Design and Analysis of Algorithms

Part I: Divide and Conquer

Lecture 7: Quicksort

Ke Xu and Yongxin Tong

（许可 与 童咏昕）

School of CSE, Beihang University

Outline

- Review to Divide-and-Conquer Paradigm

- Quicksort Problem
  - Basic partition
  - Randomized partition and randomized quicksort
  - Analysis of the randomized quicksort

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

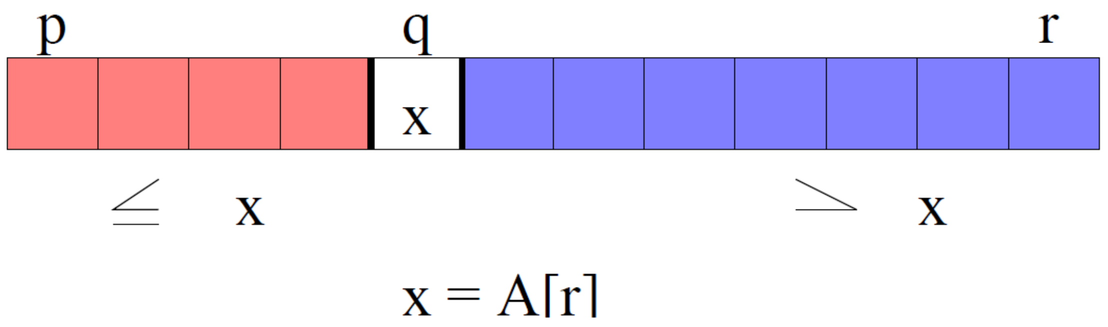
- Review to Divide-and-Conquer Paradigm

- Quicksort Problem
  - Basic partition
  - Randomized partition and randomized quicksort
  - Analysis of the randomized quicksort

Partition

- Partition
  - Given: An array of numbers
  - Partition: Rearrange the array A[p..r] in place into two (possibly empty) subarrays A[p..q-1] and A[q+1..r] such that

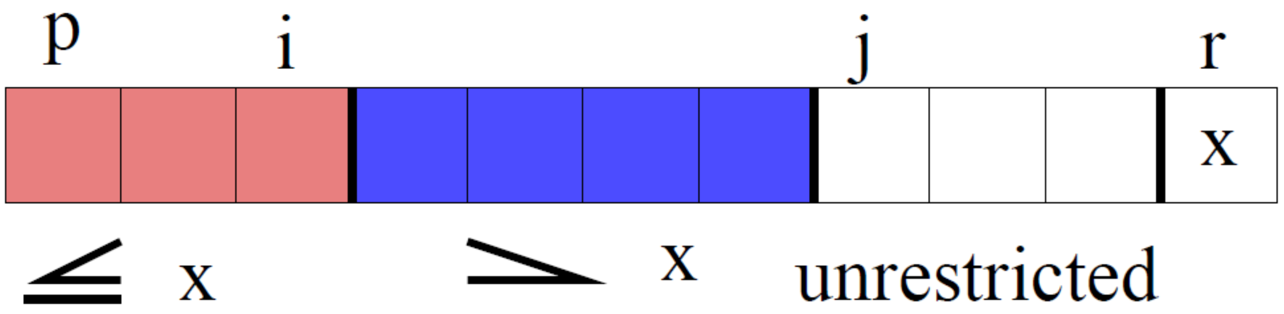
$$A[u] < A[q] < A[v] \ \ for\ any\ p \le u \le q-1\ and\ q+1 \le v \le r$$



  - x is called the pivot. Assume x = A[r]; if not, swap first
  - Quicksort works by:
    - calling partition first
    - recursively sorting A[p..q-1] and A[q+1..r]

Partition

- The idea of Partition(A, p, r)
  - Use A[r] as the pivot, and grow partition from left to right



