Algorithm Analysis and Data Structures

CSCI 7432 - Fall 2022

Randomized Algorithms

Dr. Yao XU

Assistant Professor

Department of Computer Science

Georgia Southern University

Email: yxu@georgiasouthern.edu

Table of Contents

Randomized Algorithms

Probabilistic Analysis

Probability Basics

- A ***(discrete) random variable*** X is a function that takes values (real numbers) in some range according to a ***probability distribution***.
  - **Example 1:** $\Pr\{X = 1\} = 0.5, \Pr\{X = 2\} = 0.2, \Pr\{X = 3\} = 0.3.$
  - Probability must be non-negative and sums to 1.
- The ***expectation*** of a random variable is a weighted average of the outcome according to the probability distribution:

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

Indicator Random Variables

- The ***indicator random variable*** takes values in $\{0, 1\}$ indicating whether some event happened or didn't happen.

$$X_A = \begin{cases} 1, & \text{if } A \text{ occurs,} \\ 0, & \text{if } A \text{ does not occur.} \end{cases}$$

- $E[X_A] = \Pr\{A\} = \Pr\{X_A = 1\}$
- **Example:** Define X_H as the number of heads when tossing a fair coin once.
  - $\Pr\{X_H = 1\} = \Pr\{X_H = 0\} = \frac{1}{2}$
  - The expected number of heads in one toss is

$$E[X_H] = \Pr\{X_H = 1\} = \frac{1}{2}$$

Linearity of Expectation

- Expectation has a beautiful property – Expectation is linear.
- For any two random variables X and Y, we have:

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

Example: Given a fair coin, what is the expected number of heads when the coin is tossed 1,000,000 times?

Solution: Let X_j be the number of heads in j-th toss. Then the total number of heads in 1,000,000 tosses is $X = \sum_{j=1}^{1,000,000} X_j$.

$$E[X] = E\left[\sum_{j=1}^{1,000,000} X_j\right] = \sum_{j=1}^{1,000,000} E[X_j] = \sum_{j=1}^{1,000,000} \Pr\{X_j = 1\} = 500,000$$

Motivation for Studying Expectation

In the previous coin tossing example,

- Tossing the coin many times means we expect to see about $E[X]$ "heads".
- It is very unlikely, for a fair coin, to see < 495,000 "heads".

For a ***randomized algorithm***, whose behavior is determined not only by its input but also by values produced by a ***random-number generator***.

- We analyze the ***expected*** running time.
- We can have multiple independent instances running in parallel and use whichever halts first.

A **random-number generator**: $\text{RANDOM}(a, b)$ returns an integer between a and b, inclusive, with each such integer being equally likely.

Randomly Permuting Arrays

- When an array is part of the input, a randomized algorithm may randomize the input by permuting the given input array.
- An $O(n)$-time method of permuting a given array:

```
RANDOMIZE-IN-PLACE(A)
n = A.length
for i = 1 to n
    swap A[i] with A[RANDOM(i, n)]
```

RANDOM(a, b) returns an integer between a and b, inclusive, with each such integer being equally likely.

- RANDOMIZE-IN-PLACE(A) computes a ***uniform random permutation***.*
- That is, all possible permutations of the array A are equally likely.
  - Can be proved using **LI**. *

** See Lemma 5.5 on p.127-128 of the textbook.*

Randomized Algorithms

Randomized Linear Search

Linear Search

```
LINEAR-SEARCH(A, x)
1  for i = 1 to A.length
2      if A[i] == x
3          return i
4  return NIL
```

Example:

1	2	3	4	...	*n*
28	53	17	36	...	9

- **Worst-case** running time: $\Theta(n)$
- **Best-case** running time: $\Theta(1)$
- **Average-case** running time?
  - Assume some probability distribution over the inputs.
  - **Example:** Assume all possible permutations of the array are equally likely. - ***Uniform random permutation***

Average Case Analysis

```
LINEAR-SEARCH(A, x)
1  for i = 1 to A.length
2      if A[i] == x
3          return i
4  return NIL
```

- Assume uniform random permutation.
- Assume there is exactly one index i such that $A[i] = x$. Then,

$$\Pr\{A[i] = x\} = \frac{1}{n}, \text{ where } n = A.length$$

- Let t_i be the number of ***Key Comparisons (KC)*** when $A[i] = x$.
- Then, the average number of KC made by LINEAR-SEARCH is

$$E[T(n)] = \sum_{i=1}^{n} t_i \cdot \Pr\{T(n) = t_i\}$$

$$= 1 \cdot \frac{1}{n} + 2 \cdot \frac{1}{n} + 3 \cdot \frac{1}{n} + \cdots + n \cdot \frac{1}{n} = \frac{n+1}{2}.$$

