# Order Statistics

David Kauchak
cs140
Spring 2024

1

## Administrative

Group session timing (see slack post)

2

## Medians

The median of a set of numbers is the number such that half of the numbers are larger and half smaller

A = [50, 12, 1, 97, $\boxed{30}$]

How might we calculate the median of a set?

Sort the numbers, then pick the n/2 element

A = [1, 12, $\boxed{30,}$ 50, 97]

runtime?

3

## Medians

The median of a set of numbers is the number such that half of the numbers are larger and half smaller

A = [50, 12, 1, 97, $\boxed{30}$]

How might we calculate the median of a set?

Sort the numbers, then pick the n/2 element

A = [1, 12, $\boxed{30,}$ 50, 97]

$\Theta$(n log n)

4

## Selection

More general problem:
find the *k*-th smallest element in an array

- i.e. element where exactly k-1 things are smaller than it
- aka the "selection" problem
- can use this to find the median if we want

Can we solve this in a similar way?

- Yes, sort the data and take the kth element
- Θ(n log n)

5

## Can we do better?

Are we doing more work than we need to?

To get the k-th element (or the median) by sorting, we're finding *all* the k-th elements at once

We just want the one!

Often when you find yourself doing more work than you need to, there is a faster way (though not always)

6

## selection problem

Our tools

- divide and conquer
- sorting algorithms
- other functions
  - merge
  - partition
  - binary search

7

## Partition

Partition takes Θ(n) time and performs a similar operation

given an element A[q], Partition can be seen as dividing the array into three sets:

- < A[q]
- = A[q]
- > A[q]

Ideas?

8

## An example

We're looking for the 5th smallest

5 2 34 9 17 2 1 34 18 5 3 2 1 6 5

If we called partition, what would be the in three sets?

< 5:

= 5:

> 5:

9

## An example

We're looking for the 5th smallest

5 2 34 9 17 2 1 34 18 5 3 2 1 6 5

< 5: 2 2 1 3 2 1

= 5: 5 5 5

> 5: 34 9 17 34 18 6

Does this help us?

10

## An example

We're looking for the 5th smallest

5 2 34 9 17 2 1 34 18 5 3 2 1 6 5

< 5: 2 2 1 3 2 1

We know the 5th smallest has to be in this set

= 5: 5 5 5

> 5: 34 9 17 34 18 6

11

```
Selection(A, k, p, r)
  q ← Partition(A,p,r)
  relq = q-p+1
  if k = relq
    Return A[q]
  else if k < relq
    Return Selection(A, k, p, q-1)
  else // k > relq
    Return Selection(A, k-relq, q+1, r)
```

A: array of data
k: find the kth smallest
p,r: current span we're exploring (initially 1, len(A))

12

## Selection: divide and conquer

Call partition

- decide which of the three sets contains the answer we're looking for
- recurse

Like binary search on unsorted data

```
Selection(A, k, p, r)
  q ← Partition(A,p,r)
  relq = q-p+1
  if k = relq
    Return A[q]
  else if k < relq
    Return Selection(A, k, p, q-1)
  else // k > relq
    Return Selection(A, k-relq, q+1, r)
```

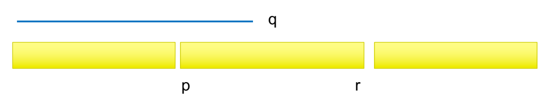
?

13

## relq

relq = q – p + 1

Partition returns the absolute index, we want an index relative to the current p (window start)

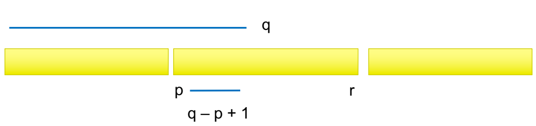



14

## relq

relq = q – p + 1

Partition returns the absolute index, we want an index relative to the current p (window start)

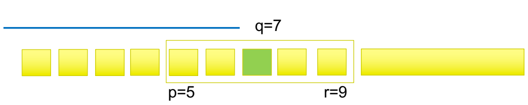



15

## relq

relq = q – p + 1

Partition returns the absolute index, we want an index relative to the current p (window start)



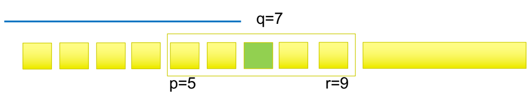

What is relq?

