Announcements

- HW1 is slightly updated

CS6161: Design and Analysis of Algorithms (Fall 2020)

Probability Basics and Randomized Algorithms

Instructor: Haifeng Xu

Outline

- Probability Basics
- Randomized Quick Sort

Random Events

- Capture events with uncertainty
  - E.g., raining or not tomorrow, you will get A or not for CS 6161…

Definition. *Random variables* are variables with random uncertainty.

- For example, your algorithm may terminate in n steps with probability $\frac{1}{2}$ and in n^2 steps with probability $\frac{1}{2}$

Expected Time = $\frac{1}{2}(n + n^2)$

Calculating expectation is an important skill in randomized algorithm design and analysis

Expectation: Examples

Q: Your algorithm may terminate in i steps with probability $\frac{1}{n}$, where $i = 1,2,\cdots,n$. What is its expected running time?

Expected Time = $\frac{1}{n}(1 + 2 + \cdots + n)$

$$= \frac{1}{n} \times \frac{n(n+1)}{2}$$

$$= \frac{(n+1)}{2}$$

Expectation: Examples

Q: Your algorithm may terminate in i steps with probability $\frac{1}{n}$, where $i = 1,2,\cdots,n$. What is its expected running time?

- If only care about order but not exact time, using relaxation and inequalities can make your calculation much easier

$$\text{Expected Time } \leq \frac{1}{n}(n + n + \cdots + n)$$

$$= n$$

$$\text{Expected Time } \geq \frac{1}{n}(\underbrace{0 + \cdots 0}_{n/2} + \underbrace{\frac{n}{2} + \cdots \frac{n}{2}}_{n/2})$$

Expectation: Examples

Q: Your algorithm may terminate in i steps with probability $\frac{1}{n}$, where $i = 1,2,\cdots,n$. What is its expected running time?

- If only care about order but not exact time, using relaxation and inequalities can make your calculation much easier

$$\left.\begin{aligned}
\text{Expected Time } &\le \frac{1}{n}(n + n + \cdots + n) \\
&= n \\
\text{Expected Time } &\ge \frac{1}{n}(0 + \cdots 0 + \frac{n}{2} + \cdots \frac{n}{2}) \\
&= \frac{1}{n} \times \frac{n}{2} \times \frac{n}{2} = \frac{n}{4}
\end{aligned}\right\} O(n)$$

Expectation: Examples

Q: Your algorithm may terminate in $i \log i$ steps with probability $\frac{1}{n}$, where $i = 1,2,\cdots,n$. What is its expected running time?

$$\text{Expected Time} = \frac{1}{n}\sum_{i=1}^{n} i \log i$$

$$= \frac{1}{n}\sum_{i=1}^{n} \int_{i}^{i+1} i \log i \;\, dx$$

Since $i \log i = \int_{i}^{i+1} i \log i \;\, dx$

Expectation: Examples

Q: Your algorithm may terminate in $i \log i$ steps with probability $\frac{1}{n}$, where $i = 1,2,\cdots,n$. What is its expected running time?

$$\text{Expected Time} = \frac{1}{n}\sum_{i=1}^{n} i \log i$$

$$= \frac{1}{n}\sum_{i=1}^{n} \int_{i}^{i+1} i \log i \ \ dx$$

$$\leq \frac{1}{n}\sum_{i=1}^{n} \int_{i}^{i+1} x \log x \ \ dx$$

Since $i \log i \leq x \log x, \forall x \in [i, i+1]$

Expectation: Examples

Q: Your algorithm may terminate in $i \log i$ steps with probability $\frac{1}{n}$, where $i = 1,2,\cdots,n$. What is its expected running time?

$$\text{Expected Time} = \frac{1}{n}\sum_{i=1}^{n} i \log i$$

$$= \frac{1}{n}\sum_{i=1}^{n} \int_{i}^{i+1} i \log i \ \ dx$$

$$\leq \frac{1}{n}\sum_{i=1}^{n} \int_{i}^{i+1} x \log x \ \ dx$$

$$= \frac{1}{n}\int_{1}^{n+1} x \log x \ \ dx$$

Since $\int_a^b f(x)\,dx + \int_b^c f(x)\,dx = \int_a^c f(x)\,dx$

Expectation: Examples

Q: Your algorithm may terminate in $i \log i$ steps with probability $\frac{1}{n}$, where $i = 1,2,\cdots,n$. What is its expected running time?

