CS38
Introduction to Algorithms

Lecture 7
April 22, 2014

---

## Outline

- Divide and Conquer design paradigm
  - Mergesort
  - Quicksort
  - Selection
    - both with random pivot
    - deterministic selection
  - Closest pair



---

## Divide and conquer

- General approach
  - break problem into subproblems
  - solve each subproblem recursively
  - combine solutions
- typical running time recurrence:

$T(1) = O(1)$
$T(n) \leq a \cdot T(N/b) + O(n^c)$

\# subproblems — size of subproblems — cost to split and combine



---

## Solving D&C recurrences

$T(1) = O(1)$
$T(n) \leq a \cdot T(N/b) + O(n^c)$

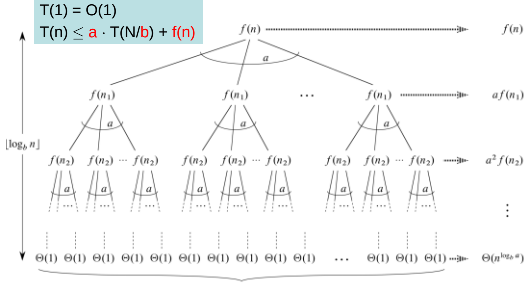
- key quantity: $\log_b a = D$
  - if c < D then $T(n) = O(n^D)$
  - if c = D then $T(n) = O(n^D \cdot \log n)$
  - if c > D then $T(n) = O(n^c)$
- can prove easily by induction



---

## Why $\log_b a$?

$T(1) = O(1)$
$T(n) \leq a \cdot T(N/b) + f(n)$




---

## First example: mergesort

- Input: n values; sort in increasing order.
- Mergesort algorithm:
  - split list into 2 lists of size n/2 each
  - recursively sort each
  - merge in linear time (how?)
- Running time:
  - $T(1) = 1$; $T(n) = 2 \cdot T(n/2) + O(n)$
- Solution: $T(n) = O(n \log n)$

## Second example: quicksort

- Quicksort: effort on split rather than merge
- Quicksort algorithm:
  - take first value x as pivot
  - split list into "< x" and "> x" lists
  - recursively sort each
- Why isn't this the running time recurrence:

  $T(1) = 1;\ T(n) = 2 \cdot T(n/2) + O(n)$
- Worst case running time: $O(n^2)$ (why?)



## Quicksort with random pivot

**Random-Pivot-Quicksort(array of n elements: a)**
1. if n = 1, return(a)
2. pick i uniformly at random from {1,2,… n}
3. partition array a into "< $a_i$" and "> $a_i$" arrays
4. **Random-Pivot-Quicksort**("< $a_i$")
5. **Random-Pivot-Quicksort**("> $a_i$")
6. return("< $a_i$", $a_i$, "> $a_i$")

- Idea: hope that $a_i$ splits array into two subarrays of size $\approx n/2$
  - would lead to $T(1) = 1;\ T(n) = 2 \cdot T(n/2) + O(n)$

    and then $T(n) = O(n \log n)$



## Quicksort with random pivot

**Random-Pivot-Quicksort(array of n elements: a)**
1. if n = 1, return(a)
2. pick i uniformly at random from {1,2,… n}
3. partition array a into "< $a_i$" and "> $a_i$" arrays
4. **Random-Pivot-Quicksort**("< $a_i$")
5. **Random-Pivot-Quicksort**("> $a_i$")
6. return("< $a_i$", $a_i$, "> $a_i$")

- we will analyze expected running time
  - suffices to count aggregate # comparisons
  - rename elements of a: $x_1 \le x_2 \le \cdots \le x_n$
  - when is $x_i$ compared with $x_j$?