16

## relq

relq = q – p + 1

Partition returns the absolute index, we want an index relative to the current p (window start)




7-5+1=3

17

## Selection: divide and conquer

Call partition

- decide which of the three sets contains the answer we're looking for
- recurse

Like binary search on unsorted data

```
Selection(A, k, p, r)
  q ← Partition(A,p,r)
  relq = q-p+1
  if k = relq
    Return A[q]
  else if k < relq
    Return Selection(A, k, p, q-1)
  else // k > relq
    Return Selection(A, k-relq, q+1, r)
```

As we recurse, we may update the k that we're looking for because we update the lower end

19

Selection(A, 3, 1, 8)

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

```
Selection(A, k, p, r)
  q ← Partition(A,p,r)
  relq = q-p+1
  if k = relq
    Return A[q]
  else if k < relq
    Selection(A, k, p, q-1)
  else // k > relq
    Selection(A, k-relq, q+1, r)
```

20

Selection(A, 3, 1, 8)

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

relq = 6 – 1 + 1 = 6

```
Selection(A, k, p, r)
  q ← Partition(A,p,r)
  relq = q-p+1
  if k = relq
    Return A[q]
  else if k < relq
    Selection(A, k, p, q-1)
  else // k > relq
    Selection(A, k-relq, q+1, r)
```

21

## Selection(A, 3, 1, 8)

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

relq = 6 – 1 + 1 = 6

```
Selection(A, k, p, r)
  q ← Partition(A,p,r)
  relq = q-p+1
  if k = relq
    Return A[q]
  else if k < relq
    Selection(A, k, p, q-1)
  else // k > relq
    Selection(A, k-relq, q+1, r)
```

22

## Selection(A, 3, 1, 5)

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

At each call, discard part of the array

```
Selection(A, k, p, r)
  q ← Partition(A,p,r)
  relq = q-p+1
  if k = relq
    Return A[q]
  else if k < relq
    Selection(A, k, p, q-1)
  else // k > relq
    Selection(A, k-relq, q+1, r)
```

23

## Selection(A, 3, 1, 5)

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

relq = 2 – 1 + 1 = 2

```
Selection(A, k, p, r)
  q ← Partition(A,p,r)
  relq = q-p+1
  if k = relq
    Return A[q]
  else if k < relq
    Selection(A, k, p, q-1)
  else // k > relq
    Selection(A, k-relq, q+1, r)
```

24

## Selection(A, 1, 3, 5)

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

```
Selection(A, k, p, r)
  q ← Partition(A,p,r)
  relq = q-p+1
  if k = relq
    Return A[q]
  else if k < relq
    Selection(A, k, p, q-1)
  else // k > relq
    Selection(A, k-relq, q+1, r)
```

25

Selection(A, 1, 3, 5)

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

```
Selection(A, k, p, r)
  q ← Partition(A,p,r)
  relq = q-p+1
  if k = relq
    Return A[q]
  else if k < relq
    Selection(A, k, p, q-1)
  else // k > relq
    Selection(A, k-relq, q+1, r)
```

26

Selection(A, 1, 3, 5)

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

relq = 5 – 3 + 1 = 3

```
Selection(A, k, p, r)
  q ← Partition(A,p,r)
  relq = q-p+1
  if k = relq
    Return A[q]
  else if k < relq
    Selection(A, k, p, q-1)
  else // k > relq
    Selection(A, k-relq, q+1, r)
```

27

Selection(A, 1, 3, 4)

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

```
Selection(A, k, p, r)
  q ← Partition(A,p,r)
  relq = q-p+1
  if k = relq
    Return A[q]
  else if k < relq
    Selection(A, k, p, q-1)
  else // k > relq
    Selection(A, k-relq, q+1, r)
```

28

Selection(A, 1, 3, 4)

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

```
Selection(A, k, p, r)
  q ← Partition(A,p,r)
  relq = q-p+1
  if k = relq
    Return A[q]
  else if k < relq
    Selection(A, k, p, q-1)
  else // k > relq
    Selection(A, k-relq, q+1, r)
```

29

Selection(A, 1, 3, 4)

1 2 3 4 5 6 7 8

1 2 3 4 5 6 8 7

↑

Selection(A, k, p, r)

```
q ← Partition(A,p,r)
relq = q-p+1
if k = relq
   Return A[q]
else if k < relq
   Selection(A, k, p, q-1)
else // k > relq
   Selection(A, k-relq, q+1, r)
```

30

Selection(A, 1, 3, 4)

1 2 3 4 5 6 7 8

1 2 $\boxed{3}$ 4 5 6 8 7

↑

relq = 3 – 3 + 1 = 1

Selection(A, k, p, r)

```
q ← Partition(A,p,r)
relq = q-p+1
if k = relq
   Return A[q]
else if k < relq
   Selection(A, k, p, q-1)
else // k > relq
   Selection(A, k-relq, q+1, r)
```

31

## Running time of Selection?

Best case?

We get lucky and the element at the end of the list is the kth smallest element!

One call to partition: θ(n)

32

## Running time of Selection?

Worst case?

Each call to Partition only reduces our search by 1

Recurrence?

$$T(n) = T(n-1) + \Theta(n)$$

$\theta(n^2)$

33

## Running time of Selection?

Worst case?
Each call to Partition only reduces our search by 1

When does this happen?

- sorted
- reverse sorted
- others…

34

## How can randomness help us?

```
RSelection(A, k, p, r)
  q ← RPartition(A,p,r)
  if k = q
    Return A[q]
  else if k < q
    Return Selection(A, k, p, q-1)
  else // k > q
    Return Selection(A, k, q+1, r)
```

35

## Running time of RSelection?

Best case

- $\theta(n)$

Worst case

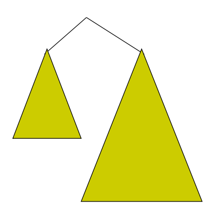
- Still $\theta(n^2)$
- As with Quicksort, we can get unlucky

Average case?

