Design and Analysis of Algorithms
Part I: Divide and Conquer
Lecture 8: Selection Problem

Ke Xu and Yongxin Tong
（许可 与 童咏昕）

School of CSE, Beihang University

Outline

- Review to Divide-and-Conquer Paradigm

- Selection Problem
  - Problem Definition
  - First solution: Selection by sorting
  - A randomized divide-and-conquer algorithm
  - Analysis of the divide-and-conquer algorithm

Review to Divide-and-Conquer Paradigm

- Divide-and-conquer (D&C) is an important algorithm design paradigm.
  - **Divide**

    Dividing a given problem into two or more subproblems (ideally of approximately equal size)

  - **Conquer**

    Solving each subproblem (directly if small enough or recursively)

  - **Combine**

    Combining the solutions of the subproblems into a global solution

Review to Divide-and-Conquer Paradigm

- In Part I, we will illustrate Divide-and-Conquer using several examples:
  - Maximum Contiguous Subarray (最大子数组)
  - Counting Inversions (逆序计数)
  - Polynomial Multiplication (多项式乘法)
  - QuickSort and Partition (快速排序与划分)
  - Randomized Selection (随机化选择)
  - Lower Bound for Sorting (基于比较的排序下界)

Outline

- Review to Divide-and-Conquer Paradigm

- Selection Problem
  - Problem Definition
  - First solution: Selection by sorting
  - A randomized divide-and-conquer algorithm
  - Analysis of the divide-and-conquer algorithm

Linear Time Selection

Definition (Selection Problem)

Given a sequence of numbers $\langle a_1, \ldots, a_n \rangle$, and an integer i, $1 \leq i \leq n$, find the ith smallest element. When $i = \lceil n/2 \rceil$, it is called the median problem.

Example

Given $\langle 1, 8, 23, 10, 19, 33, 100 \rangle$, the 4th smallest element is 19.

Question

How do you solve this problem?

Outline

- Review to Divide-and-Conquer Paradigm

- Selection Problem
  - Problem Definition
  - First solution: Selection by sorting
  - A randomized divide-and-conquer algorithm
  - Analysis of the divide-and-conquer algorithm

First Solution: Selection by Sorting

- Sort the elements in ascending order with any algorithm of complexity O(n log n).
- Return the *i*th element of the sorted array.

The complexity of this solution is O(n log n)

Question

Can we do better?

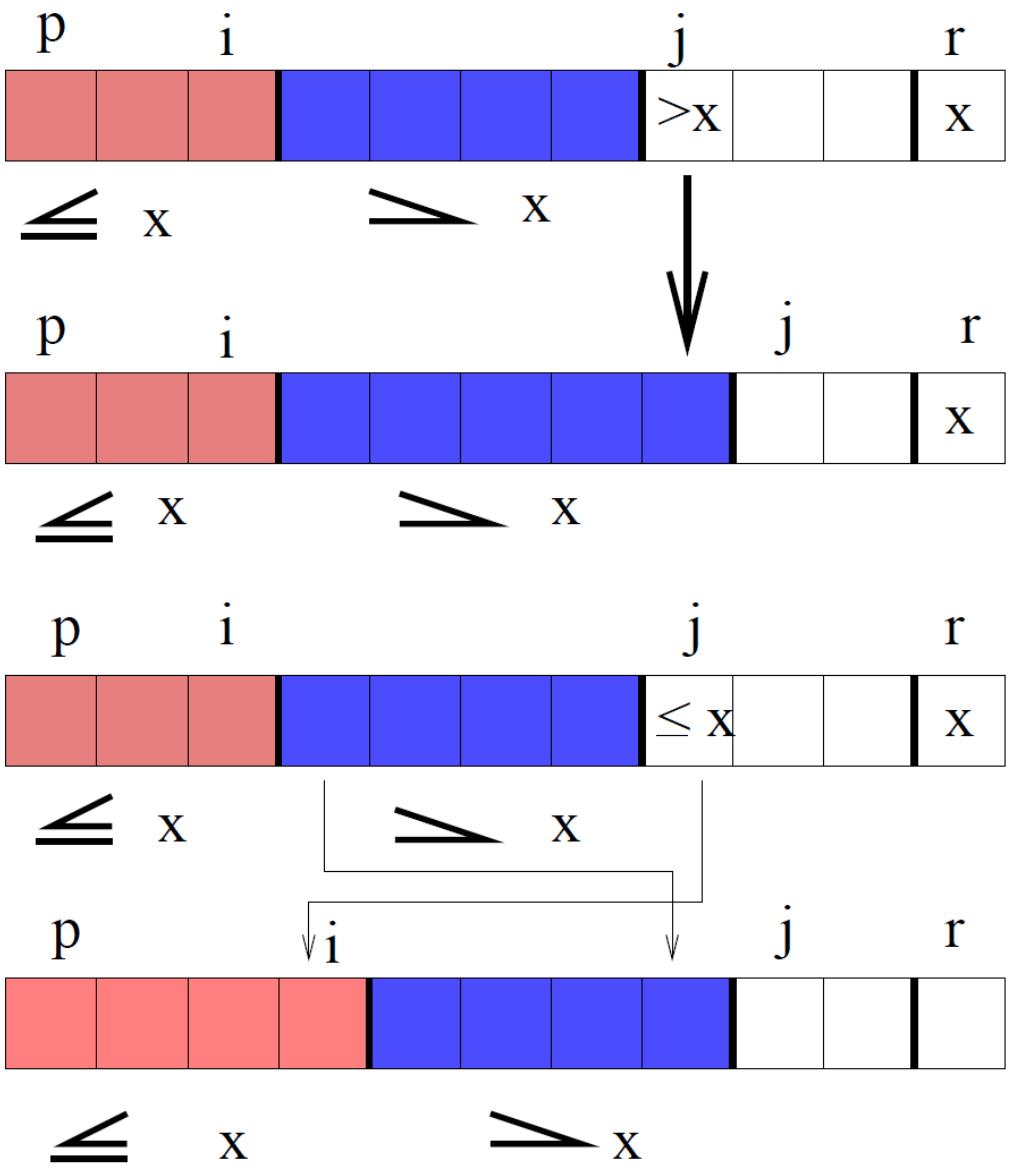
Answer: YES, but we need to recall Partition(A,p,r) used in Quicksort!

