CSC263H
Data Structures and Analysis

Prof. Bahar Aameri & Prof. Marsha Chechik

Winter 2024 – Week 1

Abstract Data Types and Data Structures

- **Abstract Data Type (ADT)**: a set of objects together with a set of operations on these objects.
  Example: Stack ADT
  PUSH(S,v): add element v to the collection S.
  POP(S): removes the most recently added element that was not yet removed.
  ISEMPTY(S): returns whether the collection S is empty.

- **Data Structure**: an implementation of an ADT.
  Example: Data structures for Stack:

  1. Linked list (keep pointer to head).
     ISEMPTY: test head == None
     PUSH: insert at front of the list
     POP: remove front of the list (if not empty)

  2. Array with counter (size of stack).
     ISEMPTY: test counter == 0
     PUSH: insert at front of array and increase counter.
     POP: remove front of array (if not empty) and decrease counter.

In CSC263 we will:

1. **Motivate** a new ADT.
2. **Introduce** a data structure, discussing both its mechanisms for how it stores data and how it implements operations on this data.
3. **Analyze** the running time performance of these operations.
4. **Justify** why the operations are correct with respect to the description of the ADT.

Review: Algorithm Analysis

- **Complexity**: Amount of resources required for running an algorithm, measured as a function of input size.
- Resource: running-time or memory space (usually).
- Why analyze complexity?
  To **choose** between different implementations.

Review: Complexity of Algorithms

- **Time Complexity**: Number of steps executed by an algorithm.

- **Space Complexity**: Number of units of space required by an algorithm.
  Example:
  - Number of elements in a list
  - Number of nodes in a tree

Review: Running-Time Complexity of Algorithms

- **Running-Time Analysis**: the relationship between an algorithm's input size and the number of basic operations the algorithm performs.

- **Basic Operation**: any operation whose run-time **does not depend** on the input size.
  Example: arithmetic operations, assignments, array accesses, comparisons, return statements, etc.

- We do not try to precisely quantify the exact number of basic operations.

Measuring running time by counting steps

- Represent as a function $T(n)$ of input size n.

- We don't care about exact step counts, an estimate for $T(n)$ is sufficient:
  - every **chunk** of instructions is represented by a constant.
  - **chunk:** sequence of instructions that always gets executed together.

- **Note:** Runtime can be measured by counting the number of times **all lines** are executed, or the number of times **some important lines** are executed.
  It is up to the problem, or what the question asks, so always read the question carefully.

- Most of the time, NO simple algebraic expression for $T(n)$.
  Instead, we prove bounds on $T(n)$ using *asymptotic notation*.

- **Upper bound:** $T(n) \in \mathcal{O}(f(n))$.
- **Lower bound:** $T(n) \in \Omega(f(n))$.
- **Tight bound:** $T(n) \in \Theta(f(n))$: $T(n) \in \mathcal{O}(f(n))$ and $T(n) \in \Omega(f(n))$.

Some Rules for Big-Oh Notation - Review

- If $T(n)$ is a **polynomial** of degree k, then $T(n) \in \mathcal{O}(n^k)$.

- If $T(n) = g(n) + f(n)$, and $f(n)$ **asymptotically dominates** $g(n)$, then $T(n) \in \mathcal{O}(f(n))$.

$$f(n) = n^n$$
$$f(n) = 2^n$$
$$f(n) = n^3$$
$$f(n) = n^2$$
$$f(n) = n \log n$$
$$f(n) = n$$
$$f(n) = \sqrt{n}$$
$$f(n) = \log n$$
$$f(n) = 1$$

grow fast

grow slowly

Different **Cases** of Running Time

Let $t(x)$ represent number of steps executed by an algorithm A on input x.

- **Worst-Case** Running Time of A: The maximum running time of A for all inputs of size n.
$$T(n) = max\{t(x) : x \text{ is an input of size } n\}$$

- **Best-Case** Running Time of A: The minimum running time of A for all inputs of size n.
$$T(n) = min\{t(x) : x \text{ is an input of size } n\}$$

- **Average-Case** Running Time of A: The expected running time of A for all inputs of size n.
$$T(n) = \mathbb{E}[t_n]$$

Worst-Case Running Time Analysis

1. Identify the input size.
   Example:
   - For numbers: number of bits.
   - For lists: number of elements.
   - For graphs: number of vertices and/or edges.

2. Identify the case in which the performance of the algorithm is worst; i.e., takes longer to terminate (You need to understand how the algorithm works).