36

## Average case

Depends on how much data we throw away at each step



37

## Average case

We'll call a partition "good" if the pivot falls within within the $25^{th}$ and $75^{th}$ percentile

- a "good" partition throws away at least a quarter of the data
- Or, each of the partitions contains at least 25% of the data

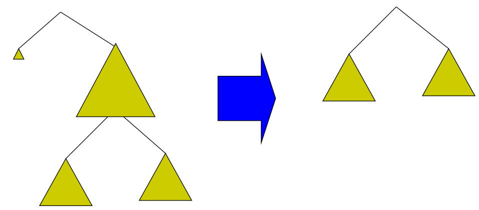
What is the probability of a "good" partition?

Half of the elements lie within this range and half outside, so 50% chance

38

## Average case

Recall, that like Quicksort, we can absorb the cost of a constant number of "bad" partitions



39

## Average case

On average, how many times will Partition need to be called before we get a good partition?

Let E be the number of times

Recurrence:

$$E = 1 + \frac{1}{2}E$$

half the time we get a good partition on the first try and half of the time, we have to try again

$$= 1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{16} + \ldots$$

$$= 2$$

40

## Mathematicians and beer

An infinite number of mathematicians walk into a bar. The first one orders a beer. The second orders half a beer. The third, a quarter of a beer. The bartender says "You're all idiots", and pours two beers.

41

## Average case

If on average we can get a "good" partition ever other time, what is the recurrence?

- recall the pivot of a "good" partition falls in the 25th and 75th percentile

43

## Average case

If on average we can get a "good" partition ever other time, what is the recurrence?

- recall the pivot of a "good" partition falls in the 25th and 75th percentile

$$T(n) = T\left(\frac{3}{4}n\right) + \theta(n)$$

We throw away at least ¼ of the data

roll in the cost of the "bad" partitions

44

## Which is?

$$T(n) = T(3/4n) + \theta(n)$$

45

$$T(n) = T(3/4n) + \Theta(n)$$

$$\text{if } f(n) = O(n^{\log_b a - \varepsilon}) \text{ for } \varepsilon > 0, \text{ then } T(n) = \Theta(n^{\log_b a})$$
$$\text{if } f(n) = \Theta(n^{\log_b a}), \text{ then } T(n) = \Theta(n^{\log_b a} \log n)$$
$$\text{if } f(n) = \Omega(n^{\log_b a + \varepsilon}) \text{ for } \varepsilon > 0 \text{ and } af(n/b) \le cf(n) \text{ for } c < 1$$
$$\text{then } T(n) = \Theta(f(n))$$

a = 1
b = 4/3
*f(n)* = *n*

$$n^{\log_b a} = n^{\log_{4/3} 1} = n^0$$

$$\text{is n} = O(n^{0-\varepsilon})?$$
$$\text{is n} = \Theta(n^{0})?$$
$$\text{is n} = \Omega(n^{0+\varepsilon})?$$

**Case 3:** Θ(n)

Average case running time!

46

## Selection

Worst case: $\Theta(n^2)$

Best case: $\Theta(n)$

Average case: $\Theta(n)$

47

## An aside…

Notice a trend?

$$T(n) = T(n/2) + \Theta(n) \qquad \Theta(n)$$

$$T(n) = T(3/4n) + \Theta(n) \qquad \Theta(n)$$

48

$T(n) = T(pn) + f(n)$

for $0 < p < 1$ and

$f(n) \notin \Theta(1)$

if $f(n) = O(n^{\log_b a - \varepsilon})$ for $\varepsilon > 0$, 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 $f(n) = \Omega(n^{\log_b a + \varepsilon})$ for $\varepsilon > 0$ and $af(n/b) \le cf(n)$ for $c < 1$ then $T(n) = \Theta(f(n))$

a = 1
b = 1/p
f(n) = f(n)

$$n^{\log_b a} = n^{\log_{1/p} 1} = n^0$$

**Case 3:** $\Theta(f(n))$

49

## Divide and conquer strategy

Split data in half and recurse on two halves

Assume it works! How do we get the answer to the entire problem?

- Often have to do a bit of extra work
- Be careful about solutions that could span/combine the two halves

50

## Data structures

What is a data structure?

Way of storing data that **facilitates particular operations**

51

## Data structures

What are some of the data structures that you've seen?

52

## Data structures review

List

Ordered Set

Priority Queue

Unordered Set

1. What operations do they support?
2. What are they good at?
3. How can we implement them? (Are there variations?)
4. What are the runtimes for the operations? (Do variations matter?)

53

## Lists

get/set at index

append (add at the end of the list)

remove

add/insert

54

## Ordered Set

insert

remove

contains

next/prev (successor/predecessor)

55

## Priority Queue

insert

remove

min/max

56

## Unordered Set

insert

remove

contains

57