Expected Time $= \frac{1}{n}\sum_{i=1}^{n} i \log i$

$$= \frac{1}{n}\sum_{i=1}^{n} \int_i^{i+1} i \log i \ \ dx$$

$$\leq \frac{1}{n}\sum_{i=1}^{n} \int_i^{i+1} x \log x \ \ dx$$

$$= \frac{1}{n}\int_1^{n+1} x \log x \ \ dx$$

$$= \frac{n}{2}\log n - \frac{n}{4}$$

By standard calculus (omitted)

Expectation: Examples

Q: Your algorithm may terminate in $i \log i$ steps with probability $\frac{1}{n}$, where $i = 1,2,\cdots,n$. What is its expected running time?

Expected Time $\leq \frac{n}{2}\log n - \frac{n}{4}$

Similarly

Expected Time $= \frac{1}{n}\sum_{i=1}^{n} i \log i$

$$\geq \frac{1}{n}\sum_{i=1}^{n} \int_{i-1}^{i} x \log x \;\, dx$$

$$= \frac{n-1}{2}\log(n-1) - \frac{n-1}{4}$$

$O(n \log n)$

Conditional Probabilities/Expectation

- Two random variables are correlated
  - E.g., $X = \#$ weekly hours spent on CS 6161, $Y =$ your point grade

	$X = 3$	$X = 6$
$Y = 80$	0.4	0.1
$Y = 90$	0.1	0.4

Joint Probability Table

- Marginal probability: $\Pr(Y = 90) = 0.1 + 0.4 = 0.5$
- Given $X = 6$, can more accurately estimate $\Pr(Y = 90)$
  - Condition probability: $\Pr(Y = 90 | X = 6) = 0.4/0.5 = 0.8$
  - Condition expectation: $\mathrm{E}(Y | X = 6) = 0.8 \times 90 + 0.2 \times 80 = 88$
  - Useful equations: $\Pr(X, Y) = \Pr(Y|X)\ \Pr(X)$

Exercise: Generating Random Permutations

- Input: numbers $1,2,\cdots,n$
- Output: a permutation of $1,2,\cdots,n$ generated *uniformly at random*

$\mathsf{RandomPermute}(1,2,\cdots,n)$

Exercise: Generating Random Permutations

- Input: numbers $1,2,\cdots,n$
- Output: a permutation of $1,2,\cdots,n$ generated *uniformly at random*

```
RandomPermute(1,2,⋯,n)
    List A = {1,2,⋯,n} and B = {}
```

Exercise: Generating Random Permutations

- Input: numbers $1,2,\cdots,n$
- Output: a permutation of $1,2,\cdots,n$ generated *uniformly at random*

RandomPermute$(1,2,\cdots,n)$

```
List A = {1,2,⋯,n} and B = {}
While A not empty
    Sample i from A at random
    Remove i from A
    Insert i into B

```

Exercise: Generating Random Permutations

- Input: numbers $1,2,\cdots,n$
- Output: a permutation of $1,2,\cdots,n$ generated *uniformly at random*

$\mathsf{RandomPermute}(1,2,\cdots,n)$

List $A = \{1,2,\cdots,n\}$ and $B = \{\}$

While A not empty

  Sample i from A at random

  Remove i from A

  Insert i into B

Return B

$O(n)$ time

Why B Is a Uniform Random Permutation?

- Need to prove

$$\Pr(B = \{b_1, b_2, \cdots, b_n\}) = \frac{1}{n!}, \text{ for any permutation } \{b_1, b_2, \cdots, b_n\}.$$

Proof idea: conditioning on selected numbers in the past

RandomPermute$(1,2,\cdots,n)$

List $A = \{1,2,\cdots,n\}$ and $B = \{\}$
While A not empty
  Sample i from A at random
  Remove i from A
  Insert i into B
Return B

Why B Is a Uniform Random Permutation?

