Analysis and Design of Algorithms

CS3230

Week 5

Randomized algorithm

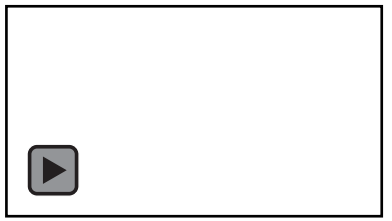
Ken Sung

Diptarka Chakraborty

Question 1

Suppose we are given n strings which has at most 10 characters each. Which of the following method is asymptotically the fastest?

- Radix Sort
- Merge Sort
- Quicksort



Solution

Answer: Radix sort.

Merge sort takes $\Theta(n \lg n)$ in the worst case.

Randomized quicksort takes $\Theta(n \lg n)$ expected time.

Since the range is a constant, radix sort takes $O(n)$ time.

Question 2

Let $A(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_n x^n$
Let $B(x) = b_0 + b_1 x + b_2 x^2 + \ldots + b_n x^n$.
Assume addition and multiplication of two numbers take O(1) time.
The coefficients of $C(x) = A(x) \cdot B(x)$ can be computed in $O(n^2)$ time.

Show how to compute the coefficients of $C(x) = A(x) \cdot B(x)$ in time $O\left(n^{\log_2 3}\right)$.

Question 2: Solution

Write

$$A(x) = x^{n/2}A_1(x) + A_2(x)$$
$$B(x) = x^{n/2}B_1(x) + B_2(x)$$

where A_1, A_2, B_1, B_2 are degree $\leq n/2$. Then:

$$C(x) = x^n A_1(x)B_1(x) + x^{n/2}A_1(x)B_2(x) + x^{n/2}A_2(x)B_1(x) + A_2(x)B_2(x)$$

This problem can be solved:

1. By recursion, compute the coefficients of $A_1(x)B_1(x)$, $A_1(x)B_2(x)$, $A_2(x)B_1(x)$ and $A_2(x)B_2(x)$.
2. From the coefficients of the 4 polynomials $x^n A_1(x)B_1(x)$, $x^{n/2}A_1(x)B_2(x)$,$x^{n/2}A_2(x)B_1(x)$ and $A_2(x)B_2(x)$, obtain the coefficients of $C(x)$.

Then, $T(n) = 4T\left(\frac{n}{2}\right) + O(n)$. Applying Master, $T(n) = O(n^2)$.

Question 2: Solution

So, it's enough to evaluate A_1B_1, A_2B_2 and $(A_1 + A_2)(B_1 + B_2)$, because

$A_1B_2 + A_2B_1 = (A_1 + A_2)(B_1 + B_2) - A_1B_1 - A_2B_2$.

Then, $T(n) = 3T\left(\frac{n}{2}\right) + O(n)$. Applying Master, $T(n) = O\left(n^{\log_2 3}\right)$.

Admin

- Prog1 is available in codecrunch
  - Deadline is 5 Mar (Week 7)

- Mid-term test:
  - Time: 7 Mar 2020 (Sat) 11:00am – 01:00pm.
  - Venue: MPSH2C
  - Open book
- Due to the outbreak, we will have other arrangement.

Admin

Consultation hours:

- Ken Sung: Friday 2-3pm (COM2-02-06)
- Diptarka: Tuesday 11:00-12:00 (COM2-03-17)
- Wei Liang (ganweiliang@u.nus.edu): Monday 13:00-1400 (AS6-04-01)
- Eldon Chung (eldon.chung@u.nus.edu): Wednesday 14:00-15:00 (COM1-B1-12)
- Govind Venugopalan (gv94@u.nus.edu): Wednesday 16:00-17:00 (COM1-01-20 Programming Lab 6)
- Tran Tan Phat (e0196695@u.nus.edu): Wednesday 16:00-17:00 (COM1 02-15)
- Joshua Casey Darian (joshuac@comp.nus.edu.sg): Thursday 13:00-14:00 (COM1 #01-18 MR5)
- Le Quang Tuan (e0313526@u.nus.edu): Monday 14:00-15:00 (COM1 02-15)
- Zhang Dongping (e0427788@u.nus.edu)

Randomized Algorithms

- An algorithm is called **randomized** if its behavior is determined not only by its input but also by values produced by a random-number generator.

