Order Statistics

David Kauchak
cs140
Spring 2023

Administrative

Assignment 2: how did it go?
Assignment 3 out soon

Pseudocode
- Make sure it’s understandable
- Use indenting where appropriate to highlight structure
- Consider using "verbatim" to format

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, 30]

How might we calculate the median of a set?
Sort the numbers, then pick the n/2 element

A = [1, 12, 30, 50, 97]

runtime?

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, 30]

How might we calculate the median of a set?
Sort the numbers, then pick the n/2 element

A = [1, 12, 30, 50, 97]

Θ(n log n)
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 $k$th element
  $\Theta(n \log n)$

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)

selection problem
Our tools
- divide and conquer
- sorting algorithms
- other functions
  - merge
  - partition
  - binary search

Partition
Partition takes $\Theta(n)$ 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?
An example
We're looking for the 5th smallest

5 2 3 4 9 17 2 1 3 4 1 8 5 3 2 1 6 5

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

< 5:

= 5:

> 5:

An example
We're looking for the 5th smallest

5 2 3 4 9 17 2 1 3 4 1 8 5 3 2 1 6 5

< 5: 2 2 1 3 2 1

= 5: 5 5 5

> 5: 34 9 17 3 4 1 8 6

Does this help us?

An example
We're looking for the 5th smallest

5 2 3 4 9 17 2 1 3 4 1 8 5 3 2 1 6 5

< 5: 2 2 1 3 2 1

= 5: 5 5 5

> 5: 34 9 17 3 4 1 8 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
    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))
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)

\( q \leftarrow \text{Partition}(A, p, r) \)

\( \text{relq} \geq q-p+1 \)

if \( k = \text{relq} \)

Return \( A[q] \)

else if \( k < \text{relq} \)

Return Selection(A, k, p, q-1)

else // \( k > \text{relq} \)

Return Selection(A, k-relq, q+1, r)

relq

\( \text{relq} = q-p+1 \)

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

relq

\( \text{relq} = q-p+1 \)

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

What is \( \text{relq} \)?
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

Selection(A, 3, 1, 8)

\[
\text{relq} = 6 - 1 + 1 = 6
\]
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, 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, 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, 1, 3, 5)

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

Selection(A, 1, 3, 5)

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

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)

\[
q \leftarrow \text{Partition}(A, p, r) \\
\text{relq} = q - p + 1 \\
\text{if } k = \text{relq} \\
\text{Return } A[q] \\
\text{else if } k < \text{relq} \\
\text{Selection}(A, k, p, q-1) \\
\text{else if } k > \text{relq} \\
\text{Selection}(A, k-\text{relq}, q+1, r)
\]
Selection(A, k, p, r)

\[ q \leftarrow \text{Partition}(A, p, r) \]
relq = \( q - p + 1 \)
if \( k = \text{relq} \)
Return A[q]
else if \( k < \text{relq} \)
Selection(A, k, p, q - 1)
else \( k > \text{relq} \)
Selection(A, k - relq, q + 1, r)

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: \( \Theta(n) \)

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

Recurrence?
\[ T(n) = T(n - 1) + \Theta(n) \]
\[ O(n^2) \]
Running time of Selection?
Worst case?
Each call to Partition only reduces our search by 1

When does this happen?
- sorted
- reverse sorted
- others...

Running time of RSelection?
Best case
- \( \Theta(n) \)
Worst case
- Still \( O(n^2) \)
- As with Quicksort, we can get unlucky

Average case?
Depends on how much data we throw away at each step

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)
Average case

We'll call a partition “good” if the pivot falls within the 25th and 75th 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

Average case

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

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.
Average case

If on average we can get a “good” partition every other time, what is the recurrence?

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

\[ T(n) = T\left(\frac{3}{4} n\right) + O(n) \]

We throw away at least \( \frac{1}{4} \) of the data roll in the cost of the “bad” partitions

Which is?

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

Selection

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

Best case: \( \Theta(n) \)

Average case: \( \Theta(n) \)
An aside…

Notice a trend?

\[ T(n) = T(n/2) + \Theta(n) \quad \Theta(n) \]

\[ T(n) = T(3/4n) + \Theta(n) \quad \Theta(n) \]

\[ f(n) = n^2 \quad f(n) = n^{\log_p a} = n^{\log_{b/p} 1} \]

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

---

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

Data structures

What is a data structure?

Way of storing data that facilitates particular operations
Data structures

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

Data structures review

List

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?)

Ordered Set

Heap

Unordered Set

Lists

get/set at index
append (add at the end of the list)
remove
add/insert

Ordered Set

insert
remove
contains
next/prev (successor/predecessor)
Heap
insert
remove
min/max

Unordered Set
insert
remove
contains