- Need to prove

$$\Pr(B = \{b_1, b_2, \cdots, b_n\}) = \frac{1}{n!}$$, for any permutation $\{b_1, b_2, \cdots, b_n\}$.

Proof

- $\Pr($first sampled number is $b_1) = 1/n$
- $\Pr($second sampled number is $b_2 \mid b_1 \text{ removed}) = 1/(n-1)$
- . . .
- $\Pr(i'\text{th}$ sampled number is $b_i \mid \text{previous numbers}) = 1/(n-i+1)$

Conditional probability equality implies

$$\Pr(B = \{b_1, b_2, \cdots, b_n\}) = \frac{1}{n} \cdot \frac{1}{n-1} \cdots \frac{1}{1}$$

Outline

- Probability Basics
- Randomized Quick Sort

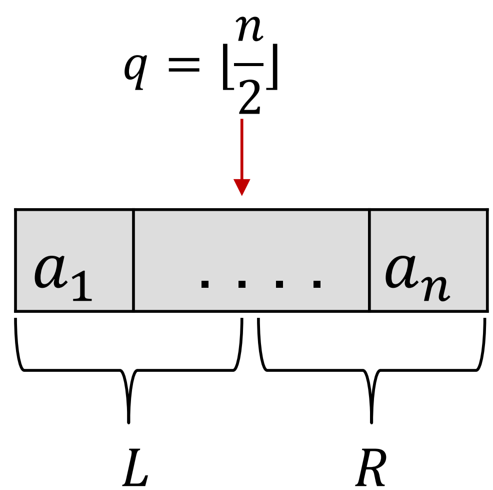
QuickSort: Another D-and-C Algorithm

Recall *MergeSort*

(1) Divide into L and R → easy

(2) Sort L and R

(3) Merge them into a globally sorted sequence → $O(n)$



QuickSort uses a smarter way for partition in $O(n)$ time, but does not require merge

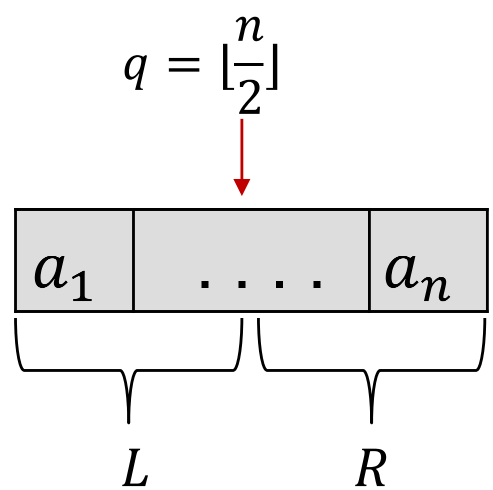
QuickSort: **Another D-and-C Algorithm**

Recall *MergeSort*

(1) Divide into L and R → $O(n)$

(2) Sort L and R

~~(3) Merge them into a globally sorted sequence~~



After Step (2), the array will be automatically sorted

QuickSort uses a smarter way for partition in $O(n)$ time, but does not require merge

How to Divide in QuickSort?

- Pick any "pivot" element q, e.g., $q = a_n$
- Divide input into L and R side based on whether elements are smaller or larger than q

L

a_1	a_2	a_3	. . . .	a_n

$> q?$ Yes

R a_1

How to Divide in QuickSort?