Two types of randomized algorithms

- **Monte Carlo Algorithm**: Randomized algorithm that gives the correct answer with probability $1 - o(1)$ (”high probability”), but the runtime bounds hold deterministically.

- **Las Vegas Algorithm**: Randomized algorithm that always gives the correct answer, but the runtime bounds depend on the random numbers.

- Both types of algorithms use random number generator!

Average running time versus Expected running time

- For non-randomized algorithm, the running time depends on the input.
- If we know the distribution of the input, we can find the **average running time** of the algorithm.

- The running time of a randomized algorithm depends on the random number generator.
- For the same input, the running time of a randomized algorithm can be different depends on the random numbers.
- The “average” running time of a randomized algorithm over all possible random numbers is called the **expected running time**.

Question 3

- Consider three algorithms:
  - **Standard quick sort**: Choose the first element as pivot. Then, partition the elements into two subarrays. Recursively sort the two subarrays.
  - **Finding π**: Randomly sample (x,y) n times with x and y uniformly distributed in [-1, 1] and count the fraction satisfying $x^2+y^2 \leq 1$. Multiply the fraction by 4 and use it as the estimate for π.
  - **Random search**: Given a sorted array A[1..n]. You aim to find x in A. You randomly choose an index i and compare A[i] with x. If A[i]=x, found. If A[i]<x, recursively search A[i+1..n]. If A[i]>x, recursively search A[1..i-1].

- Determine if they are (1) Monte Carlo Algorithm, (2) Las Vegas Algorithm or (3) non-randomized algorithm.

Answer

- Standard quick sort is a non-randomized algorithm.
  - Its worst case running time is $\Theta(n^2)$
  - Its best case running time is $\Theta(n)$
  - Its **average case** running time is $\Theta(n \log n)$
  - Guarantee to find the exact solution. Running time is **dependent** on input.
- Finding π is a Monte Carlo algorithm.
  - Its running time is $\Theta(n)$.
  - However, it only reports an approximate π.
- Random search is a Las Vegas algorithm.
  - Its worst case running time is $\Theta(n)$
  - Its **expected** running time is $\Theta(\log n)$.
  - Guarantee to find the exact solution. Running time **depends** on the random numbers and is **independent** on input.

Aim of this lecture

- This lecture learns how to analyze the expected running time of a Las Vegas algorithm.

Revision on Probability

Axioms of Probability

Sample space *S*: a set whose elements are called **elementary events**.

- For throwing a dice, the sample space is $S = \{1, 2, 3, 4, 5, 6\}$ consisting of the 6 possible outcomes which are the elementary events

An **event** is a subset of the sample space *S*.

- The event of throwing an even number is $A = \{2, 4, 6\}$

Probability Distribution

A **probability distribution Pr[]** is a mapping from events to real number satisfying

- $\Pr[A] \geq 0$ for any event A
- $\Pr[S] = 1$
- $\Pr[A \cup B] = \Pr[A] + \Pr[B]$ for any two mutually exclusive events A and B

Example: A uniform distribution is a reasonable model for the throw of a fair dice

- Pr[i] = 1/6 for i = 1, ... , 6
- If A is the set of even dice throws
  Pr[A] = Pr[2] + Pr[4] + Pr[6] = 3/6
  as they are mutually exclusive

A and B are not mutually exclusive

- When A and B are not **mutually exclusive** (i.e. $A \cap B \neq \Phi$), we have
  - $\Pr\{A \cup B\} = \Pr\{A\} + \Pr\{B\} - \Pr\{A \cap B\}$

- Example: If A is the set of even dice throws (A = {2, 4, 6}) and B is the set of dice throws whose outcome is greater than 3 (B = {4, 5, 6})
- A and B are not mutually exclusive in this case ($A \cap B$ = {4, 6})