## Quicksort with random pivot

- when is $x_i$ compared with $x_j$? ($i < j$)
  - consider elements $[x_i, x_{i+1}, x_{i+2}, \ldots, x_{j-1}, x_j]$
  - if any blue element is chosen first, then no comparison (why?)
  - if either red element is chosen first, then exactly one comparison (why?)
  - probability $x_i$ and $x_j$ compared = $2/(j - i + 1)$



## Quicksort with random pivot

- probability $x_i$ and $x_j$ compared = $2/(j - i + 1)$
- so expected number of comparisons is

$$\sum_{i<j} 2/(j - i + 1)$$
$$= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} 2/(j - i + 1)$$
$$= \sum_{i=1}^{n} \sum_{k=1}^{n-i} 2/(k+1)$$
$$< \sum_{i=1}^{n} \sum_{k=1}^{n-i} 2/k$$
$$= \sum_{i=1}^{n} O(\log n)$$
$$= O(n \log n)$$

harmonic series: $\sum_{i=1}^{n} 1/i = O(\log n)$



## Quicksort with random pivot

**Random-Pivot-Quicksort(array of n elements: a)**
1. if n = 1, return(a)
2. pick i uniformly at random from {1,2,… n}
3. partition array a into "< $a_i$" and "> $a_i$" arrays
4. **Random-Pivot-Quicksort**("< $a_i$")
5. **Random-Pivot-Quicksort**("> $a_i$")
6. return("< $a_i$", $a_i$, "> $a_i$)

we proved:

**Theorem**: Random-Pivot-Quicksort runs in expected time $O(n \log n)$.

note: by Markov, prob. running time > 100 · expectation < 1/100

## Selection

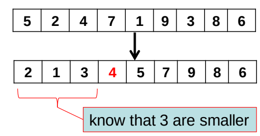
- Input: $n$ values; find k-th smallest
  - minimum: $k = 1$
  - maximum: $k = n$
  - median: $k = \lfloor (n+1)/2 \rfloor$
  - running time for min or max?
- running time for general $k$?
  - using sorting: $O(n \log n)$
  - using a min-heap: $O(n + k \log n)$



## Selection

- running time for general $k$?
  - using sorting: $O(n \log n)$
  - using a min-heap: $O(n + k \log n)$
  - we will see: $O(n)$
- Intuition: like quicksort with recursion on only 1 subproblem

$T(n) = T(n/c) + O(n)$
Solution: $T(n) = O(n)$






## Selection with random pivot

**Random-Pivot-Select(k; array of n elements: a)**
1. pick i uniformly at random from {1,2,… n}
2. partition array a into "< $a_i$" and "> $a_i$" arrays
3. s = size of "< a_i" array
4. if s = k-1, then return(a_i)
5. else if s < k-1, **Random-Pivot-Select**(k – s + 1, "> $a_i$")
6. else if s > k-1, **Random-Pivot-Select**(k, "<$a_i$")

- Bounding the expected # comparisons:
  - $T(n,k)$ = expected # for $k$-th smallest from $n$
  - $T(n) = \max_k T(n,k)$
  - Observe: $T(n)$ monotonically increasing



## Selection with random pivot

**Random-Pivot-Select(k; array of n elements: a)**
1. pick i uniformly at random from {1,2,… n}
2. partition array a into "< $a_i$" and "> $a_i$" arrays
3. s = size of "< a_i" array
4. if s = k-1, then return(a_i)
5. else if s < k-1, **Random-Pivot-Select**(k – s + 1, "> $a_i$")
6. else if s > k-1, **Random-Pivot-Select**(k, "<$a_i$")

- Bounding the expected # comparisons:
  - probability of choosing $i$-th largest = ?
  - resulting subproblems sizes are $n-i$, $i-1$
  - upper bound expectation by taking larger



## Selection with random pivot

**Random-Pivot-Select(k; array of n elements: a)**
1. pick i uniformly at random from {1,2,… n}
2. partition array a into "< $a_i$" and "> $a_i$" arrays
3. s = size of "< a_i" array
4. if s = k-1, then return(a_i)
5. else if s < k-1, **Random-Pivot-Select**(k – s + 1, "> $a_i$")
6. else if s > k-1, **Random-Pivot-Select**(k, "<$a_i$")

- Claim: [to split] [probability pivot is i-th largest]