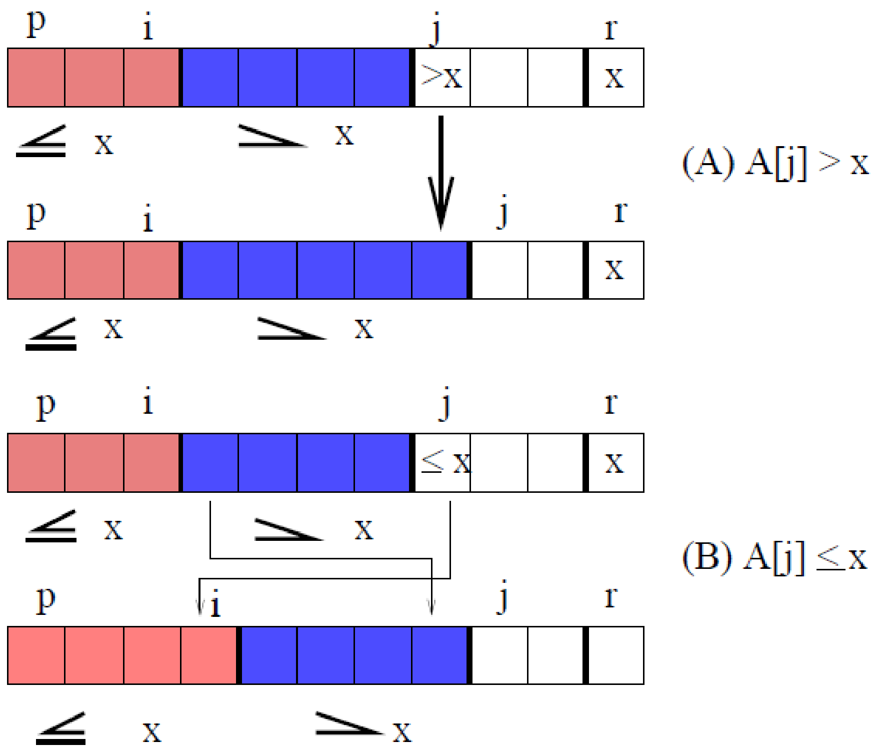
  - Initially (i, j) = (p–1, p)
  - Increase j by 1 each time to find a place for A[j]

    At the same time increase i when necessary
  - Stops when j = r

Partition

- One Iteration of the Procedure Partition
  - Increase j by 1 each time to find a place for A[j]

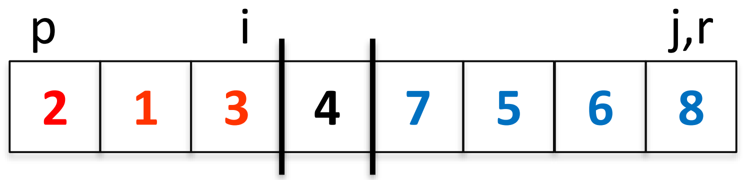
    At the same time increase i when necessary



  - Case (A): Only increase j by 1
  - Case (B): i = i + 1; A[i] ↔ A[j]; j = j + 1.

Partition-Example

- The Operation of Partition(A, p, r)



$$A[i+1] \leftrightarrow A[r]$$

Partition - Pseudocode

Partition(A,p,r)

```
Input: An array $\boldsymbol{A}$ waiting to be sorted,the range of index $\boldsymbol{p}$,$\boldsymbol{r}$
Output: Index of the pivot after partition
$x \leftarrow A[r]$;//$A[r]$ is the pivot element
$i \leftarrow p - 1$;
for $j \leftarrow p$ to $r - 1$ do
    if $A[j] \leq x$ then
        $i \leftarrow i + 1$;
        exchange $A[i]$ and $A[j]$;
    end
end
exchange$A[i+1]$ and $A[r]$;//Put pivot in position
return $i + 1$;//$q \leftarrow i + 1$
```

- Running time is O(r – p)
  - linear in the length of the array A[p..r]

Quicksort

Quicksort(A,p,r)

```
Input: An array $A$ waiting to be sorted,the range of index $p$,$r$
Output: Sorted array $A$
if $p < r$ then
    $q \leftarrow$Partition$(A, p, r)$;
    Quicksort$(A, p, q - 1)$;
    Quicksort$(A, q + 1, r)$;
end
return $A$;
```

A Divide-and-Conquer Framework

- If we could always partition the array into halves, then we have the recurrence $T(n) \leq 2T(n/2) + O(n)$, hence $T(n) = O(n \log n)$.
- However, if we always get unlucky with very unbalanced partitions, then $T(n) \leq T(n-1) + O(n)$, hence $T(n) = O(n^2)$.