3. Give an approximation of number of basic operations that execute in that case. Denote it by $T(n)$.

4. Give an upper-bound/lower-bound/tight-bound for $T(n)$.

Worst-Case Running Time Analysis: Example

L is a linked-list

```
def LinkedSearch(L):
    z = L.head
    while z != None and z.key != 42:   → n
        z = z.next
    return z
```

1. Input size: $n = \mathrm{Len}(L)$
2. What is the worst-case: 42 is not in L or is at the last node
3. Worst-case run-time: $n+1$, $cn+b$ $\in \left.\begin{array}{l} O(n^2) \\ O(n) \\ \Omega(n) \\ \Omega(1) \end{array}\right\} \Rightarrow \Theta(n)$
4. Upper-bound/lower-bound/tight-bound for $T(n)$: $\Theta(n)$

Worst-Case Running Time Analysis: Example

L is a list.

```
def EvilEvens(L):
    if every number in L is even:
        repeat L.length times:
            calculate and print the sum of L
        return 1
    else:
        return 0
```

n

n

n

$n \times n$

1. Input size: $n = len(L)$
2. What is the worst-case: All elements in L are even
3. Worst-case run-time: $T(n) = n^2 + n$
4. Upper-bound/lower-bound/tight-bound for $T(n)$: $\Theta(n^2)$

IMPORTANT: Bounds vs Cases

- **Misconceptions**:
  $\mathcal{O}$ is for describing worst-case running time
  Ω is for describing best-case running time

- $\mathcal{O}$ and Ω specify bounds over a ***mathematical function***.
- Worst-case and best-case correspond to ***algorithms***.
- $\mathcal{O}$ and Ω can ***both*** be used to upper-bound and lower-bound the worst-case running time.
- $\mathcal{O}$ and Ω can ***both*** be used to upper-bound and lower-bound the best-case running time.

Worst-Case Running Time Analysis

Recall that the **worst-case** running time of an algorithm $A(x)$ is defined as the maximum running time of A for all inputs of size n. That is:

$$T(n) = max\{t(x) : x \text{ is an input of size } n\}$$

where $t(x)$ represent number of steps executed by A on input x.

How to argue algorithm $A(x)$ worst-case runtime is in $\mathcal{O}(n^2)$?

We need to argue that ____________ input x of size n, the number of steps executed by A on input x, i.e., $t(x)$ is ____________ than cn^2, where $c > 0$ is a constant.

- for every
- there exists an

- no larger
- no smaller

Worst-Case Running Time Analysis

Analogy: Proving an "upper-bound" on the height of people in a room.

To prove **the tallest person in the room is at most 2 metres**,
we need to show **every/some** person in the room is
no taller/no smaller than 2 metres.

Recall that the **worst-case** running time of an algorithm $A(x)$ is defined as the maximum running time of A for all inputs of size n. That is:

$$T(n) = max\{t(x) : x \text{ is an input of size } n\}$$

where $t(x)$ represent number of steps executed by A on input x.

How to argue algorithm $A(x)$ worst-case runtime is in $\Omega(n^2)$?

We need to argue that ____________ input x of size n, the number of steps executed by A on input x, i.e., $t(x)$ is ____________ than cn^2, where $c > 0$ is a constant.

- for every
- there exists an

- no larger
- no smaller

Analogy: Proving an "lower-bound" on the height of people in a room.

To prove **the tallest person in the room is at least 2 metres**,
we need to show **every/some** person in the room is
no taller/no smaller than 2 metres.

- In reality, the running time is *NOT* always the best case or the worst case.
  It is ***distributed*** between the best and the worst.
  Example: For the *LinkedSearch(L)* algorithm the runtime is distributed between:
  1 to n+1 (inclusive)

- Computing **average-case** running time for an algorithm A:

  1. Define S_n: space of ***all inputs*** of size n.
  2. Assume a ***probability distribution*** over S_n: specifying likelihood of each input.
  3. Define the ***random variable*** t_n over S_n, representing the running time of A:
     $t_n(x)$: number of steps executed by A on an input x in S_n.
     Example: For the *LinkedSearch(L)*, t_n takes values between 1 to n+1
  4. ***Compute the expected value*** of $t_n(x)$:

$$T(n) = \mathbb{E}[t_n] = \sum_i i \times Pr[t_n = i]$$

     $Pr[i = t_n]$: Probability of t_n obtaining the value i (according to the probability distribution).

- To know $Pr(i = t_n)$, we need to know the *probability distribution* on the inputs. E.g., by specifying how inputs are generated.

- **Example Distribution:**
  For each key in the linked list, we pick an integer between 1 and 100 (inclusive), *independently, uniformly at random*.

Average-Case Running Time Analysis – Example

Assumption: For each key in the linked list, we pick an integer between 1 and 100 (inclusive), *independently, uniformly at random*.