$\Pr\{A \cup B\} = \Pr\{A\} + \Pr\{B\} - \Pr\{A \cap B\}$
$= \Pr\{2, 4, 6\} + \Pr\{4, 5, 6\} - \Pr\{4, 6\}$
$= 3/6 + 3/6 - 2/6 = 4/6$

Independence

Two events are **independent** if

$$\Pr[A \cap B] = \Pr[A] \cdot \Pr[B]$$

Example of independent

Example: Suppose we flip two fair coins. Let A_1 be the event that the first coin is head, and A_2 be the event that the second coin is head.

$$\Pr\{A_1\} = 1/2$$

$$\Pr\{A_2\} = 1/2$$

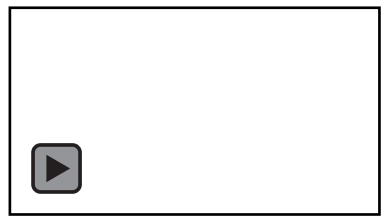
$$\Pr\{A_1 \cap A_2\} = 1/2 \times 1/2 = 1/4$$

Question 4

Suppose we randomly select a pair of numbers from $\{1, \ldots, n\}$.

The event that 1 is in the selection is independent of the event that 2 is in the selection.

- True
- False



Question 4: Solution

False!

$$\frac{1}{\binom{n}{2}} \neq \frac{n-1}{\binom{n}{2}} \cdot \frac{n-1}{\binom{n}{2}}$$

Pr(the pair we selected is (1,2))

Pr(1 is in the pair we selected)

Pr(2 is in the pair we selected)

Conditional Probability

The **conditional probability** of an event *A* given another event *B* is defined as

$$\Pr[\, A \mid B \,] = \frac{\Pr[A \cap B]}{\Pr[B]}$$

whenever $\Pr[B] \neq 0$.

Bayes's Theorem

- Bayes's theorem can be used for computing conditional probability.

$$\Pr\{A|B\} = \frac{\Pr\{A\}\Pr\{B|A\}}{\Pr\{B\}}$$

$$= \frac{\Pr\{A\}\Pr\{B|A\}}{\Pr\{A\}\Pr\{B|A\} + \Pr\{\bar{A}\}\Pr\{B|\bar{A}\}}$$

$$\Pr\{A|B\} = \frac{\Pr\{A\}\Pr\{B|A\}}{\Pr\{B\}}$$
$$= \frac{\Pr\{A\}\Pr\{B|A\}}{\Pr\{A\}\Pr\{B|A\} + \Pr\{\bar{A}\}\Pr\{B|\bar{A}\}}$$

Example:

Suppose we have one fair coin and one biased coin that always give heads.

Let *A* be the event of choosing a biased coin.

Let *B* be the event that the chosen coin gives two heads in two consecutive tosses.

We have:

$$\Pr\{A\} = 1/2, \Pr\{B|A\} = 1, \Pr\{\bar{A}\} = 1/2, \Pr\{B|\bar{A}\} = 1/4$$

$$\Pr\{A|B\} = \frac{(1/2)\cdot 1}{(1/2)\cdot 1 + (1/2)(1/4)} = 4/5$$

Random Variables

A random variable X is a function that maps the sample space S to real numbers.

The function $f(x) = \Pr\{X = x\}$ is the probability density function of X.

Example: Rolling a pair of dice. Sample space contains 36 elementary events. Let X be *max* of the two values shown on the dice.

Then $\Pr\{X=3\}=5/36$: elementary events (1,3), (2,3), (3,3), (3,2), (3,1) has max of 3.