Outline

- Review to Divide-and-Conquer Paradigm

- Selection Problem
  - Problem Definition
  - First solution: Selection by sorting
  - A randomized divide-and-conquer algorithm
  - Analysis of the divide-and-conquer algorithm

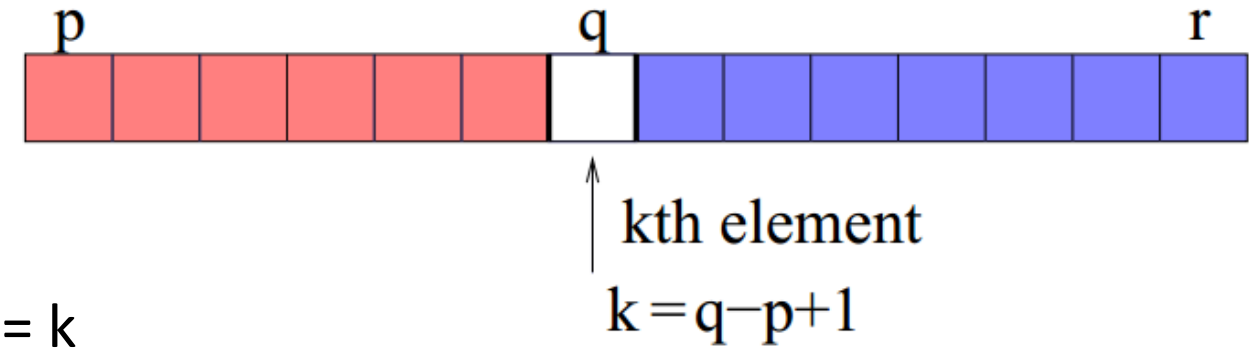
Review of Randomized-Partition (A,p,r)



Randomized-Select(A,p,r,i), 1≤i≤r−p+1

Problem: Select the *i*th smallest element in A[p..r], where 1 ≤ i ≤ r−p+1

Solution: Apply Randomized-Partition(A, p, r), getting



- i = k
  - pivot is the solution
- i < k
  - the *i*th smallest element in A[p..r] must be the *i*th smallest element in A[p..q−1]
- i > k
  - the *i*th smallest element in A[p..r] must be the (*i* − *k*)th smallest element in A[q+1..r]

If necessary, recursively call the same procedure to the subarray

Randomized-Select(A,p,r,i), 1≤i≤r−p+1

Randomized-Select(A,p,r,i)

