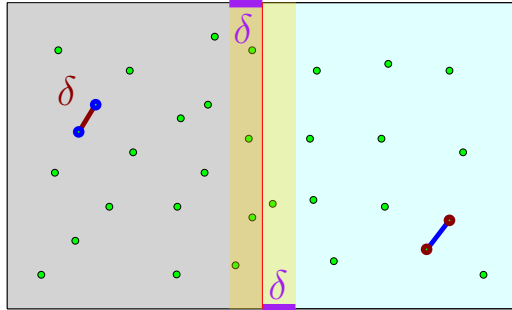
- (A) 2-3; Divide the set of points into two equal parts via vertical line
- (B) Find closest pair in each half recursively
- (C) 4-; Find closest pair with one point in each half
- (D) Return the best pair among the above 3 solutions
- (A) Sort points based on  $x$ -coordinate and pick the median. Time =  $O(n \log n)$
- (B) How to find closest pair with points in different halves?  $O(n^2)$  is trivial. Better?

### 7.2.5.2 Combining Partial Solutions

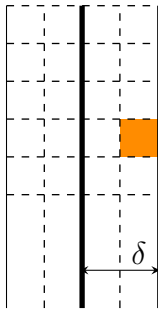
- (A) Does it take  $O(n^2)$  to combine solutions?
- (B) Let  $\delta$  be the distance between closest pairs, where both points belong to the same half.

### 7.2.5.3 Combining Partial Solutions

- (A) Let  $\delta$  be the distance between closest pairs, where both points belong to the same half.
- (B) Need to consider points within  $\delta$  of dividing line



### 7.2.5.4 Sparsity of Band

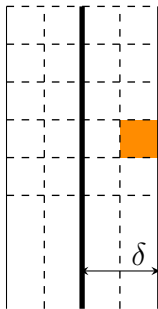


Divide the band into square boxes of size  $\delta/2$

**Lemma 7.2.1.** *Each box has at most one point*

*Proof:* If not, then there are a pair of points (both belonging to one half) that are at most  $\sqrt{2}\delta/2 < \delta$  apart! ■

### 7.2.5.5 Searching within the Band

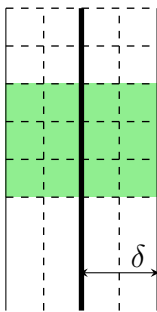


**Lemma 7.2.2.** *Suppose  $a, b$  are both in the band  $d(a, b) < \delta$  then  $a, b$  have at most two rows of boxes between them.*

*Proof:* Each row of boxes has height  $\delta/2$ . If more than two rows then  $d(a, b) > 2 \cdot \delta/2!$  ■

### 7.2.5.6 Searching within the Band

**Corollary 7.2.3.** *Order points according to their  $y$ -coordinate. If  $p, q$  are such that  $d(p, q) < \delta$  then  $p$  and  $q$  are within 11 positions in the sorted list.*



*Proof:*

(A)  $\leq 2$  points between them if  $p$  and  $q$  in same row.

(B)  $\leq 6$  points between them if  $p$  and  $q$  in two consecutive rows.

(C)  $\leq 10$  points between them if  $p$  and  $q$  one row apart.

(D)  $\implies$  More than ten points between them in the sorted  $y$  order than  $p$  and  $q$  are more than two rows apart.

(E)  $\implies d(p, q) > \delta$ . A contradiction. ■

### 7.2.5.7 The Algorithm

**ClosestPair**( $P$ ):

1. <2-3>Find vertical line  $L$  splits  $P$  into equal halves:  $P_1$  and  $P_2$
2.  $\delta_1 \leftarrow$  **ClosestPair**( $P_1$ ).
3.  $\delta_2 \leftarrow$  **ClosestPair**( $P_2$ ).
4.  $\delta = \min(\delta_1, \delta_2)$
5. <4-5>Delete points from  $P$  further than  $\delta$  from  $L$
6. <6-7>Sort  $P$  based on  $y$ -coordinate into an array  $A$
7. <8-9>**for**  $i = 1$  to  $|A| - 1$  **do**  
    <8-9>**for**  $j = i + 1$  to  $\min\{i + 11, |A|\}$  **do**  
        <8-9>**if** ( $\text{dist}(A[i], A[j]) < \delta$ ) update  $\delta$  and closest pair

- (A) Step 1, involves sorting and scanning. Takes  $O(n \log n)$  time.
- (B) Step 5 takes  $O(n)$  time.
- (C) Step 6 takes  $O(n \log n)$  time
- (D) Step 7 takes  $O(n)$  time.