Expectation

The **expectation** or **mean** of a random variable X is

$$E[X] = \sum_{x} x \cdot \Pr[X = x]$$

Example: Suppose X is the outcome of a dice.

$$E[X]$$
$$= 1 * \Pr[X = 1] + 2 * \Pr[X = 2] + 3 * \Pr[X = 3] + 4$$
$$* \Pr[X = 4] + 5 * \Pr[X = 5] + 6 * \Pr[X = 6] = 3.5$$

Linearity of Expectations

$$E[X+Y] = E[X] + E[Y]$$

$$E[aX] = aE[X]$$

- Example: Let X_1 and X_2 be the outcomes of two dice.
- $E[X_1+X_2]=E[X_1]+E[X_2]=3.5 + 3.5 = 7.$

Expectation of Product

If X and Y are **independent**:

$$E[XY] = E[X]E[Y]$$

Bernoulli Trial

An instance of a ***Bernoulli trial*** has probability p of success and probability $q = 1 - p$ of failure.

Geometric Distribution

Suppose we have a sequence of independent Bernoulli trials, each with prob *p* of success. Let *X* be the number of trials needed to obtain success for the first time.

Then, *X* follows the ***geometric distribution***

$$\Pr[X = k] = q^{k-1}p$$

$$E[X] = \frac{1}{p}$$

Question 5

Show that the first success occurs after $1/p$ independent Bernoulli trials with probability at most $1/e$.

$$\Pr\left[X > \frac{1}{p}\right] \leq \frac{1}{e}$$

Question 5: Solution

$$\Pr\left[X > \frac{1}{p}\right] \leq q^{1/p} = (1-p)^{\frac{1}{p}} \leq e^{-1}$$

Binomial Distribution

Let *X* be the number of successes in *n* Bernoulli trials.

Then *X* follows the ***binomial distribution***

$$\Pr\{X = k\} = \binom{n}{k} p^k q^{n-k} \qquad E[X] = np$$

Indicator Variable Method

Indicator Random Variable

Indicator random variable for an event A:

$$I[A] = \begin{cases} 1, & A \text{ occurs} \\ 0, & \text{otherwise} \end{cases}$$

$$E[I[A]] = \Pr[A]$$

Example

Find the expected number of heads in n coin tosses.

Let X_i be the indicator random variable indicating that the i-th flip comes up head.

Let X be the total number of heads in n coin tosses.

$$X = \sum_{i=1}^{n} X_i$$

Sum of indicators

$$E[X] = E\left[\sum_{i=1}^{n} X_i\right]$$

Expected value

$$= \sum_{i=1}^{n} E[X_i]$$

Linearity of expectation

$$= \sum_{i=1}^{n} 1/2 = n/2.$$

$E[X_i] = \Pr\{X_i = 1\}$

Question 6

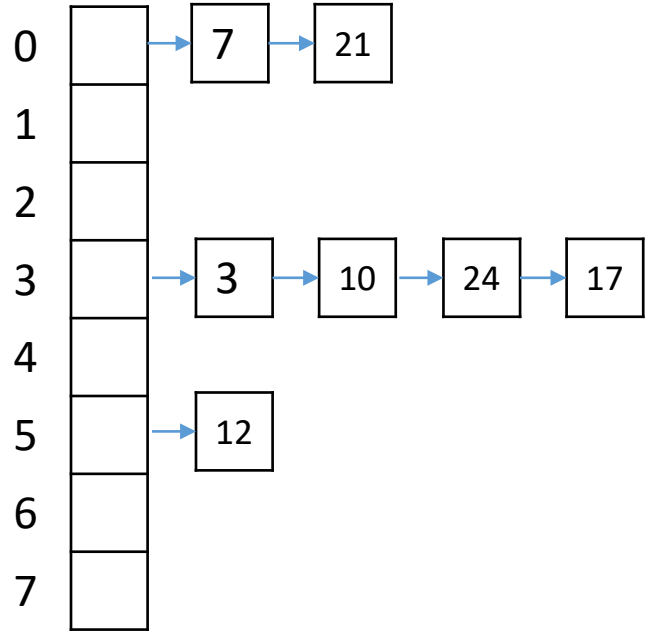
Given a hash table T with m slots that stores n elements, the **load factor** $\alpha = n/m$ is the average number of elements in a chain.