Outline

- Review to Divide-and-Conquer Paradigm

- Quicksort Problem
  - Basic partition
  - Randomized partition and randomized quicksort
  - Analysis of the randomized quicksort

Randomized-Partition(A, p, r)

- Idea
  - In the algorithm Partition(A, p, r), A[r] is always used as the pivot x to partition the array A[p..r].
  - In the algorithm Randomized-Partition(A, p, r), we randomly choose an j , p ≤ j ≤ r , and use A[j] as pivot.
  - Idea is that if we choose randomly, then the chance that we get unlucky every time is extremely low.

Randomized-Partition(A, p, r)

- Pseudocode of Randomized-Partition
    - Let random(p, r) be a pseudorandom-number generator that returns a random number between p and r.

Randomized-Partition(A,p,r)

```
Input: An array $A$ waiting to be sorted,the range of index $p$,$r$
Output: A random index in $[p..j]$
$j \leftarrow$random$(p, r)$;
exchange $A[r]$ and $A[j]$;
Partition$(A, p, r)$;
return $j$;
```

Randomized-Partition(A, p, r)

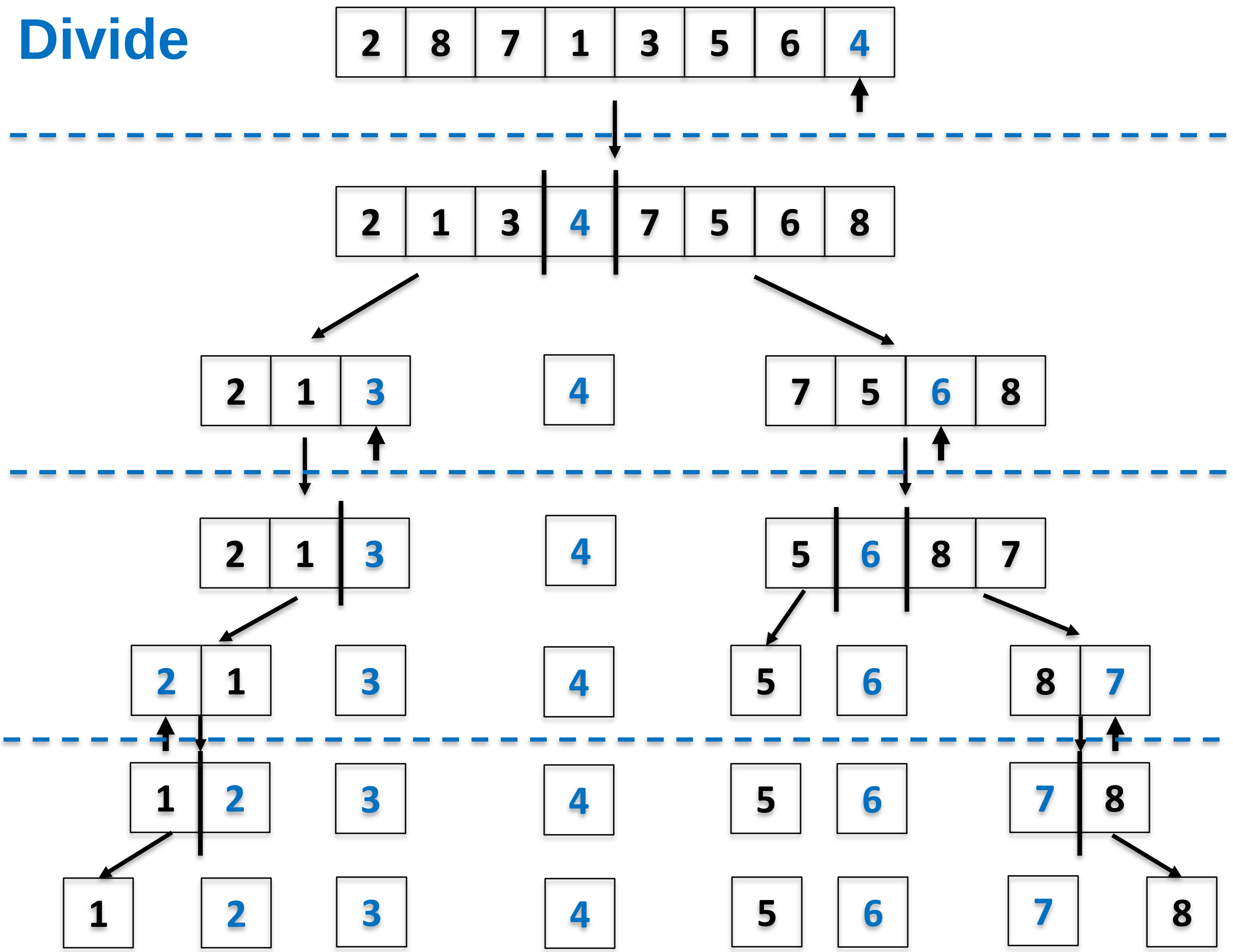
- Pseudocode of Randomized-Quicksort
  - We make use of the Randomized-Partition idea to develop a new version of quicksort.