$T(n) \le n + 1/n\cdot[T(n/2)+T(n/2+1)+\ldots+T(n-1)]$
$+ 1/n\cdot[T(n/2)+T(n/2+1)+\ldots+T(n-1)]$

[max (n-i, i-1) as i = 1…n]



## Selection with random pivot

$T(n) \le n + 1/n\cdot[T(n/2)+T(n/2+1)+\ldots+T(n-1)]$
$+ 1/n\cdot[T(n/2)+T(n/2+1)+\ldots+T(n-1)]$

**Claim**: $T(n) \le 4n$.

Proof: induction on n.
- assume true for 1… n-1
- $T(n) \le n + 2/n\cdot[T(n/2)+T(n/2+1)+\ldots+T(n-1)]$
- $T(n) \le n + 2/n\cdot[4(n/2)+4(n/2+1)+\ldots+4(n-1)]$
- $T(n) \le n + 8/n\cdot[(3/8)n^2] < 4n.$

## Linear-time selection

**Select(k; array of n elements: a)**

1. pick i from {1,2,… n} and partition array a into "< $a_i$" and "> $a_i$" arrays
   *** guarantee that both arrays have size at most (7/10) n
1. s = size of "< a_i" array
2. if s = k-1, then return(a_i)
3. else if s < k-1, **Select**(k – s + 1, "> $a_i$")
4. else if s > k-1, **Select**(k, "<$a_i$")

solution is T(n) = O(n) because 1/5 + 7/10 < 1

- Clever way to achieve guarantee:
  - break array up into subsets of 5 elements
  - recursively compute median of medians of these sets
  - leads to T(n) = T((1/5)n) + T((7/10)n) + O(n)



## Linear-time selection

Median of medians example (n = 48):

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 17 | 3 | 38 | 25 | 14 | 48 | 37 | 40 | 8 | 13 |
| 27 | 2 | 26 | 11 | 19 | 6 | 12 | 31 | 20 | 22 |
| 47 | 16 | 41 | 1 | 44 | 36 | 43 | 4 | 46 | 32 |
| 10 | 34 | 7 | 42 | 15 | 5 | 18 | 39 | 21 | |
| 28 | 23 | 35 | 45 | 30 | 24 | 29 | 9 | 33 | |



## Linear-time selection

find median in each column

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 17 | 3 | 38 | 25 | 14 | 48 | 37 | 40 | 8 | 13 |
| 27 | 2 | 26 | 11 | 19 | 6 | 12 | 31 | 20 | 22 |
| 47 | 16 | 41 | 1 | 44 | 36 | 43 | 4 | 46 | 32 |
| 10 | 34 | 7 | 42 | 15 | 5 | 18 | 39 | 21 | |
| 28 | 23 | 35 | 45 | 30 | 24 | 29 | 9 | 33 | |



## Linear-time selection

find median in each column

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 17 | 3 | 38 | 25 | 14 | 48 | 37 | 40 | 8 | 13 |
| 27 | 2 | 26 | 11 | 19 | 6 | 12 | 31 | 20 | 22 |
| 47 | 16 | 41 | 1 | 44 | 36 | 43 | 4 | 46 | 32 |
| 10 | 34 | 7 | 42 | 15 | 5 | 18 | 39 | 21 | |
| 28 | 23 | 35 | 45 | 30 | 24 | 29 | 9 | 33 | |



## Linear-time selection

find median of this subset of $\lceil n/5 \rceil = 10$

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 17 | 3 | 38 | 25 | 14 | 48 | 37 | 40 | 8 | 13 |
| 27 | 2 | 26 | 11 | 19 | 6 | 12 | 31 | 20 | 22 |
| 47 | 16 | 41 | 1 | 44 | 36 | 43 | 4 | 46 | 32 |
| 10 | 34 | 7 | 42 | 15 | 5 | 18 | 39 | 21 | |
| 28 | 23 | 35 | 45 | 30 | 24 | 29 | 9 | 33 | |



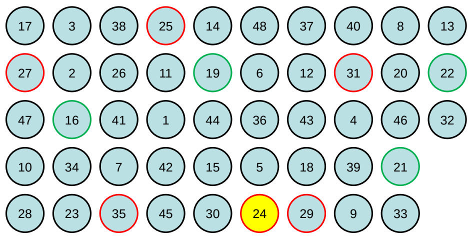
## Linear-time selection

How many < this one?