The hash function h is a randomized procedure satisfying the **universal hashing** assumption: for any pair of distinct keys u, v,

$$\Pr[h(u) = h(v)] \leq \frac{1}{m}$$

For a key x contained in T, the expected length of the chain at $h(x)$ is at most

- $\alpha/2$?
- α?
- $1 + \alpha$?
- $2 + \alpha$?



Question 6: Hint

- Let $C_y = I[h(y) = h(x)]$. What is $E[C_y]$ if $y \neq x$?

- Write the length of the chain in terms of the C_y's and use linearity of expectation.

Question 6: Solution

- Note that $E[C_y] = \Pr[h(y) = h(x)] \leq 1/m$.
- The length of chain at $h(x)$ is:

$$1 + \sum_{y \neq x, y \in T} C_y$$

- So, by linearity of expectation,

$$\mathrm{E}\left[1 + \sum_{y \neq x, y \in T} C_y\right] = 1 + \sum_{y \neq x, y \in T} E[C_y] = 1 + \frac{n-1}{m} \leq 1 + \alpha.$$

Coupon Collecting

There are *n* types of coupon that a collector would like to collect. The coupons are put into a box and randomly drawn with replacement. What is the expected number of times that the collector needs to draw from the box before he has at least one of each type of coupon?

Let T_i be the number of draws
used to collect the i-th coupon
and T be the total number of draws.

$$T = \sum_{i=1}^{n} T_i$$ Total number of draws

$$E[T] = E\left[\sum_{i=1}^{n} T_i\right]$$ Expected value

$$= \sum_{i=1}^{n} E[T_i]$$ Linearity of expectation

T_i is the time to collect the i-th coupon after i-1 coupon has been collected.

The prob of collecting a new coupon after i-1 coupon has been collected $p_i = (n-(i-1))/n$.

T_i has a geometric distribution with expectation $1/p_i = n/(n-(i-1))$.

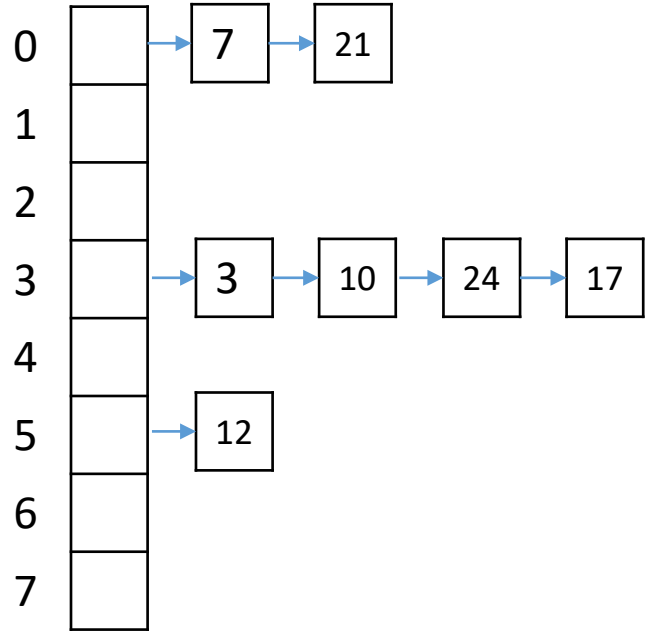
$$
\begin{aligned}
E[T] &= \sum_{i=1}^{n} E[T_i] \\
&= \sum_{i=1}^{n} \frac{n}{n-(i-1)} \\
&= \frac{n}{n} + \frac{n}{n-1} + \cdots + \frac{n}{1} \\
&= n \cdot \left(\frac{1}{1} + \frac{1}{2} + \cdots + \frac{1}{n} \right) \\
&= n \cdot H_n \\
&= O(n \lg n)
\end{aligned}
$$

where H_n is the n-th harmonic number and is of size $O(\lg n)$

Question 7

- You store elements by hashing with chaining. Your hash table H[1..n] of size n.
- What is the expected number of insertions to make the hash table full? (H[1..n] is full if every hash entry has at least 1 element.)

