Lecture 1

Part A

Measuring Running Time via Experiments

Example Experiment

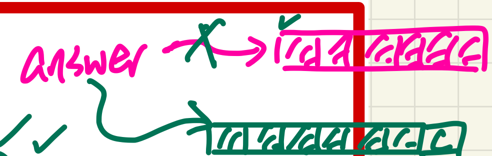
Computational Problem:

- **Input**: A character *c* and an integer *n*
- **Output**: A string consisting of *n* repetitions of character *c*
  e.g., Given input `'*'` and `15`, output `***************`.

Algorithm 1 using *String* Concatenations:

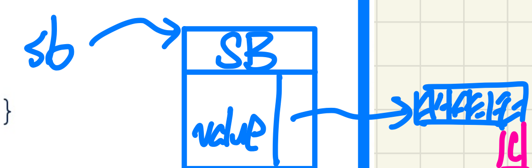
```
public static String repeat1(char c, int n) {
  String answer = "";
  for (int i = 0; i < n; i ++) { answer += c; }
  return answer; }
```

answer = answer + c



Algorithm 2 using *StringBuilder* `append`'s:

```
public static String repeat2(char c, int n) {
  StringBuilder sb = new StringBuilder();
  for (int i = 0; i < n; i ++) { sb.append(c); }
  return sb.toString(); }
```



Lecture 1

Part B

Counting Primitive Operations

Primitive Operation (taking constant time)

Attribute Access

look up the address stored in reference variable acc

$\boxed{\text{acc}}$. balance

context object

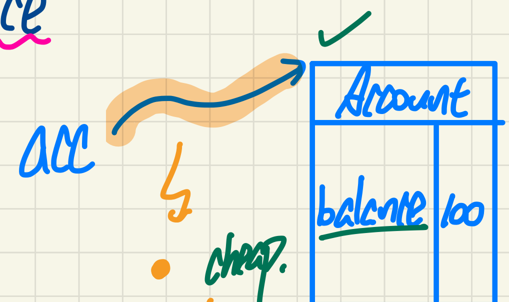
Account acc

reference type

In general, you may have to access an attribute using "multiple dots":

obj . a1 . a2 . a3 . . . . an

constant time prim. op.



• cheap. (constant time operation)

Method Call

obj.m();

Case 1 PO

```
void m() {
    ______
    ______
    ______
    ______
    ______
    ______
}
```

all primitive operations

Case 2 non-PO

```
void m() {
  :
  for(int i = i < a.length; i++)
    :
    :
  m2();
}
```

int findMax(int[] a, int n) ✓ assumed to be a.length

int[] seq = { 2, -1, 3, 1, 10 }

findMax(seq, 5)

seq.length.

Example 1: Counting Number of Primitive Operations

```
1      int findMax (int[] a, int n) {
2  2     currentMax = a[0];
3  n+1   for (int i = 1; i < n; ) {
4  (n-1)·2   if (a[i] > currentMax) {
5  (n-1)·2     currentMax = a[i]; }
6  (n-1)·2   i ++ }
7  1     return currentMax; }
```

n = 5

$1 \ast n$

a → [0 | 1 | 2 | 3 | 4]

i: 1, 2, 3, 4 — n-1 times — i<n T

5 — i<n F

i++

i = i + 1

2 POs.

$2 + n+1 + (n-1)\cdot 6 + 1$

$= (n+3) + (6n-6) + 1$

$= 7n - 2$

Q. # of times **i < n** in **Line 3** is executed?

i<n evaluates to T ← $\boxed{(n-1)} + \boxed{1} = n$ → i<n evaluates to F.

Q. # of times **loop body** (**Lines 4 to 6**) is executed?

$n - \boxed{1}$ → i<n → F

↳ exit from loop

Example 2: Counting Number of Primitive Operations

```
1    boolean foundEmptyString = false;
2    int i = 0;
3    while (!foundEmptyString && i < names.length) {
4      if (names[i].length() == 0) {
5        /* set flag for early exit */
6        foundEmptyString = true;
7      }
8      i = i + 1;
9    }
```

String[] names;

method call (in this case a PO).