## 7.2.6 Running Time Analysis

### 7.2.6.1 Running Time

The running time of the algorithm is given by

$$T(n) \leq 2T(n/2) + O(n \log n)$$

Thus,  $T(n) = O(n \log^2 n)$ . Improved Algorithm Avoid repeated sorting of points in band:  
two options

- (A) Sort all points by  $y$ -coordinate and store the list. In conquer step use this to avoid sorting
- (B) Each recursive call returns a list of points sorted by their  $y$ -coordinates. Merge in conquer step in linear time.

Analysis:  $T(n) \leq 2T(n/2) + O(n) = O(n \log n)$

## 7.3 Selecting in Unsorted Lists

### 7.3.1 Quick Sort

#### 7.3.1.1 Quick Sort

Quick Sort [Hoare]

- (A)  $i_4$  Pick a pivot element from array
- (B)  $i_2$ - $i_3$  Split array into 3 subarrays: those smaller than pivot, those larger than pivot, and the pivot itself. Linear scan of array does it. Time is  $O(n)$
- (C) Recursively sort the subarrays, and concatenate them.

Example:

- (A) array: 16, 12, 14, 20, 5, 3, 18, 19, 1

- (B) pivot: 16
- (C) split into 12, 14, 5, 3, 1 and 20, 19, 18 and recursively sort
- (D) put them together with pivot in middle

### 7.3.1.2 Time Analysis

- (A) Let  $k$  be the rank of the chosen pivot. Then,  $T(n) = T(k - 1) + T(n - k) + O(n)$
- (B) If  $k = \lceil n/2 \rceil$  then  $T(n) = T(\lceil n/2 \rceil - 1) + T(\lfloor n/2 \rfloor) + O(n) \leq 2T(n/2) + O(n)$ . Then,  $T(n) = O(n \log n)$ .
- (A) Theoretically, median can be found in linear time.
- (C) Typically, pivot is the first or last element of array. Then,

$$T(n) = \max_{1 \leq k \leq n} (T(k - 1) + T(n - k) + O(n))$$

In the worst case  $T(n) = T(n - 1) + O(n)$ , which means  $T(n) = O(n^2)$ . Happens if array is already sorted and pivot is always first element.

## 7.3.2 Selection

### 7.3.2.1 Problem - Selection

**Input** Unsorted array  $A$  of  $n$  integers

**Goal** Find the  $j$ th smallest number in  $A$  (*rank  $j$  number*)

**Example 7.3.1.**  $A = \{4, 6, 2, 1, 5, 8, 7\}$  and  $j = 4$ . The  $j$ th smallest element is 5.

**Median:**  $j = \lfloor (n + 1)/2 \rfloor$

### 7.3.3 Naïve Algorithm

#### 7.3.3.1 Algorithm I

- (A) Sort the elements in  $A$
  - (B) Pick  $j$ th element in sorted order
- Time taken =  $O(n \log n)$   
 Do we need to sort? Is there an  $O(n)$  time algorithm?

#### 7.3.3.2 Algorithm II

If  $j$  is small or  $n - j$  is small then

- (A) Find  $j$  smallest/largest elements in  $A$  in  $O(jn)$  time. (How?)
- (B) Time to find median is  $O(n^2)$ .



## 7.3.4 Divide and Conquer

### 7.3.4.1 Divide and Conquer Approach

- (A) Pick a pivot element  $a$  from  $A$
- (B) Partition  $A$  based on  $a$ .  
 $A_{\text{less}} = \{x \in A \mid x \leq a\}$  and  $A_{\text{greater}} = \{x \in A \mid x > a\}$
- (C)  $|A_{\text{less}}| = j$ : return  $a$
- (D)  $|A_{\text{less}}| > j$ : recursively find  $j$ th smallest element in  $A_{\text{less}}$
- (E)  $|A_{\text{less}}| < j$ : recursively find  $k$ th smallest element in  $A_{\text{greater}}$  where  $k = j - |A_{\text{less}}|$ .

### 7.3.4.2 Time Analysis

- (A) Steps:
  - (A) Partitioning step:  $O(n)$  time to scan  $A$
  - (B) How do we choose pivot? Recursive running time?
- (B) Suppose we always choose pivot to be  $A[1]$ .
- (C) Say  $A$  is sorted in increasing order and  $j = n$ .
- (D) Exercise: show that algorithm takes  $\Omega(n^2)$  time.