Input: An array $\boldsymbol{A}$,the range of index $\boldsymbol{p}$,$\boldsymbol{r}$,the $\boldsymbol{i}$th smallest element that we want to select
Output: The ith smallest element $\boldsymbol{A}[\boldsymbol{i}]$
if p *is equal to* r **then**
  return $A[p]$;
end
$q \leftarrow$Randomized-Partition(A, p, r);
$k \leftarrow q - p + 1$;
if $i \leftarrow k$ **then**
  return $A[q]$;//The pivot is the answer
end
else if $i < k$ **then**
  return Randomized-Select$(A, p, q - 1, i)$;
end
else
  return Randomized-Select$(A, q + 1, r, i - k)$;
end

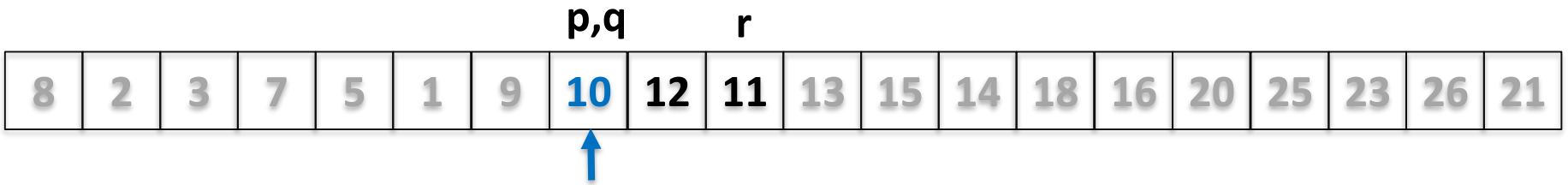
To find the ith smallest element in A[1..n], call Randomized-Select(A, 1, n, i)

Randomized Selection - Example

- Find the 8th smallest element of the following list of numbers:
  - 8 25 2 14 3 20 15 13 12 11 7 5 9 10 16 18 1 23 26 21

Randomized Selection - Example

- Select the *i*th smallest element in A[p..r], pivot is A[q], k = q-p+1.
  - i = k : pivot is the solution
  - i < k : the *i*th smallest element in A[p..q-1]
  - i > k : the (*i*-*k*)th smallest element in A[q+1..r]



$$i = 1, p = 8, r = 10$$
$$q = 8, k = 1$$

10 is the 8th smallest element of the array.

Outline

- Review to Divide-and-Conquer Paradigm

- Selection Problem
  - Problem Definition
  - First solution: Selection by sorting
  - A randomized divide-and-conquer algorithm
  - Analysis of the divide-and-conquer algorithm

Running Time of Randomized-Select(A,1,n,i)

- We let the running time on an input array A[p..r] of n elements be a random variable denoted by $T(n)$.
- The procedure RANDOMIZED-PARTITION is equally likely to return any element as the pivot.
- Therefore, for each k such that $1 \leq k \leq n$, the subarray $A[p..q]$ has k elements (all less than or equal to the pivot) with probability 1/n.
- For k = 1,2,…,n, we define indicator random variables X_k where

  $$X_k = I\{the\ subarray\ A[p..q] has\ exactly\ k\ elements\}$$

  we have

  $$E[X_k] = 1/n$$

Running Time of Randomized-Select(A,1,n,i)

- When we choose $A[q]$ as the pivot element, we will either
  - terminate immediately with the correct answer
  - recurse on the subarray $A[p..q-1]$
  - recurse on the subarray $A[q+1..r]$
- To obtain an upper bound, we assume that the *i*th element is always on the side of the partition with the greater number of elements.
- When $X_k = 1$, the two subarrays on which we might recurse have sizes $k-1$ and $n-k$.Hence,

$$T(n) \leq \sum_{k=1}^{n} X_k \cdot \big(T(\max(k-1, n-k)) + O(n)\big)$$

Running Time of Randomized-Select(A,1,n,i)

Taking expected values, we have

$$E[T(n)] \le E\left[\sum_{k=1}^{n} X_k \cdot \left(T(\max(k-1, n-k)) + O(n)\right)\right]$$

$$= \sum_{k=1}^{n} E[X_k \cdot T(\max(k-1, n-k))] + O(n)$$

$$= \sum_{k=1}^{n} E[X_k] \cdot E[T(\max(k-1, n-k))] + O(n)$$

$$= \sum_{k=1}^{n} \frac{1}{n} \cdot E[T(\max(k-1, n-k))] + O(n)$$

$\boldsymbol{E[X_k] = 1/n}$

Running Time of Randomized-Select(A,1,n,i)