- **Average-case** running time: $\Theta(n)$

Randomized Linear Search (1/2)

```
LINEAR-SEARCH(A, x)
1  for i = 1 to A.length
2      if A[i] == x
3          return i
4  return NIL
```

```
RANDOMIZED-LINEAR-SEARCH(A, x)
1  RANDOMIZE-IN-PLACE(A)
2  for i = 1 to A.length
3      if A[i] == x
4          return i
5  return NIL
```

- We discuss the ***expected*** running time when the algorithm itself makes random choices.
- We will compute the expected number of KC made by RANDOMIZED-LINEAR-SEARCH.

Randomized Linear Search (2/2)

```
RANDOMIZED-LINEAR-SEARCH(A, x)
1  RANDOMIZE-IN-PLACE(A)
2  for i = 1 to A.length
3      if A[i] == x
4          return i
5  return NIL
```

- Assume there is exactly one index i such that $A[i] = x$.
- Then, $\Pr\{A[i] = x\} = \frac{1}{n}$.
- Let t_i be the number of KC made when $A[i] = x$.

- The expected number of KC made by RANDOMIZED-LINEAR-SEARCH is

$$E[T(n)] = \sum_{i=1}^{n} t_i \cdot \Pr\{T(n) = t_i\} = \frac{n+1}{2}.$$

- The **expected** running time of RANDOMIZED-LINEAR-SEARCH is the same as the **average-case** running time of LINEAR-SEARCH.

Review: Quicksort Algorithm

The Quicksort Algorithm

```
QUICKSORT(A, p, r)
if p < r
    q = PARTITION(A, p, r)
    QUICKSORT(A, p, q - 1)
    QUICKSORT(A, q + 1, r)

PARTITION(A, p, r)
x = A[r]  // pivot is the last element
i = p – 1
for j = p to r - 1
    if A[j] ≤ x
        i = i + 1
        swap A[i] and A[j]
swap A[i + 1] and A[r]
return i + 1
```

- **Divide-and-conquer**
- **Ideas:**
  - Pick one key (***pivot***), compare it to all others.
  - Rearrange A to be:

$\leq pivot$	$pivot$	$> pivot$

  - Recursively sort subarrays before and after the pivot.
- The PARTITION procedure returns q such that
  - $A[q] = pivot$
  - All elements $\leq pivot$ are in $A[p..(q-1)]$
  - All elements $> pivot$ are in $A[(q+1)..r]$

Quicksort Running Time

```
QUICKSORT(A, p, r)
if p < r
    q = PARTITION(A, p, r)
    QUICKSORT(A, p, q - 1)
    QUICKSORT(A, q + 1, r)

PARTITION(A, p, r)
x = A[r]
i = p – 1
for j = p to r - 1
    if A[j] ≤ x
        i = i + 1
        swap A[i] and A[j]
swap A[i + 1] and A[r]
return i + 1
```

- **Running time** of PARTITION:

$$\Theta(n), \text{ where } n = r - p + 1.$$

- **Running time** of QUICKSORT:

$$T(n) = \begin{cases} \Theta(1), & n \leq 1 \\ T(n_1) + T(n - 1 - n_1) + \Theta(n), & n \geq 2 \end{cases}$$

where $0 \leq n_1 \leq n - 1$.

- This raises the question:

How can we estimate n_1?

There is no single answer.

Quicksort Worst-Case Running Time

- Running time recurrence:

$$T(n) = \begin{cases} \Theta(1), & n \leq 1 \\ T(n_1) + T(n-1-n_1) + \Theta(n), & n \geq 2 \end{cases}$$

  where $0 \leq n_1 \leq n-1$.

- **WC running time**:

$$T(n) = T(0) + T(n-1) + \Theta(n)$$

- Solving the recurrence (by substitution method):

$$T(n) \in \Theta(n^2).$$

Quicksort Best-Case Running Time

- Running time recurrence:

$$T(n) = \begin{cases} \Theta(1), & n \leq 1 \\ T(n_1) + T(n-1-n_1) + \Theta(n), & n \geq 2 \end{cases}$$

  where $0 \leq n_1 \leq n-1$.