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 17 | 3 | 38 | 25 | 14 | 48 | 37 | 40 | 8 | 13 |
| 27 | 2 | 26 | 11 | 19 | 6 | 12 | 31 | 20 | 22 |
| 47 | 16 | 41 | 1 | 44 | 36 | 43 | 4 | 46 | 32 |
| 10 | 34 | 7 | 42 | 15 | 5 | 18 | 39 | 21 | |
| 28 | 23 | 35 | 45 | 30 | 24 | 29 | 9 | 33 | |

## Linear-time selection

How many $\leq$ this one? $\lceil n/5 \rceil/2$-1=10/2 -1=4





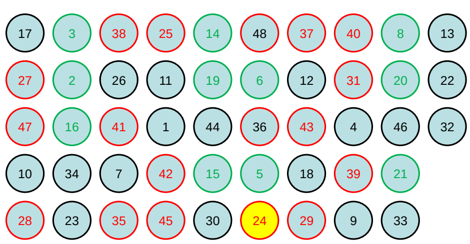

## Linear-time selection

Total $\leq$ this one? at least $(\lceil n/5 \rceil / 2 - 2) \cdot 3$




## Linear-time selection

Total $\geq$ this one? at least $(\lceil n/5 \rceil / 2 - 2) \cdot 3$






## Linear-time selection

- To find pivot: break array into subsets of 5
  - find median of each 5
  - find median of medians

**Claim**: at most (7/10)n + 6 elements are smaller than pivot

Proof: at least

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

are larger or equal.



## Linear-time selection

- To find pivot: break array into subsets of 5
  - find median of each 5
  - find median of medians

**Claim**: at most (7/10)n + 6 elements are larger than pivot

Proof: at least

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

are smaller.



## Linear-time selection

**Select(k; array of n elements: a)**

1. pick i from {1,2,… n} using median of medians method
2. partition array a into "< $a_i$" and "> $a_i$" arrays
3. s = size of "< a_i" array
4. if s = k-1, then return(a_i)
5. else if s < k-1, **Select**(k – s + 1, "> $a_i$")
6. else if s > k-1, **Select**(k, "<$a_i$")

- Running time:
  - $T(n) = O(1)$ if n < 140
  - $T(n) \leq T(n/5 + 1) + T(7/10 + 6) + cn$ otherwise
  - we claim that $T(n) \leq 20cn$

## Linear-time selection

- Running time:
  - T(n) = O(1) if n < 140
  - $T(n) \le T(n/5 + 1) + T(7/10 + 6) + cn$ otherwise

**Claim**: $T(n) \le 20cn$

Proof: induction on n; base case easy

$T(n) \le T(n/5 + 1) + T(7/10 + 6) + cn$

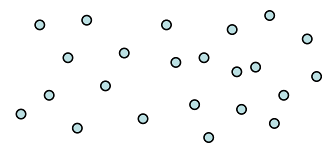
$T(n) \le 20c(n/5 + 1) + 20c(7/10 + 6) + cn$

$T(n) \le 19cn + 140c \le 20cn$ provided $n \ge 140$



## Closest pair in the plane

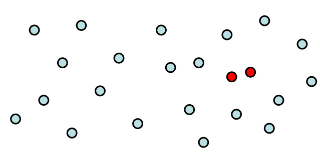
- Given n points in the plane, find the closest pair





## Closest pair in the plane

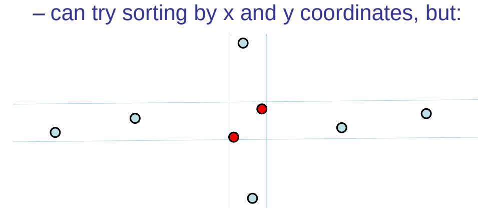
- Given n points in the plane, find the closest pair
  - $O(n^2)$ if compute all pairwise distances
  - 1 dimensional case?