- Consider the expression $\max(k-1, n-k)$,

$$\max(k-1, n-k) = \begin{cases} k-1 \ \ if\ k > \lceil n/2 \rceil \\ n-k \ if\ k \leq \lceil n/2 \rceil \end{cases}$$

- Assuming that $T(n)$ is monotonically increasing
  - If n is even, each term from $T(\lceil n/2 \rceil)$ up to $T(n-1)$ appears exactly twice in the summation
  - if n is odd, all these terms appear twice and $T(\lfloor n/2 \rfloor)$ appears once

  Thus we have

$$E[T(n)] \leq \frac{2}{n} \sum_{k=\lceil n/2 \rceil}^{n-1} E[T(k)] + O(n)$$

- It is a complicated recurrence!
- Use Guess and Induction (Substitution Method)

Running Time of Randomized-Select(A,1,n,i)

- Guess:

$$T(n) \leq c\,n, \qquad for\ all\ n$$

for some constant *c* to be figured out later.

Induction step: Assume that $T(m) \leq c\,m$ for all $m \leq n-1$.
Then try to show $T(n) \leq cn$:

$$E[T(n)] \leq \frac{2}{n}\sum_{k=\lfloor n \rfloor}^{n-1} ck + an$$

$$= \frac{2c}{n}\left(\sum_{k=1}^{n-1} k - \sum_{k=1}^{\left\lfloor \frac{n}{2} \right\rfloor - 1} k\right) + an$$

$$= \frac{2c}{n}\left(\frac{(n-1)n}{2} - \frac{(\lfloor n/2 \rfloor - 1)\lfloor n/2 \rfloor}{2}\right) + an$$

Running Time of Randomized-Select(A,1,n,i)

$$= \frac{2c}{n}\left(\frac{(n-1)n}{2} - \frac{(\lfloor n/2 \rfloor - 1)\lfloor n/2 \rfloor}{2}\right) + an$$

$$\leq \frac{2c}{n}\left(\frac{(n-1)n}{2} - \frac{(n/2-2)(n/2-1)}{2}\right) + an$$

$$= \frac{2c}{n}\left(\frac{n^2-n}{2} - \frac{n^2/4 - 3n/2 + 2}{2}\right) + an$$

$$= \frac{c}{n}\left(\frac{3n^2}{4} + \frac{n}{2} - 2\right) + an$$

$$= c\left(\frac{3n}{4} + \frac{1}{2} - \frac{2}{n}\right) + an$$

$$\leq \frac{3cn}{4} + \frac{c}{2} + an$$

$$= cn - \left(\frac{cn}{4} - \frac{c}{2} - an\right)$$

Running Time of Randomized-Select(A,1,n,i)

- In order to complete the proof, we need to show that for sufficiently large n, this last expression is at most $c \cdot n$ or, equivalently, that $\frac{cn}{4} - \frac{c}{2} - an \geq 0$.
- If $\frac{c}{4} - a > 0$, we have

$$n \geq \frac{2c}{c - 4a}$$

If we choose $c \geq 12a$, then the induction step works for $n \geq 3$.

Induction basis:

$$T(1) \leq c \cdot 1, T(2) \leq c \cdot 2$$

So if we choose $c = \max\{12, T(1), T(2)/2\}$, then the entire proof works.

Running Time of Randomized-Select(A,1,n,i)

Worst Case:

$$T(n) = n - 1 + T(n-1), T(n) = O(n^2)$$

Expected running time much better than worst case!

Randomized Quicksort vs Randomized Selection

Question

Why does Randomized Selection take $O(n)$ time while Randomized Quicksort takes $O(n \log n)$ time?

Answer:

- Randomized Selection needs to work on only 1 of the two subproblems.
- Randomize Quicksort needs to work on both of the two subproblems.

Summary of Divide-and-Conquer and Randomization

- Randomization is needed when it is not easy to divide evenly.
- Use expected case analysis for randomized algorithms.
- The running time is the expected running time for any given input, expectation is with respect to the random choices made by the algorithm internally.

谢谢

BDA
Beihang University