- **BC running time**: each partition is a *bipartition*

$$T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil - 1) + \Theta(n)$$
$$\approx 2T(n/2) + \Theta(n)$$

- Solving the recurrence (by case 2 of Master Theorem):

$$T(n) \in \Theta(n \log n)$$

Quicksort Almost-BC Running Time

- Assume that at each round we get an *approximated bipartition*.
- If each split is $\frac{3}{4}n$ and $\frac{1}{4}n$, the recurrence will be

$$T(n) \approx T\left(\frac{3n}{4}\right) + T\left(\frac{n}{4}\right) + \Theta(n),$$

- A more extreme case with split $\frac{9}{10}n$ and $\frac{1}{10}n$ resulting in recurrence

$$T(n) \approx T\left(\frac{9n}{10}\right) + T\left(\frac{n}{10}\right) + \Theta(n).$$

- In both cases, $T(n) \in \Theta(n \log n)$.
- In fact, for any split of constant proportionality, the running time remains to be $\Theta(n \log n)$.*

---

* *See p.175-176 of the textbook for detailed explanations.*

Quicksort Average-Case Running Time

- Running time recurrence:

$$T(n) = \begin{cases} \Theta(1), & n \leq 1 \\ T(n_1) + T(n-1-n_1) + \Theta(n), & n \geq 2 \end{cases}$$

  where $0 \leq n_1 \leq n-1$.

- **Q:** "What is the probability for the left subarray to have size n_1?"
- **Average Case (AC)**: always ask "average over what input distribution?"
- **Ans:** We make a huge assumption about the input data.
- **Example:** Assume each possible input is equiprobable (uniform distribution).

  That is, n_1 can be $0, 1, 2, \cdots, n-2, n-1$, with the same probability $\frac{1}{n}$.

Quicksort Space Complexity

```
QUICKSORT(A, p, r)
if p < r
    q = PARTITION(A, p, r)
    QUICKSORT(A, p, q - 1)
    QUICKSORT(A, q + 1, r)

PARTITION(A, p, r)
x = A[r]
i = p – 1
for j = p to r - 1
    if A[j] ≤ x
        i = i + 1
        swap A[i] and A[j]
swap A[i + 1] and A[r]
return i + 1
```

- Extra space required at each recursive call is only constant.
- **Space complexity** is $\Theta(1)$.
- A sorting algorithm is said to be ***in place*** if
  - it rearranges all the elements within the array,
  - with at most a constant number of extra memory units required.
- QUICKSORT is an ***in-place*** sorting algorithm.

Randomized Quicksort

Randomized Algorithm v.s. AC Analysis

- AC analysis means we make an assumption on the input
  - No guarantee that the assumption holds.
  - Input is chosen once: on avg we might have a good running time, but once input is given our running time is determined.
- A randomized algorithm works for any input (WC)
  - Randomness in the coins we toss (not in the input) - so we control the distribution of the coin toss.
  - We can always start the algorithm anew if it takes too long; or run it multiple times in parallel and use whichever halts first.

Randomized Quicksort

- To improve the QUICKSORT algorithm, we use a random pivot.
- We invoke RANDOMIZED-PARTITION rather than PARTITION.

```
RANDOMIZED-QUICKSORT(A, p, r)
if p < r
    q = RANDOMIZED-PARTITION(A, p, r)
    RANDOMIZED-QUICKSORT(A, p, q - 1)
    RANDOMIZED-QUICKSORT(A, q + 1, r)
```

```
RANDOMIZED-PARTITION(A, p, r)
// Randomly choose an integer between p and r
i = RANDOM(p, r)
swap A[r] with A[i]
return PARTITION(A, p, r)
```

- How to analyze the **expected running time** of RANDOMIZED-QUICKSORT?
  - **Technique #1**: Find the recurrence relation for the expectation.
  - **Technique #2** ***(Optional)***: Find sum of expected indicator random variables, $X_{i,j}$, indicating whether elements a_i and a_j are compared.

    (See p.182-184 of the textbook for details.)

Randomized Partition

```
RANDOMIZED-PARTITION(A, p, r)
1  i = RANDOM(p, r)
2  swap A[r] with A[i]
3  return PARTITION(A, p, r)
```

```
PARTITION(A, p, r)
1  x = A[r]
2  i = p – 1
3  for j = p to r - 1
4       if A[j] ≤ x
5            i = i + 1
6            swap A[i] and A[j]
7  swap A[i + 1] and A[r]
8  return i + 1
```

- **Running time** of PARTITION: $\Theta(n)$
- **Expected running time** of RANDOMIZED-PARTITION:
  - Each element $A[i]$ gets swapped with $A[r]$ with the same probability, $\frac{1}{n}$.
  - $\sum_{i=1}^{n} \Theta(n) \cdot \frac{1}{n} = \Theta(n)$
  - Same as the running time of PARTITION.

