CS38
Introduction to Algorithms
Lecture 7
April 22, 2014

Outline

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

Divide and conquer

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

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

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 \( \log_b a \)?

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 \( 1, 2, \ldots, n \)
3. partition array a into “< a” and “> a” arrays
4. Random-Pivot-Quicksort(“< a’”)
5. Random-Pivot-Quicksort(“> a’”)
6. return(“< a”, a, “> a’”)

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

Quicksort with random pivot

• when is \( x_i \) compared with \( x_j \)?
  \( (i < j) \)
  – consider elements \( [x_i, x_{i+1}, x_{i+2}, \ldots, x_j, x_{j+1}, x_{j+2}, \ldots, x_n] \)
  – 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

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

we proved:

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

note: by Markov, prob. running time > 100 \( \implies \) 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: \(O(n + k \log n)\)

Claim:
Bounding the running time for general \(T(n)\)

Running time for general \(T(n)\):
- 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

\[ T(n) = T(n/2) + T(n/2+1) + \ldots + T(n-1) \]

Claim: \(T(n) \leq n + 1/n[T(n/2)+T(n/2+1)+\ldots+T(n-1)]\)

Proof: induction on n.
- assume true for \(1 \ldots n-1\)
- \(T(n) \leq n + 2/n[T(n/2)+T(n/2+1)+\ldots+T(n-1)]\)
- \(T(n) \leq n + 2/n[4(n/2)+4(n/2+1)+\ldots+4(n-1)]\)
- \(T(n) \leq n + 8/n[(3/8)n^2] < 4n.\)
Linear-time selection

Select(k; array of n elements: a)
1. pick i from \{1, 2, \ldots, n\} and partition array a into "$< a_i$" and "$> a_i$" arrays

*** guarantee that both arrays have size at most \((7/10)n\)
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 \(1/5 + 7/10 < 1\)

Linear-time selection

Median of medians example (n = 48):

April 22, 2014
CS38 Lecture 7

1. 17 27 3 38 26 14 48 20 40 8 37 40 13 8
2. 25 19 11 15 5 6 31 26 22
3. 47 41 7 44 36 43 4 46 32
4. 18 34 7 42 15 5 18 29 21
5. 28 33 35 46 40 34 29 9 33

April 22, 2014
CS38 Lecture 7

Linear-time selection

find median in each column

17 3 38 26 14 48 20 40 8 37 40 13 8
27 2 26 11 19 5 10 31 26 22
47 16 41 7 44 36 43 4 46 32
18 34 7 42 15 5 18 29 21
28 33 35 46 40 34 29 9 33

April 22, 2014
CS38 Lecture 7

Linear-time selection

find median in each column

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

April 22, 2014
CS38 Lecture 7

Linear-time selection

find median of this subset of \([n/5] = 10\)

17 3 38 26 14 48 20 40 8 37 40 13 8
27 2 26 11 19 5 10 31 26 22
47 16 41 7 44 36 43 4 46 32
10 34 7 42 15 5 18 29 21
28 33 35 46 40 34 29 9 33

April 22, 2014
CS38 Lecture 7

Linear-time selection

How many < this one?

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

April 22, 2014
CS38 Lecture 7
Linear-time selection

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

- 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

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

are larger or equal.

- 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 $n < 140$
  - $T(n) \leq T(n/5 + 1) + T(7/10 + 6) + cn$ otherwise

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

Proof: induction on $n$; base case easy

- $T(n) \leq T(n/5 + 1) + T(7/10 + 6) + cn$
- $T(n) \leq 20c(n/5 + 1) + 20c(7/10 + 6) + cn$
- $T(n) \leq 19cn + 140c \leq 20cn$ provided $n \geq 140$

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:
    
    $$T(n) = 2T(n/2) + \text{time for middle} + O(n \log n)$$

- **Claim:** time for middle only $O(n \log n)$ !!
  - **key:** we know $d = \min$ of distance between closest pair on left and distance between closest pair on right
  - **Observation:** only need to consider points within distance $d$ of the midline

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

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

- $d/2 \times d/2$ boxes

- no 2 points lie in same box (why?)

- if 2 points are within $\geq 16$ positions of each other in list sorted by y coord…
  - … then they must be separated by $\geq 3$ rows

  - implies dist. $> (3/2) \cdot d$

Closest pair in the plane

- Running time:

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