- (1) $\Theta(n)$
- (2) $\Theta(n \lg n)$
- (3) $\Theta(n^2)$
- (4) $\Theta(n^2 \lg n)$



Answer

- (2) $\Theta(n \lg n)$

- We model each H[i] as one type of coupon. There are n types of coupons.
- Insert one element in the hash table is equivalent as collecting one coupon.

- It is expected the hash table is full after $\Theta(n \lg n)$ insertions.

Question 8

Who is the Master of Algorithms pictured below?

- Ronald Rivest
- Alan Turing
- Donald Knuth
- Tony Hoare

Sir Anthony Hoare

Turing Award winner

- Invented quicksort
- Also invented quickselect (later lecture).
- Known for Hoare logic (for correctness), communicating sequential processes, monitor, axiomatic specification of computer language, ...

Analysis of Quicksort

Agenda

- First, we discuss average-case and worst-case running time of Quicksort

- Then, we discuss the analysis of randomized quicksort.

Quicksort

- Proposed by C.A.R. Hoare in 1962.
- Divide-and-conquer algorithm.
- Sorts “in place” (like insertion sort, but not like merge sort).
- Very practical (with tuning).

Divide and conquer

Quicksort an n-element array:

1. ***Divide:*** Partition the array into two subarrays around a ***pivot*** x such that elements in lower subarray $\leq x \leq$ elements in upper subarray.

2. ***Conquer:*** Recursively sort the two subarrays.
3. ***Combine:*** Trivial.

Key: *Linear-time partitioning subroutine.*

Pseudocode for quicksort

```
QUICKSORT(A, p, r)
  if p < r
    then q ← PARTITION(A, p, r)
         QUICKSORT(A, p, q–1)
         QUICKSORT(A, q+1, r)
```

Initial call: QUICKSORT (*A*, 1, *n*)

Analysis of quicksort

- Let $A(n)$ = average-case running time on an array of n elements.
- Let $T(n)$ = worst-case running time on an array of n elements.
- Our analysis assumes all input elements are distinct.
  - If duplicates exist, the running time of quicksort is better if we use a better partitioning algorithm.

Average case analysis

- Denote $A_i(n)$ be the running time of quick sort when the pivot is rank-i.
- Hence, $A_i(n) = A(i-1) + A(n-i) + n + 1$.

- The chance that the pivot is rank-i = $\frac{1}{n}$.
- Hence, we have $A(n) = \sum_{i=1}^{n} \frac{1}{n} A_i(n)$
- → $A(n) = \sum_{i=1}^{n} \frac{1}{n} (A(i-1) + A(n-i) + n + 1)$
- → $nA(n) = \sum_{i=1}^{n} \big(A(i-1) + A(n-i)\big) + n(n+1)$
- → $nA(n) = 2\sum_{i=1}^{n} A(i-1) + n(n+1)$
- Hence, $nA(n) - (n-1)A(n-1) = 2A(n-1) + 2n$

- So, $nA(n) = (n+1)A(n-1) + 2n$.

Average case analysis

- $nA(n) = (n+1)A(n-1) + 2n$
- $\rightarrow \frac{A(n)}{n+1} = \frac{A(n-1)}{n} + \frac{2}{n+1}$
- By telescoping, we have:

$$\frac{A(n)}{n+1} = \frac{A(n-1)}{n} + \frac{2}{n+1}$$
$$\frac{A(n-1)}{n} = \frac{A(n-2)}{n-1} + \frac{2}{n}$$
$$\frac{A(n-2)}{n-1} = \frac{A(n-3)}{n-2} + \frac{2}{n-1}$$
$$\cdots$$
$$\frac{A(1)}{2} = \frac{A(0)}{1} + \frac{2}{2}$$

- $\frac{A(n)}{n+1} = \frac{A(0)}{1} + 2\left(\frac{1}{1} + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n+1}\right) = \Theta(\lg n)$
- $\rightarrow A(n) = \Theta(n \lg n)$

The average-case running time of Quicksort is $\Theta(n \lg n)$.