$S_n = \{L : L$ is a list of size n which includes $\}$

L is a linked-list

```
def LinkedSearch(L):
    z = L.head
    while z != None and z.key != 42:
        z = z.next
    return z
```

$t_n: 1, 2, 3, \ldots, n, n+1$

$P(t_n = 1) = \frac{1}{100}$ (head is 42)

$P(t_n = 2) = \frac{99}{100} \times \frac{1}{100}$ (head is not 42 but second node is)

$P(t_n = 3) = \left(\frac{99}{100}\right)^2 \times \frac{1}{100}$

$P(t_n = i) = \left(\frac{99}{100}\right)^{i-1} \times \frac{1}{100}$ $\quad 1 \leq i \leq n$

$$P(t_n = n+1) = \underbrace{\frac{99}{100} \times \frac{99}{100} \times \cdots \times \frac{99}{100}}_{n} = \left(\frac{99}{100}\right)^n$$

$$E[t_n] = \sum_{i=1}^{n+1} i \times P(t_n = i)$$

Let $\underbrace{S = \sum_{i=1}^{n} i(0.99)^{i-1}}_{A}$. Then $\underbrace{0.99S = \sum_{i=1}^{n} i(0.99)^{i}}_{B}$

$$A - B = S - 0.99S = 0.01S$$

$$\Rightarrow$$

$$0.01S = \overbrace{1 + 2(0.99) + 3(0.99)^2 + \cdots + n(0.99)^{n-1}}^{A}$$
$$- [\underbrace{0.99 + 2(0.99)^2 + \cdots + n(0.99)^n}_{B}]$$
$$= 1 + 0.99 + (0.99)^2 + (0.99)^3 + \cdots + (0.99)^{n-1} - n(0.99)^n$$

sum of geometric series

$$= \sum_{i=0}^{n-1} (0.99)^i - n(0.99)^n$$

$$= \frac{1-(0.99)^n}{1-0.99} - n(0.99)^n$$

$$= 100 - (100+n)(0.99)^n$$

Two Computational Approaches

Approach 1: Direct Computation

$\mathbb{E}[t_n] = \sum_{i=1}^{n} i \times Pr[t_n = i]$

Approach 2: Indicator Random Variables

Define indicator random variables $X_1, X_2, ..., X_m$ s.t.:

- $X = X_1 + X_2 + ... + X_m$;
- Each X_i has only two possible values: 0 or 1.

Then $\mathbb{E}[X]$ is computed as follows:

$$\begin{aligned}\mathbb{E}[X] &= \mathbb{E}[X_1 + ... + X_m] \\ &= \mathbb{E}[X_1] + ... + \mathbb{E}[X_m] && \text{(by linearity of expectation)} \\ &= Pr[X_1 = 1] + ... + Pr[X_m = 1]\end{aligned}$$

where the last equality holds because for each X_i:
$\mathbb{E}[X_i] = 0 \times Pr[X_i = 0] + 1 \times Pr[X_i = 1] = Pr[X_i = 1]$.

Indicator Random Variables – Example

Assumption: For each key in the linked list, we pick an integer between 1 and 100 (inclusive), *independently, uniformly at random*.

L is a linked-list

```
def LinkedSearch(L):
1   z = L.head
2   while z != None and z.key != 42:
3       z = z.next
4   return z
```

$x_1 = 1$ iff Line 2 is executed at least 1 time

$x_2 = 1$ iff " " " " " " 2 times

$x_3 = 1$ " " " " " " " " 3 "

⋮ n "

$x_n = 1$ " " " " " " "

$x_{n+1} = 1$ " " " " " " " n+1 "

$$t_n = x_1 + x_2 + x_3 + \cdots + x_n + x_{n+1}$$

$$\begin{array}{cccc} \downarrow & \downarrow & \downarrow & \downarrow \\ 3 & 1 & 1 & 1 \end{array}$$

$$P(x_1 = 1) = 1$$

$P(x_2 = 1) = P($ L[0] is not 42 $)$

$$= \frac{99}{100}$$

$P(x_3 = 1) = P($ L[0] and L[1] are not 42 $)$

$$= \frac{99}{100} \times \frac{99}{100}$$

$$P(x_i = 1) = \left(\frac{99}{100}\right)^{i-1} \qquad 1 \leqslant i \leqslant n+1$$

$$E[t_n]$$

Direct Computation vs. Indicator Random Variables

- Which method to use?
  - Sometimes one method would be easier than the other. Try both, and see whether you get stuck.
  - You'll slowly develop intuition for which method will work for which problem.

Average-Case Running Time Analysis – Take-Home Exercise

L is a list

Define S_n

```
def EvilEvens(L):
  if every number in L is even:
     repeat L.length times:
        calculate and print the sum of L
     return 1
  else:
     return 0
```

Identify possible values for t_n

Calculate the probability of each value t_n takes:

Calculate $E[t_n]$

$$= \underbrace{n^2 \left(\frac{1}{2}\right)^n}_{\Theta(1)} + n$$

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

Summary

- **This week we learned / reviewed**
  - ADT and Data structures
  - Best-case, worst-case, average-case analysis
  - Asymptotic upper/lower bounds
- **What should you do this week?**
  - Complete the Probability Review worksheet.
  - Complete Quiz 0 (deadline Friday at 10pm).
  - Start working on Assignment 1.
- **Next week**
  ADT: Priority queue, Data structure: Heap