Randomized-Quicksort(A,p,r)

```
**Input:** An array $A$ waiting to be sorted,the range of index $p$,$r$
**Output:** Sorted array $A$
if $p < r$ then
    $q \leftarrow$ Randomized-Partition($A, p, r$);
    Randomized-Quicksort($A, p, q - 1$);
    Randomized-Quicksort($A, q + 1, r$);
end
return $A$;
```

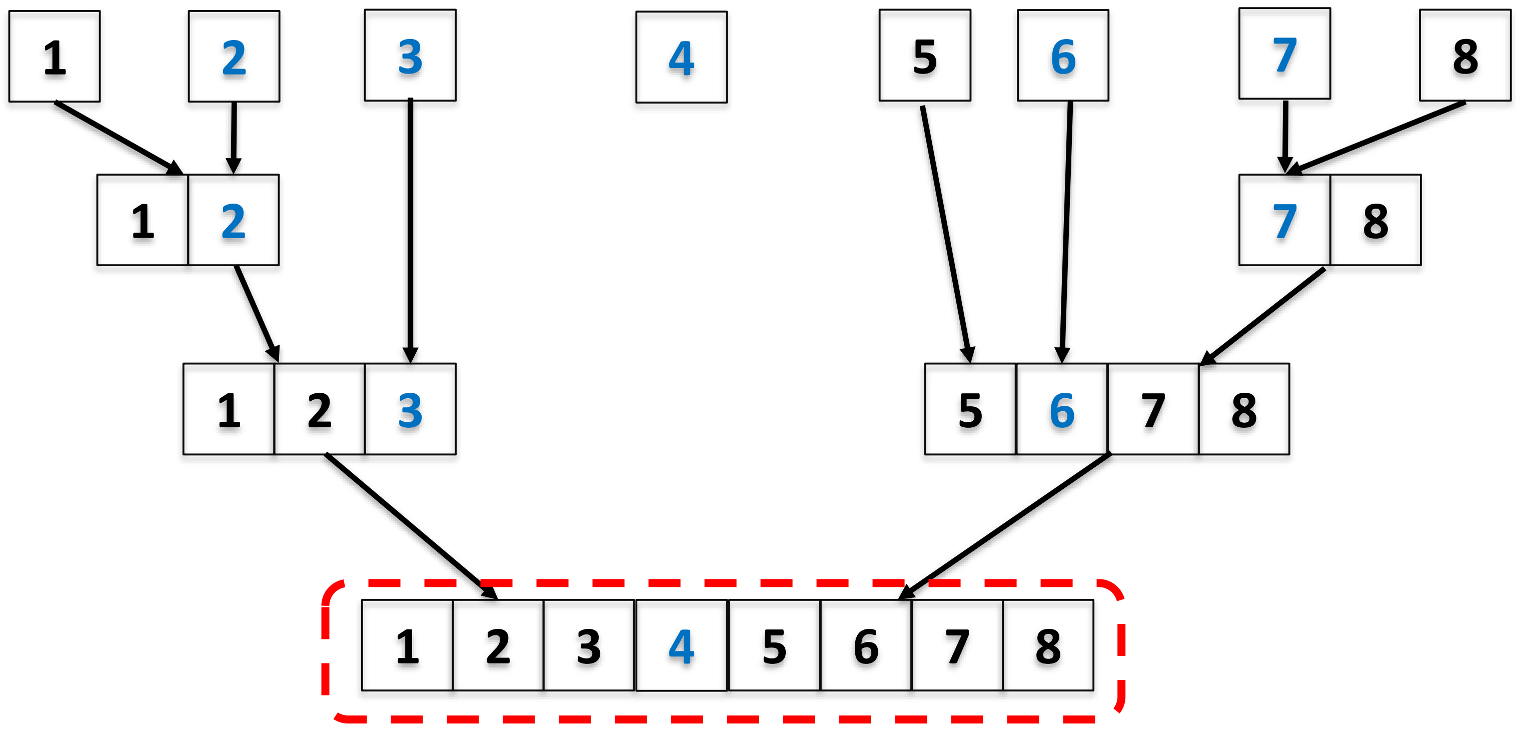
Quicksort - Example

Divide



Quicksort - Example

Conquer



Outline

- Review to Divide-and-Conquer Paradigm
- Quicksort Problem
  - Basic partition
  - Randomized partition and randomized quicksort
  - Analysis of the randomized quicksort

Running Time of the Quicksort