Worst-case of quicksort

- Input sorted or reverse sorted.
- Partition around min or max element.
- One side of partition always has no elements.

$$
\begin{aligned}
T(n) &= T(0) + T(n-1) + \Theta(n) \\
&= \Theta(1) + T(n-1) + \Theta(n) \\
&= T(n-1) + \Theta(n) \\
&= \Theta(n^2) \qquad \textit{(arithmetic series)}
\end{aligned}
$$

Randomized Quicksort

Why standard Quicksort is not good?

- For the standard quicksort, if the input distribution is bad, it will runs in n^2 time!
  - For example, if the input is in reverse order, it takes $O(n^2)$ time to sort!

- Is there any way to make the expected running good independent of the input distribution?
  - ANS: Yes!

Randomized quicksort

IDEA: Partition around a ***random*** element.

- Running time is independent of the input order.
- No assumptions need to be made about the input distribution.
- No specific input elicits the worst-case behavior.
- The worst case is determined only by the output of a random-number generator.

Randomized quicksort analysis

Let $T(n) =$ the random variable for the running time of randomized quicksort on an input of size n.

We assume random numbers are independent.

For $k = 0, 1, \ldots, n-1$, define the ***indicator random variable***

$$X_k = \begin{cases} 1 & \text{if PARTITION generates a } k : n-k-1 \text{ split,} \\ 0 & \text{otherwise.} \end{cases}$$

$E[X_k] = \Pr\{X_k = 1\} = 1/n$, since all splits are equally likely, assuming elements are distinct.

Analysis (continued)

$$T(n) = \begin{cases} T(0) + T(n-1) + \Theta(n) & \text{if } X_0=1, \text{ i.e., } 0 : n-1 \text{ split,} \\ T(1) + T(n-2) + \Theta(n) & \text{if } X_1=1, \text{ i.e., } 1 : n-2 \text{ split,} \\ \quad\quad\quad \cdots & \\ T(n-1) + T(0) + \Theta(n) & \text{if } X_{n-1}=1, \text{ i.e., } n-1 : 0 \text{ split,} \end{cases}$$

$$= \sum_{k=0}^{n-1} X_k \left(T(k) + T(n-k-1) + \Theta(n) \right)$$

Calculating expectation

$$E[T(n)] = E\left[\sum_{k=0}^{n-1} X_k \left(T(k) + T(n-k-1) + \Theta(n)\right)\right]$$

Take expectations of both sides.

Calculating expectation

$$E[T(n)] = E\left[\sum_{k=0}^{n-1} X_k \left(T(k) + T(n-k-1) + \Theta(n)\right)\right]$$

$$= \sum_{k=0}^{n-1} E\left[X_k \left(T(k) + T(n-k-1) + \Theta(n)\right)\right]$$

Linearity of expectation.

Calculating expectation

$$E[T(n)] = E\left[\sum_{k=0}^{n-1} X_k \left(T(k) + T(n-k-1) + \Theta(n)\right)\right]$$

$$= \sum_{k=0}^{n-1} E\left[X_k \left(T(k) + T(n-k-1) + \Theta(n)\right)\right]$$

$$= \sum_{k=0}^{n-1} E\left[X_k\right] \cdot E\left[T(k) + T(n-k-1) + \Theta(n)\right]$$

Independence of X_k from other random choices.

Calculating expectation

$$
\begin{aligned}
E[T(n)] &= E\left[\sum_{k=0}^{n-1} X_k \left(T(k) + T(n-k-1) + \Theta(n)\right)\right] \\
&= \sum_{k=0}^{n-1} E\left[X_k \left(T(k) + T(n-k-1) + \Theta(n)\right)\right] \\
&= \sum_{k=0}^{n-1} E\left[X_k\right] \cdot E\left[T(k) + T(n-k-1) + \Theta(n)\right] \\
&= \frac{1}{n}\sum_{k=0}^{n-1} E\left[T(k)\right] + \frac{1}{n}\sum_{k=0}^{n-1} E\left[T(n-k-1)\right] + \frac{1}{n}\sum_{k=0}^{n-1} \Theta(n)
\end{aligned}
$$

Linearity of expectation; $E[X_k] = 1/n$.