### 7.3.4.3 A Better Pivot

- (A) Suppose: pivot  $\ell$ th smallest element where  $n/4 \leq \ell \leq 3n/4$ .
- (B) That is pivot is *approximately* in the middle of  $A$ .
- (C)  $\implies n/4 \leq |A_{\text{less}}| \leq 3n/4$  and  $n/4 \leq |A_{\text{greater}}| \leq 3n/4$ .
- (D) If we apply recursion,

$$T(n) \leq T(3n/4) + O(n)$$

Implies  $T(n) = O(n)$ !

- (E) How do we find such a pivot?
- (F) Randomly? This works!  
Analysis a little bit later.
- (G) Can we choose pivot deterministically?

## 7.3.5 Median of Medians

### 7.3.6 Divide and Conquer Approach

#### 7.3.6.1 A game of medians

Idea

- (A) Break input  $A$  into many subarrays:  $L_1, \dots, L_k$ .

- (B) Find median  $m_i$  in each subarray  $L_i$ .
- (C) Find the median  $x$  of the medians  $m_1, \dots, m_k$ .
- (D) Intuition: The median  $x$  should be close to being a good median of all the numbers in  $A$ .
- (E) Use  $x$  as pivot in previous algorithm.

**But we have to be...**

More specific...

- (A) Size of each group?
- (B) How to find median of medians?

## 7.3.7 Choosing the pivot

### 7.3.7.1 A clash of medians

- (A) Partition array  $A$  into  $\lceil n/5 \rceil$  lists of 5 items each.  
 $L_1 = \{A[1], A[2], \dots, A[5]\}, L_2 = \{A[6], \dots, A[10]\}, \dots, L_i = \{A[5i + 1], \dots, A[5i + 5]\},$   
 $\dots, L_{\lceil n/5 \rceil} = \{A[5\lceil n/5 \rceil - 4], \dots, A[n]\}.$
- (B) For  $i = 1, \dots, \lceil n/5 \rceil$ : compute median  $b_i$  of  $L_i$
- (C) ...using brute-force in  $O(1)$  time. Total  $O(n)$  time
- (D) Let  $B = \{b_1, b_2, \dots, b_{\lceil n/5 \rceil}\}$
- (E) Find median  $b$  of  $B$

**Lemma 7.3.2.** *Median of  $B$  is an approximate median of  $A$ . That is, if  $b$  is used a pivot to partition  $A$ , then  $|A_{\text{less}}| \leq 7n/10 + 6$  and  $|A_{\text{greater}}| \leq 7n/10 + 6$ .*

## 7.3.8 Algorithm for Selection

### 7.3.8.1 A storm of medians

```

select( $A, j$ ):
  Form lists  $L_1, L_2, \dots, L_{\lceil n/5 \rceil}$ , where  $L_i = \{A[5i - 4], \dots, A[5i]\}$ 
  Find median  $b_i$  of each  $L_i$  using brute-force
  Find median  $b$  of  $B = \{b_1, b_2, \dots, b_{\lceil n/5 \rceil}\}$ 
  Partition  $A$  into  $A_{\text{less}}$  and  $A_{\text{greater}}$  using  $b$  as pivot
  if ( $|A_{\text{less}}| = j$ ) return  $b$ 
  else if ( $|A_{\text{less}}| > j$ )
    return select( $A_{\text{less}}, j$ )
  else
    return select( $A_{\text{greater}}, j - |A_{\text{less}}|$ )

```

How do we find median of  $B$ ? Recursively!

### 7.3.9 Running time of deterministic median selection

#### 7.3.9.1 A dance with recurrences

$$T(n) = T(\lceil n/5 \rceil) + \max\{T(|A_{\text{less}}|), T(|A_{\text{greater}}|)\} + O(n)$$

From Lemma,

$$T(n) \leq T(\lceil n/5 \rceil) + T(\lfloor 7n/10 + 6 \rfloor) + O(n)$$

and

$$T(1) = 1$$

**Exercise:** show that  $T(n) = O(n)$

**Lemma 7.3.3.** For  $T(n) \leq T(\lceil n/5 \rceil) + T(\lfloor 7n/10 + 6 \rfloor) + O(n)$ , it holds that  $T(n) = O(n)$ .