- Pick any "pivot" element q, e.g., $q = a_n$
- Divide input into L and R side based on elements are smaller or larger than q

L $\quad\quad a_1$

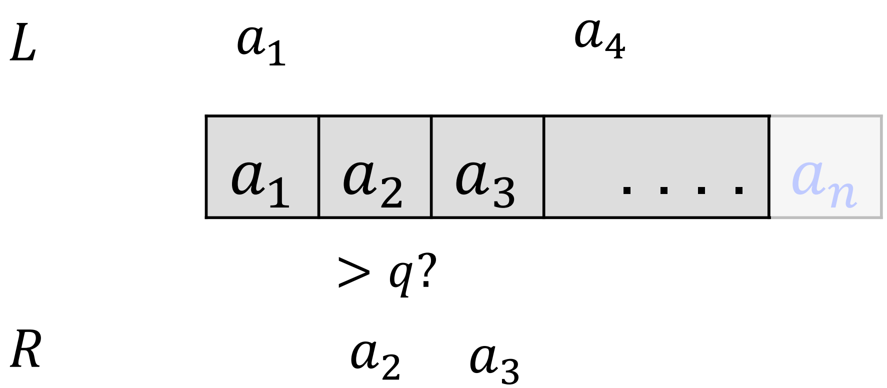
a_1	a_2	a_3	. . . .	a_n

$> q?$ No

R

How to Divide in QuickSort?

- Pick any "pivot" element q, e.g., $q = a_n$
- Divide input into L and R side based on elements are smaller or larger than q



Afterwards, sort L and R separately, entire array is automatically sorted after inserting q between L and R

Pseudo-Code

QuickSort-Deterministic$(a_1, \cdots, a_n)$

$(L, R) =$ Partition$(\{a_1, \cdots, a_n\}, q)$

$L =$ QuickSort-Deterministic(L)

$R =$ QuickSort-Deterministic(R)

Return (L, q, R)

Running Time Analysis

QuickSort-Deterministic$(a_1, \cdots, a_n)$

$(L, R) =$ Partition$(\{a_1, \cdots, a_n\}, q)$ $\longrightarrow$ $O(n)$

$L =$ QuickSort-Deterministic(L) $\longrightarrow$ $T(i)$

$R =$ QuickSort-Deterministic(R) $\longrightarrow$ $T(n - 1 - i)$

Return (L, q, R)

$i =$ length of L really depends on our choice of pivot q …

Running Time Analysis

$$T(n) = O(n) + T(i) + T(n - 1 - i)$$

$i =$ length of L really depends on our choice of pivot q …

- If $q = a_n$ is always the last element, what is the worst-case running time?
- A bad instance is to sort $n, n-1, n-2, \cdots, 1$
  - $i = 0$ always since no element smaller than the right most

$$\begin{aligned} T(n) &= O(n) + T(n-1) + T(0) \\ &= O(n) + [O(n-1) + T(n-2) + T(0)] + T(0) \\ &\quad \dots \\ &= O(n^2) \end{aligned}$$

See your first HW

Running Time Analysis

$$T(n) = O(n) + T(i) + T(n-1-i)$$

$i =$ length of L really depends on our choice of pivot q …

- What about q being the first element? Middle element?
  - Does not work neither
- There is a sophisticated way of choosing q that can make it work, but it turns out that randomization is a much easier choice

That is, simply pick q from $a_1, \cdots, a_n$ uniformly at random!

- Intuition: when worst cases make your algorithm bad, you can use randomization to "escape from" worst cases
- *Common in algorithm analysis: Yao's principle, minimax theory, zero-sum games*

Randomized Quick Sort

QuickSort-Randomized$(a_1, \cdots, a_n)$

Pick $q \in \{a_1, \cdots, a_n\}$ uniformly at random

$(L, R) =$ Partition$(\{a_1, \cdots, a_n\}, q)$ $\longrightarrow O(n)$

$L =$ QuickSort-Deterministic(L) $\longrightarrow T(i)$

$R =$ QuickSort-Deterministic(R) $\longrightarrow T(n-1-i)$

Return (L, q, R)