Calculating expectation

$$E[T(n)] = E\left[\sum_{k=0}^{n-1} X_k \left(T(k) + T(n-k-1) + \Theta(n)\right)\right]$$

$$= \sum_{k=0}^{n-1} E\left[X_k \left(T(k) + T(n-k-1) + \Theta(n)\right)\right]$$

$$= \sum_{k=0}^{n-1} E\left[X_k\right] \cdot E\left[T(k) + T(n-k-1) + \Theta(n)\right]$$

$$= \frac{1}{n}\sum_{k=0}^{n-1} E\left[T(k)\right] + \frac{1}{n}\sum_{k=0}^{n-1} E\left[T(n-k-1)\right] + \frac{1}{n}\sum_{k=0}^{n-1} \Theta(n)$$

$$= \frac{2}{n}\sum_{k=1}^{n-1} E\left[T(k)\right] + \Theta(n)$$

Summations have identical terms.

Hairy recurrence

$$E[T(n)] = \frac{2}{n}\sum_{k=2}^{n-1} E[T(k)] + \Theta(n)$$

(The $k = 0, 1$ terms can be absorbed in the $\Theta(n)$.)

Prove: $E[T(n)] \le a\,n \lg n$ for constant $a > 0$.

- Choose a large enough so that $a\,n \lg n$ dominates $E[T(n)]$ for sufficiently small $n \ge 2$.

Use fact: $\sum_{k=2}^{n-1} k \lg k \le \frac{1}{2} n^2 \lg n - \frac{1}{8} n^2$ (exercise).

Substitution method

$$E[T(n)] \le \frac{2}{n} \sum_{k=2}^{n-1} ak \lg k + \Theta(n)$$

Substitute inductive hypothesis.

Substitution method

$$E[T(n)] \le \frac{2}{n}\sum_{k=2}^{n-1} ak \lg k + \Theta(n)$$

$$\le \frac{2a}{n}\left(\frac{1}{2}n^2 \lg n - \frac{1}{8}n^2\right) + \Theta(n)$$

Use fact.

Substitution method

$$E[T(n)] \le \frac{2}{n}\sum_{k=2}^{n-1} ak \lg k + \Theta(n)$$

$$\le \frac{2a}{n}\left(\frac{1}{2}n^2 \lg n - \frac{1}{8}n^2\right) + \Theta(n)$$

$$= an \lg n - \left(\frac{an}{4} - \Theta(n)\right)$$

Express as ***desired – residual***.

Substitution method

$$
\begin{aligned}
E[T(n)] &\le \frac{2}{n}\sum_{k=2}^{n-1} ak \lg k + \Theta(n) \\
&= \frac{2a}{n}\left(\frac{1}{2}n^2 \lg n - \frac{1}{8}n^2\right) + \Theta(n) \\
&= an \lg n - \left(\frac{an}{4} - \Theta(n)\right) \\
&\le an \lg n \text{ ,}
\end{aligned}
$$

if a is chosen large enough so that $an/4$ dominates the $\Theta(n)$.

Randomized Quicksort in practice

- Randomized Quicksort is a great general-purpose sorting algorithm. It always returns the correct solution, so is a ***Las Vegas algorithm***.
- Randomized Quicksort is typically over twice as fast as merge sort.
- Randomized Quicksort can benefit substantially from ***code tuning***.
- Randomized Quicksort behaves well even with caching and virtual memory.

Summary

- Randomize your decision can make the running time faster.
- Linearity of expectation and Indicator random variable are useful technique to analyze the running time of randomize algorithm.
- Randomized quicksort runs in expected $\Theta(n \lg n)$ time.

Acknowledgement

- The slides are modified from
  - the slides from Prof. Erik D. Demaine and Prof. Charles E. Leiserson
  - the slides from Prof. Lee Wee Sun
  - The slides from Prof. Arnab Bhattacharyya