## Closest pair in the plane

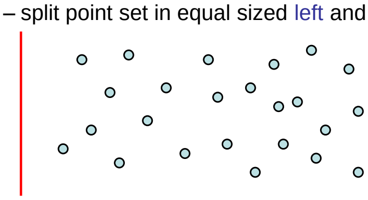
- Given n points in the plane, find the closest pair
  - can try sorting by x and y coordinates, but:





## Closest pair in the plane

- Divide and conquer approach:
  - split point set in equal sized left and right sets

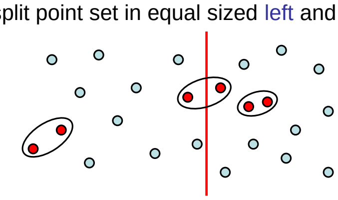


  - find closest pair in left, right, + across middle



## Closest pair in the plane

- Divide and conquer approach:
  - split point set in equal sized left and right sets



  - find closest pair in left, right, + across middle

# Closest pair in the plane

- Divide and conquer approach:
  - split point set in equal sized left and right sets
  - time to perform split?
  - sort by x coordinate: O(n log n)
  - running time recurrence:

    T(n) = 2T(n/2) + time for middle + O(n log n)

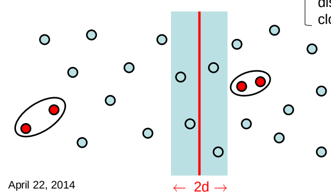
Is time for middle as bad as O($n^2$)?



# Closest pair in the plane

Claim: time for middle only O(n log n) !!

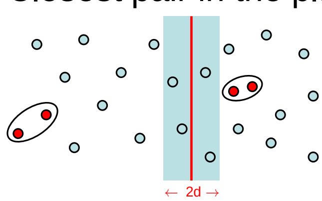
- key: we know d = min of
  - distance between closest pair on left
  - distance between closest pair on right




Observation: only need to consider points within distance d of the midline



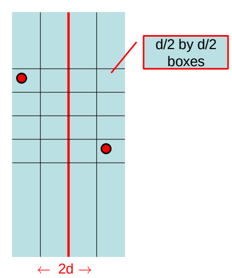
# Closest pair in the plane




- scan left to right to identify, then sort by y coord.
  - still $\Omega(n^2)$ comparisons?
  - Claim: only need do pairwise comparisons 15 ahead in this sorted list !



# Closest pair in the plane




- no 2 points lie in same box (why?)
- if 2 points are within ≥ 16 positions of each other in list sorted by y coord…
- … then they must be separated by ≥ 3 rows
- implies dist. > (3/2)· d



# Closest pair in the plane

**Closest-Pair(P: set of n points in the plane)**

1. sort by x coordinate and split equally into L and R subsets
2. (p,q) = **Closest-Pair(L)**
3. (r,s) = **Closest-Pair(R)**
4. d = min(distance(p,q), distance(r,s))
5. scan P by x coordinate to find M: points within d of midline
6. sort M by y coordinate
7. compute closest pair among all pairs within 15 of each other in M
8. return best among this pair, (p,q), (r,s)

- Running time:

  T(2) = O(1); T(n) = 2T(n/2) + O(n log n)



# Closest pair in the plane

- Running time:

  T(2) = a; T(n) = 2T(n/2) + bn·log n

  set c = max(a/2, b)

Claim: T(n) $\leq$ cn·$\log^2$n

Proof: base case easy…

$T(n) \leq 2T(n/2) + bn \cdot \log n$

$\leq 2cn/2(\log n - 1)^2 + bn \cdot \log n$

$< cn(\log n)(\log n - 1) + bn \cdot \log n$

$\leq cn\log^2 n$

## Closest pair in the plane

- we have proved:

> **Theorem**: There is an $O(n \log^2 n)$ time algorithm for finding the closest pair among $n$ points in the plane.

- can be improved to $O(n \log n)$ by being more careful about maintaining sorted lists