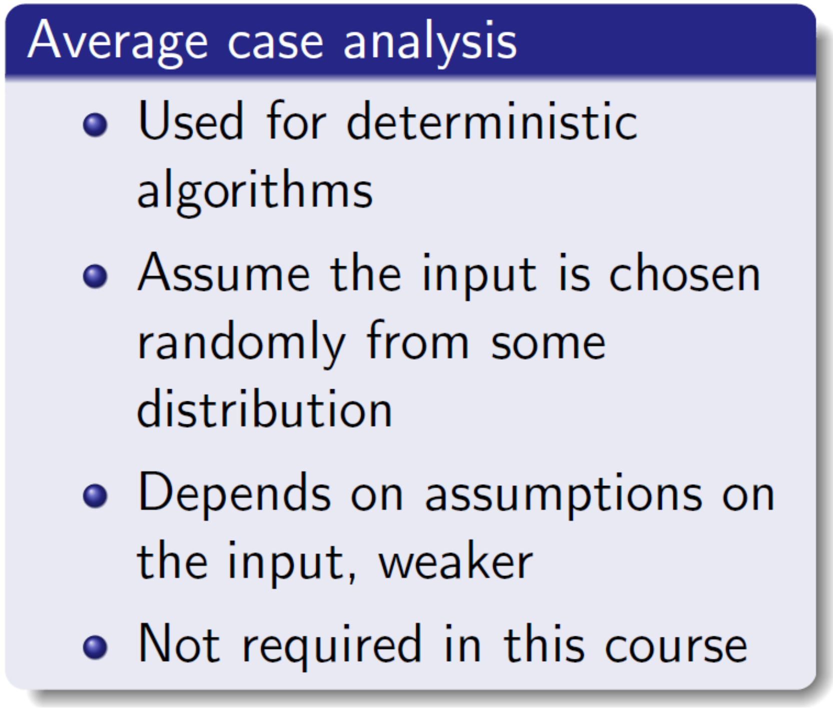
- Measuring running time:
  - The running time is dominated by the time spent in partition.
  - The running time of the partition procedure can be measured by the number of key comparisons.
- T(n): running time on array of size n.
- Recurrence: $T(n) = T(m) + T(n - m - 1) + O(n)$
- Worst Case:

$$T(n) = T(0) + T(n-1) + O(n)$$
$$T(n) = O(n^2)$$

- What inputs give worst case performance?
  - Whether performance is the worst is not determined by input.
  - An important property of randomized algorithms.
  - Worst case performance results only if the random number generator always produces the worst choice.