Expected Running Time of Randomized Quicksort (1/3)

Technique #1: Find the recurrence relation for the expectation.

- *Recall:* If PARTITION put n_1 elements in one side of the pivot and $n - n_1 - 1$ on the other side, then

$$T(n) = T(n_1) + T(n - 1 - n_1) + \Theta(n).$$

- Since the pivot is chosen uniformly at random, for $k = 0, 1, 2, \cdots, n - 1$,

$$\Pr\{n_1 = k\} = \Pr\{pivot = (k+1)\text{–st smallest element}\} = \frac{1}{n}.$$

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

$$= \Theta(n) + \frac{2}{n}\sum_{k=0}^{n-1} E[T(k)].$$

- Need to solve this recurrence relation.

Expected Running Time of Randomized Quicksort (2/3)

Solve the recurrence

$$E[T(n)] = \Theta(n) + \frac{2}{n}\sum_{k=0}^{n-1} E[T(k)]$$

```
PARTITION(A, p, r)
1  x = A[r]
2  i = p – 1
3  for j = p to r - 1
4       if A[j] ≤ x
5            i = i + 1
6            swap A[i] and A[j]
7  swap A[i + 1] and A[r]
8  return i + 1
```

- Running time of PARTITION is dominated by the total number of key comparisons (KC) (line 4)
- #KC $= n - 1 \Rightarrow$ Replace $\Theta(n)$ by $n - 1$.
- Solve the recurrence

$$E[T(n)] = (n-1) + \frac{2}{n}\sum_{k=0}^{n-1} E[T(k)]$$

Expected Running Time of Randomized Quicksort (3/3)

- Solve the recurrence

$$E[T(n)] = (n-1) + \frac{2}{n}\sum_{k=0}^{n-1} E[T(k)]$$

- **Solution:** $E[T(n)] = 2(n+1)\boldsymbol{H}(n+1) - (4n+2) \in \Theta(n \log n)$

  (The ***Harmonic number*** $\boldsymbol{H}(n) = \sum_{i=1}^{n} \frac{1}{i} = \ln n + \gamma$, where $\gamma \approx 0.577\cdots$)

- *The next two slides show how this bound is obtained. (Optional)*
- The **expected running time** of RANDOMIZED-QUICKSORT is in $\Theta(n \log n)$.

Solving $E[T(n)] = (n-1) + \frac{2}{n}\sum_{i=0}^{n-1} E[T(i)]$ *(Optional)* (1/2)

- Multiply both sides by n: $n \cdot E[T(n)] = n(n-1) + 2\sum_{i=0}^{n-1} E[T(i)]$
- Then, we have: $(n-1)E[T(n-1)] = (n-1)(n-2) + 2\sum_{i=0}^{n-2} E[T(i)]$
- Subtract the above two terms:

$$n \cdot E[T(n)] - (n-1)E[T(n-1)] = 2E[T(n-1)] + 2(n-1)$$

$$\Longrightarrow n \cdot E[T(n)] = (n+1)E[T(n-1)] + 2(n-1)$$

- With some arithmetics, we have:

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

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

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

Solving $E[T(n)] = (n-1) + \frac{2}{n}\sum_{i=0}^{n-1} E[T(i)]$ *(Optional)* (2/2)

- which gives you (by substitution method)

$$\frac{E[T(n)]}{n+1} = \frac{E[T(n-1)]}{n} + \frac{2}{n+1} - 2\left(\frac{1}{n} - \frac{1}{n+1}\right) = \cdots$$

$$= \sum_{i=1}^{n} \frac{2}{i+1} + \left(\frac{2}{n+1} - 2\right) = \sum_{i=1}^{n} \frac{2}{i+1} - \frac{2n}{n+1}.$$

- *Recall* the Harmonic number $H(n) = \sum_{i=1}^{n} \frac{1}{i} = \ln n + \gamma$, where $\gamma \approx 0.577\cdots$
- We have

$$E[T(n)] = 2(n+1)H(n+1) - (4n+2)$$

$$\approx 2(n+1)(\ln(n+1) + \gamma) - (4n+2)$$

$$\in \Theta(n \log n).$$

Thank you!
Questions?