- $T(n)$ now is also random, depending on our choice of q
- Expected running time

$$T(n) = O(n) + \frac{1}{n}\sum_{i=0}^{n-1}[T(i) + T(n-1-i)]$$

Computing $T(n)$

$$T(n) = c_1 n + \frac{1}{n}\sum_{i=0}^{n-1}[T(i) + T(n-1-i)]$$

- Idea 1: guess $T(n) \le c(n+1)\log(n+1)$
- Prove by induction
  - If our guess is true for $i < k$, then

$$T(k) = c_1 k + \frac{1}{k}\sum_{i=0}^{k-1}[T(i) + T(k-1-i)]$$

Computing $T(n)$

$$T(n) = c_1 n + \frac{1}{n}\sum_{i=0}^{n-1}[T(i) + T(n-1-i)]$$

- Idea 1: guess $T(n) \leq c(n+1)\log(n+1)$
- Prove by induction
  - If our guess is true for $i < k$, then

$$T(k) = c_1 k + \frac{1}{k}\sum_{i=0}^{k-1}[T(i) + T(k-1-i)]$$
$$\leq c_1 k + \frac{1}{k}\sum_{i=1}^{k}[c\ i\log i + c(k-i)\log(k-i)]$$

Plug in induction hypothesis

Computing $T(n)$

$$T(n) = c_1 n + \frac{1}{n}\sum_{i=0}^{n-1}[T(i) + T(n-1-i)]$$

- Idea 1: guess $T(n) \le c(n+1)\log(n+1)$
- Prove by induction
  - If our guess is true for $i < k$, then

$$T(k) = c_1 k + \frac{1}{k}\sum_{i=0}^{k-1}[T(i) + T(k-1-i)]$$

$$\le c_1 k + \frac{1}{k}\sum_{i=1}^{k}[c\ i\log i + c(k-i)\log(k-i)]$$

$$\le c_1 k + \frac{2}{k}\sum_{i=1}^{k} c\ i\log i$$

Rearrange sums

Computing $T(n)$

$$T(n) = c_1 n + \frac{1}{n}\sum_{i=0}^{n-1}[T(i) + T(n-1-i)]$$

- Idea 1: guess $T(n) \leq c(n+1)\log(n+1)$
- Prove by induction
  - If our guess is true for $i < k$, then

$$\begin{aligned}
T(k) &= c_1 k + \frac{1}{k}\sum_{i=0}^{k-1}[T(i) + T(k-1-i)] \\
&\leq c_1 k + \frac{1}{k}\sum_{i=1}^{k}[c\ i\log i + c(k-i)\log(k-i)] \\
&\leq c_1 k + \frac{2}{k}\sum_{i=1}^{k} c\ i\log i \\
&\leq c_1 k + \frac{2}{k} c\left(\frac{k^2}{2}\log k - \frac{k^2}{4}\right)
\end{aligned}$$

From our calculations in Part 1

Computing $T(n)$

$$T(n) = c_1 n + \frac{1}{n}\sum_{i=0}^{n-1}[T(i) + T(n-1-i)]$$

- Idea 1: guess $T(n) \le c(n+1)\log(n+1)$
- Prove by induction
  - If our guess is true for $i < k$, then

$$\begin{aligned}
T(k) &= c_1 k + \frac{1}{k}\sum_{i=0}^{k-1}[T(i) + T(k-1-i)] \\
&\le c_1 k + \frac{1}{k}\sum_{i=1}^{k}[c\ i\log i + c(k-i)\log(k-i)] \\
&\le c_1 k + \frac{2}{k}\sum_{i=1}^{k} c\ i\log i \\
&\le c_1 k + \frac{2}{k}c\left(\frac{k^2}{2}\log k - \frac{k^2}{4}\right) \\
&= ck\log k + \left(c_1 - \frac{c}{2}\right)k
\end{aligned}$$