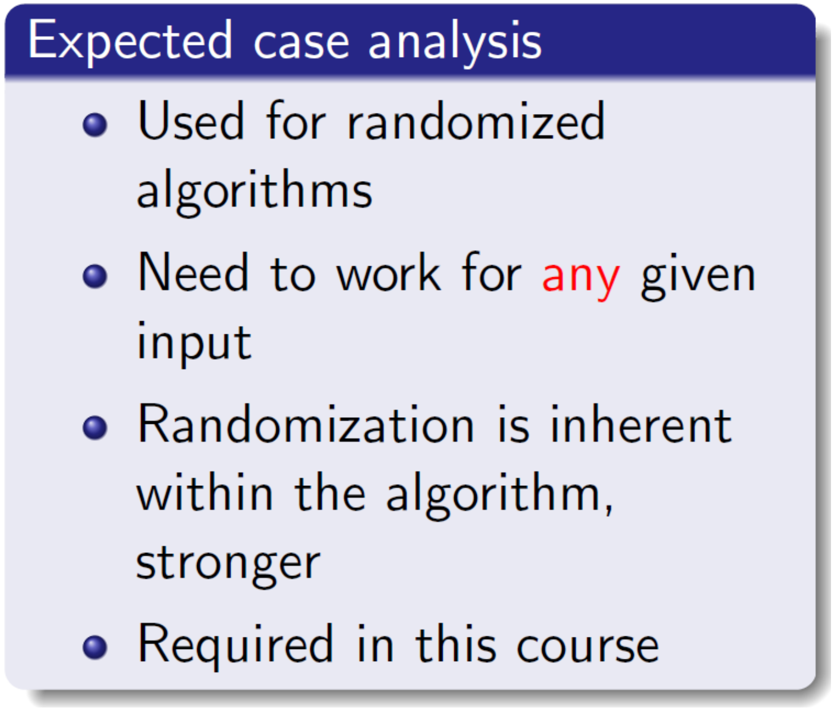
Randomized Algorithms

- Analysis for Randomized Algorithms:
  - Worst-case doesn't make sense: for any given input, the worst case is very unlikely to happen.
  - Use expected running time analysis for randomized algorithms!



Average case analysis

- Used for deterministic algorithms
- Assume the input is chosen randomly from some distribution
- Depends on assumptions on the input, weaker
- Not required in this course



Expected case analysis

- Used for randomized algorithms
- Need to work for any given input
- Randomization is inherent within the algorithm, stronger
- Required in this course

Expected Case

- Two methods to analyze the expected running time of a divide-and-conquer randomized algorithm:
  - Old fashioned: Write our a recurrence on T(n), where T(n) is the expected running time of the algorithm on an input of size n, and solve it.
    —— (Almost) always works but needs complicated maths.
  - New: Indicator variables.
    —— Simple and elegant, but needs practice to master.

Expected Case

- Two facts about key comparisons:
  - When a pivot is selected, the pivot is compared with every other elements, then the elements are partitioned into two parts accordingly
  - Elements in different partitions are never compared with each other in all operations

Expected Running Time

- Expected Running Time of Randomized-Quicksort:
  - Let $z_1 < z_2 < \cdots < z_n$ be the n elements in sorted order
  - X: total number of comparisons performed in all calls to randomized-partition
  - X_{ij} : number of comparisons between z_i and z_j
    - can only be 0 or 1

$$E[X] = E\left[\sum_{i=1}^{n-1}\sum_{j=i+1}^{n} X_{ij}\right] = \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} \mathrm{E}[X_{ij}]$$

$$= \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} [\Pr\{z_i \textit{ is compared with } z_j\} \times 1$$

$$+ \Pr\{z_i \textit{ is not compared with } z_j\} \times 0]$$

$$= \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} \Pr\{z_i \textit{ is compared with } z_j\}$$

Observations

For $1 \leq i \leq j \leq n$, let $Z_{ij} = \{z_i, z_{i+1}, \dots, z_j\}$

- remember $z_i < z_{i+1} < \cdots < z_j$

Observations:

- Partition divides an array into three segments, left, pivot, and Right.
- When z_i and z_j are first placed in DIFFERENT segments of the array by partitioning, the pivot is one of the elements in Z_{ij}
- If the pivot is either z_i or z_j
  - z_i and z_j will be compared
- If the pivot is any element in Z_{ij} other than z_i or z_j
  - z_i and z_j are not compared with each other in all randomized-partition calls

How to Find Pr{z_i is compared with z_j}?

$$
\begin{aligned}
&\Pr\{z_i \textit{ is compared with } z_j\} \\
&\quad = \Pr\{z_i \text{ or } z_j \text{ is the first pivot chosen from } Z_{ij}\} \\
&\quad = \Pr\{z_i \text{ is the first pivot chosen from } Z_{ij}\} \\
&\qquad + \Pr\{z_j \text{ is the first pivot chosen from } Z_{ij}\} \\
&\quad = \frac{1}{j-i+1} + \frac{1}{j-i+1} = \frac{2}{j-i+1}
\end{aligned}
$$

$$
E[X] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \Pr\{z_i \textit{ is compared with } z_j\} = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j-i+1}
$$

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

Note: $\sum_{k=1}^{n} \frac{1}{k} \leq \log(n)$

Hence, the expected number of comparisons is $O(n \log n)$, which is the expected running time of Randomized-Quicksort

谢谢

BDA
Beihang University