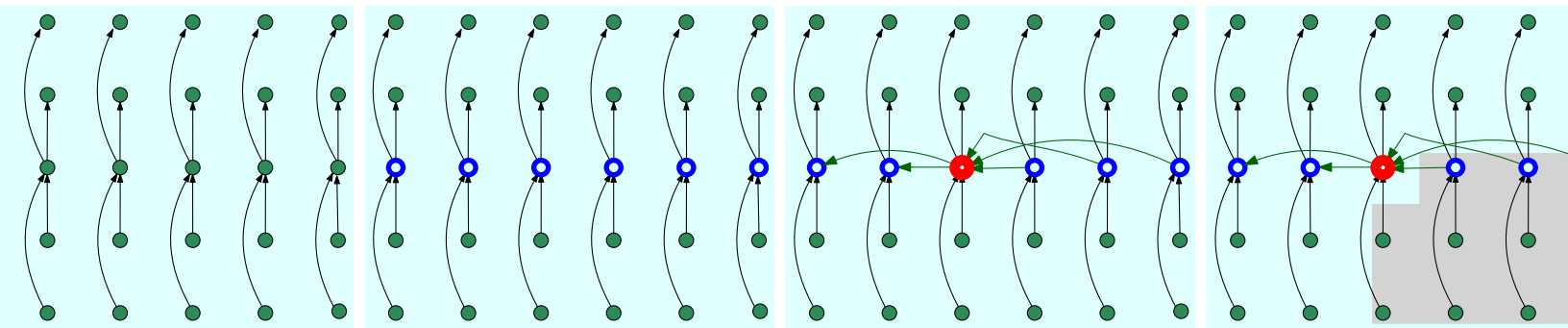
*Proof:* We claim that  $T(n) \leq cn$ , for some constant  $c$ . We have that  $T(i) \leq c$  for all  $i = 1, \dots, 1000$ , by picking  $c$  to be sufficiently large. This implies the base of the induction. Similarly, we can assume that the  $O(n)$  in the above recurrence is smaller than  $cn/100$ , by picking  $c$  to be sufficiently large.

So, assume the claim holds for any  $i < n$ , and we will prove it for  $n$ . By induction, we have

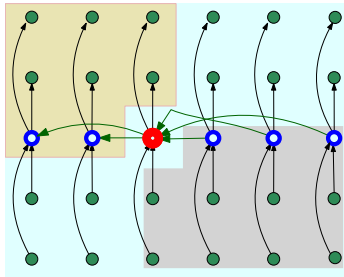
$$\begin{aligned} T(n) &\leq T(\lceil n/5 \rceil) + T(\lfloor 7n/10 + 6 \rfloor) + O(n) \\ &\leq c(n/5 + 1) + c(7n/10 + 6) + cn/100 \\ &= cn(1/5 + 7/10 + 1/100 + 1/n + 6/n) \leq cn, \end{aligned}$$

for  $n > 1000$ . ■

#### 7.3.9.2 Median of Medians: The movie



### 7.3.9.3 Median of Medians: Proof of Lemma



**Proposition 7.3.4.** *There are at least  $3n/10 - 6$  elements greater than the median of medians  $b$ .*

*Proof:* At least half of the  $\lceil n/5 \rceil$  groups have at least 3 elements larger than  $b$ , except for last group and the group containing  $b$ . So  $b$  is less than

$$3(\lceil (1/2)\lceil n/5 \rceil \rceil - 2) \geq 3n/10 - 6$$

Figure 7.1: Shaded elements are all greater than  $b$  ■

### 7.3.9.4 Median of Medians: Proof of Lemma

**Proposition 7.3.5.** *There are at least  $3n/10 - 6$  elements greater than the median of medians  $b$ .*

**Corollary 7.3.6.**  $|A_{less}| \leq 7n/10 + 6$ .

Via symmetric argument,

**Corollary 7.3.7.**  $|A_{greater}| \leq 7n/10 + 6$ .

### 7.3.9.5 Questions to ponder

- (A) Why did we choose lists of size 5? Will lists of size 3 work?
- (B) Write a recurrence to analyze the algorithm's running time if we choose a list of size  $k$ .

### 7.3.9.6 Median of Medians Algorithm

Due to: M. Blum, R. Floyd, D. Knuth, V. Pratt, R. Rivest, and R. Tarjan.  
"Time bounds for selection".

Journal of Computer System Sciences (JCSS), 1973.

How many Turing Award winners in the author list?

All except Vaughn Pratt!

### 7.3.9.7 Takeaway Points

- (A) Recursion tree method and guess and verify are the most reliable methods to analyze recursions in algorithms.
- (B) Recursive algorithms naturally lead to recurrences.
- (C) Some times one can look for certain type of recursive algorithms (reverse engineering) by understanding recurrences and their behavior.