Algebraic manipulations

Computing $T(n)$

$$T(n) = c_1 n + \frac{1}{n}\sum_{i=0}^{n-1}[T(i) + T(n-1-i)]$$

- Idea 1: guess $T(n) \le c(n+1)\log(n+1)$
- Prove by induction
  - If our guess is true for $i < k$, then

$$\begin{aligned}
T(k) &= c_1 k + \frac{1}{k}\sum_{i=0}^{k-1}[T(i) + T(k-1-i)] \\
&\le c_1 k + \frac{1}{k}\sum_{i=1}^{k}[c\ i\log i + c(k-i)\log(k-i)] \\
&\le c_1 k + \frac{2}{k}\sum_{i=1}^{k} c\ i\log i \\
&\le c_1 k + \frac{2}{k} c\left(\frac{k^2}{2}\log k - \frac{k^2}{4}\right) \\
&= ck\log k + \left(c_1 - \frac{c}{2}\right)k \\
&\le c(k+1)\log(k+1) \qquad \text{By picking } c > 2c_1
\end{aligned}$$

Computing $T(n)$

$$T(n) = c_1 n + \frac{1}{n}\sum_{i=0}^{n-1}[T(i) + T(n-1-i)]$$

- Idea 1: guess $T(n) \le c(n+1)\log(n+1)$
- Prove by induction
  - If our guess is true for $i < k$, then

$$T(k) = c_1 k + \frac{1}{k}\sum_{i=0}^{k-1}[T(i) + T(k-1-i)]$$

Not the most ideal as we have to guess the correct answer…

$$\le c_1 k + \frac{2}{k}\sum_{i=1}^{k} c\, i \log i$$

$$\le c_1 k + \frac{2}{k} c\left(\frac{k^2}{2}\log k - \frac{k^2}{4}\right)$$

$$= ck\log k + \left(c_1 - \frac{c}{2}\right)k$$

$$\le c(k+1)\log(k+1)$$

By picking $c > 2c_1$

Computing $T(n)$

$$T(n) = c_1 n + \frac{1}{n}\sum_{i=0}^{n-1}[T(i) + T(n-1-i)]$$

- Idea 2: direct calculation by examining *whether* a_i, a_j *are ever compared*
- Let $b_1 \le b_2 \le \cdots \le b_n$ be the sorted sequence
- Random var $X_{ij} = 1$, if b_i, b_j are ever compared; and 0 otherwise

Running time = $\sum_{i,j:\, i \neq j} X_{ij}$

Computing $T(n)$

$$T(n) = c_1 n + \frac{1}{n}\sum_{i=0}^{n-1}[T(i) + T(n-1-i)]$$

- Idea 2: direct calculation by examining *whether a_i, a_j are ever compared*
- Let $b_1 \le b_2 \le \cdots \le b_n$ be the sorted sequence
- Random var $X_{ij} = 1$, if b_i, b_j are ever compared; and 0 otherwise

$$\mathbf{E}(\text{Running time}) = \mathbf{E}[\textstyle\sum_{i,j:\, i\neq j} X_{ij}]$$

$$= \textstyle\sum_{i,j:\, i\neq j} \mathbf{E}(X_{ij})$$

By **linearity of expectation**

Computing $T(n)$

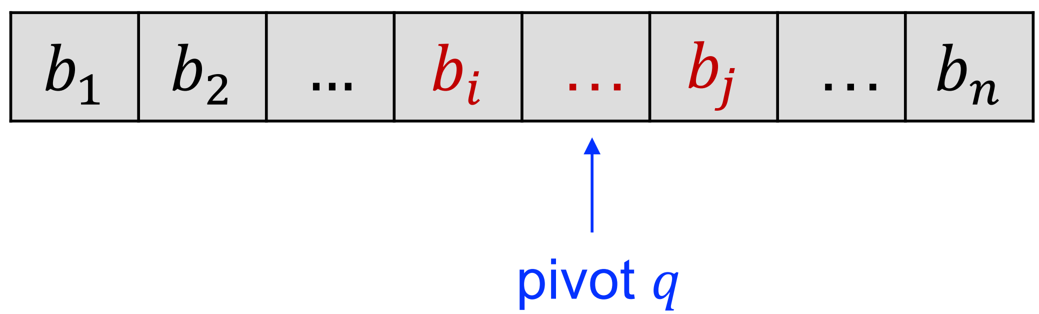
$$T(n) = c_1 n + \frac{1}{n} \sum_{i=0}^{n-1} [T(i) + T(n-1-i)]$$

- Idea 2: direct calculation by examining whether a_i, a_j are ever compared
- Let $b_1 \leq b_2 \leq \cdots \leq b_n$ be the sorted sequence
- Random var $X_{ij} = 1$, if b_i, b_j are ever compared; and 0 otherwise

$$\begin{aligned} \mathbf{E}(\text{Running time}) &= \mathbf{E}[\textstyle\sum_{i,j:\, i \neq j} X_{ij}] \\ &= \textstyle\sum_{i,j:\, i \neq j} \mathbf{E}(X_{ij}) \\ &= \textstyle\sum_{i,j:\, i \neq j} \mathbf{Pr}(b_i, b_j \text{ are compared}) \end{aligned}$$

$\Pr(b_i, b_j \text{ are compared})$

- First of all, some pairs are indeed not compared



- b_i, b_j will not be compared
  - They are compared to q, after which they are separate forever

$\Pr(b_i, b_j$ are compared)

- First of all, some pairs are indeed not compared

b_1	b_2	…	b_i	…	b_j	…	b_n

pivot q

- b_i, b_j will be compared

$\Pr(b_i, b_j$ are compared)

- First of all, some pairs are indeed not compared

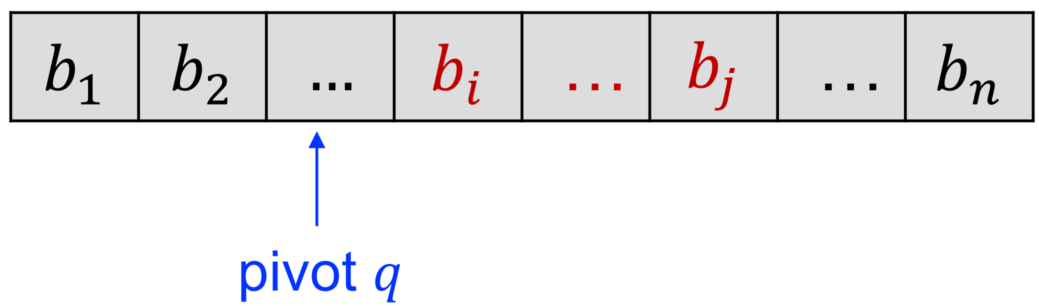
b_1	b_2	...	b_i	...	b_j	...	b_n

pivot q

- b_i, b_j will be also compared

$\Pr(b_i, b_j$ are compared$)$

- First of all, some pairs are indeed not compared



- <u>Cannot tell</u> – depending on what happens later when we sort the right side of the pivot q
- So, what really matters is the first picked pivot within $b_i, \ldots, b_j$ is b_i/b_j or any number at the middle

$$\Rightarrow \Pr\left(b_i, b_j \text{ are compared}\right) = \frac{2}{j - i + 1}$$

Back to Computing $T(n)$

$\mathbf{E}$(Running time) = $\sum_{i,j:\, i \neq j} \mathbf{Pr}(b_i, b_j$ are compared)

= $\sum_{j>i} \frac{2}{j-i+1}$

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

Counting how many i, j pairs have gap k for $k = 1, \cdots, n-1$

Back to Computing $T(n)$

$\mathbf{E}$(Running time) = $\sum_{i,j:\, i\neq j} \mathbf{Pr}(b_i, b_j$ are compared)

$$= \sum_{j>i} \frac{2}{j-i+1}$$

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

$$= \Theta(n \log n)$$

Standard calculation (omitted)

Worst-Case vs Average-Case Analysis

- Typically we do worst-case analysis
  - E.g., QuickSort-Deterministic has worst run time n^2
- Randomization can help to "interpolate" between bad and good instance, leading to better expected time
  - Randomness is introduced by algorithm designer
- Alternatively, can think of input as random instead of worst case
  - Why? Because as algorithm designer, you can preprocess input by randomize it first
- Randomization is a powerful technique in Algo design

Thank You

Haifeng Xu

University of Virginia

hx4ad@virginia.edu