T

i
0
1
2
⋮
names.length-1
names.length

i < names.length (10)

T

i < ns. length

F

Q. # of times Line 3 is executed?

T

F.

$\boxed{\text{names.length}}$ + 1

Q. # of times loop body (Lines 4 to 8) is executed?

names.length

Q. # of POs in the loop body (Lines 4 to 8)?

<u>Lecture 1</u>

<u>Part C</u>

Asymptotic Upper Bound

multiplicative constants

highest power

$$\boxed{7n^1 + 2n^1 \cdot \log n} + 3n^{2}$$

lower terms.

approx.

$$\downarrow$$

$$n^2$$

Asymptotic Upper Bound: Big-O

need to be identified in order to prove that $f(n) \in O(g(n))$

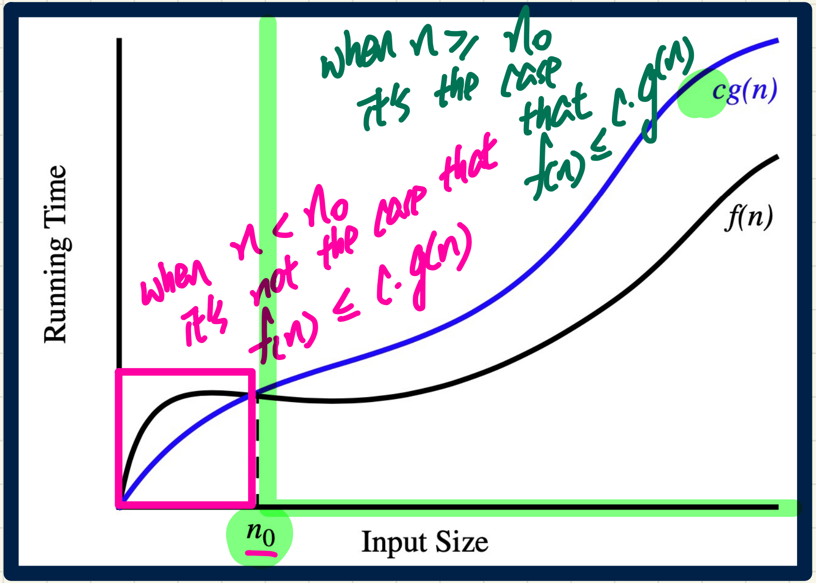
$f(n) \in O(g(n))$ if there are:

- A real *constant* $c > 0$
- An integer *constant* $n_0 \geq 1$

such that:

$$f(n) \leq c \cdot g(n) \quad \text{for } n \geq n_0$$

upper-bound effect



Example:

$f(n) = 8n + 5$

$g(n) = n$

Prove:

$f(n)$ is $O(g(n))$

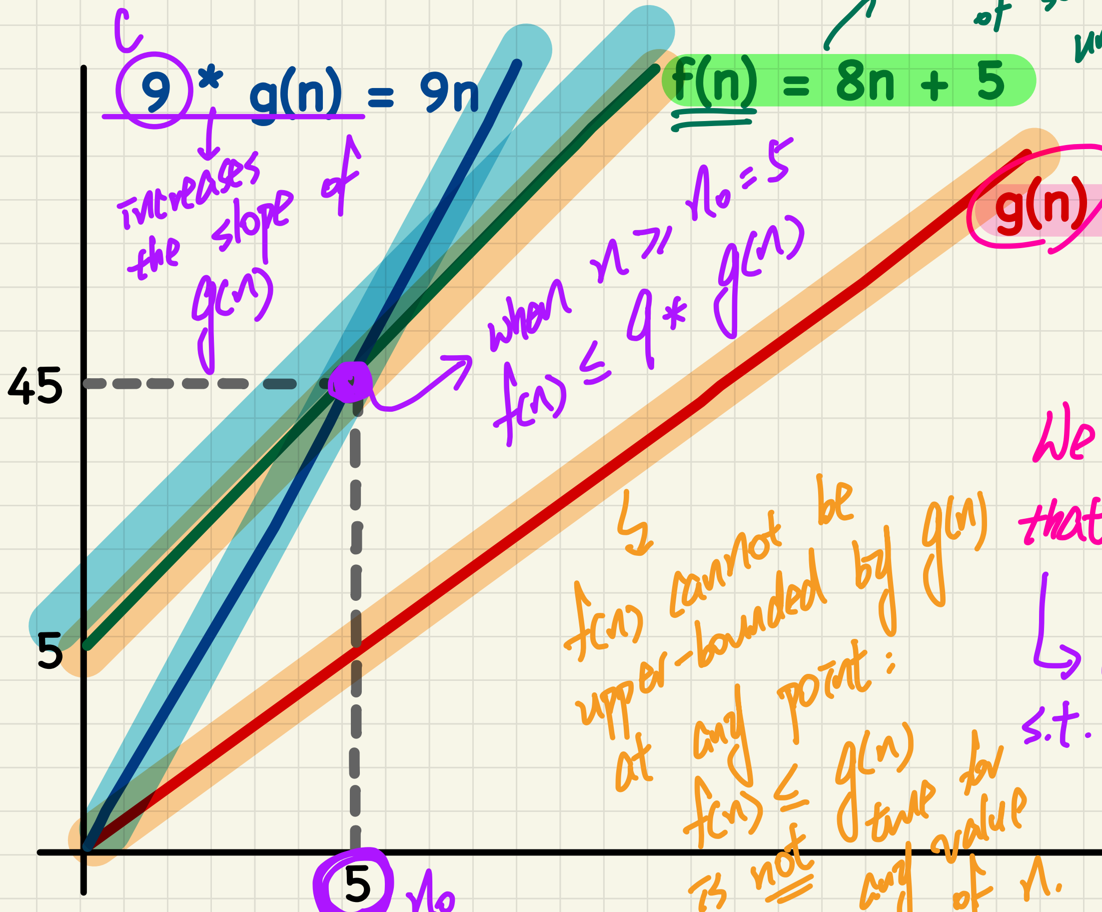
Choose:

$c = 9$

What about n0? 5

Asymptotic Upper Bound: Example

RT function of some algorithm under consideration



Reference function:

We aim at proving that $f(n) \in O(g(n))$

↳ choose c and n_0 s.t.

$f(n) \leq c * g(n)$ for $n \geq n_0$

Proving f(n) is O(g(n))

If $f(n)$ is a polynomial of degree d, i.e.,

$$f(n) = a_0 \cdot n^0 + a_1 \cdot n^1 + \cdots + a_d \cdot n^d$$

and $a_0, a_1, \ldots, a_d$ are integers (i.e., negative, zero, or positive), then $f(n)$ is $O(n^d)$.

We prove by choosing

$$c = |a_0| + |a_1| + \cdots + |a_d|$$
$$n_0 = 1$$

① int i $\quad i \le |i|$

② int i $\quad i^x \le i^y$ $\quad x \le y$

2

$2^3 \le 2^4$

Upper-bound effect: $n_0 = 1$? $\quad [f(\underset{n_0}{1}) \le \underbrace{(|a_0| + |a_1| + \cdots + |a_d|)}_{c} \cdot \underset{n_0}{1}^d]$

$$f(1) = a_0 \cdot 1^0 + a_1 \cdot 1^1 + \cdots + a_d \cdot 1^d$$
$$\le |a_0| \cdot 1^d + |a_1| \cdot 1^d + \cdots + |a_d| \cdot 1^d = (|a_0| + |a_1| + \cdots + |a_d|) \cdot 1^d$$

Upper-bound effect holds? $\quad [f(n) \le \underbrace{(|a_0| + |a_1| + \cdots + |a_d|)}_{c} \cdot n^d]$

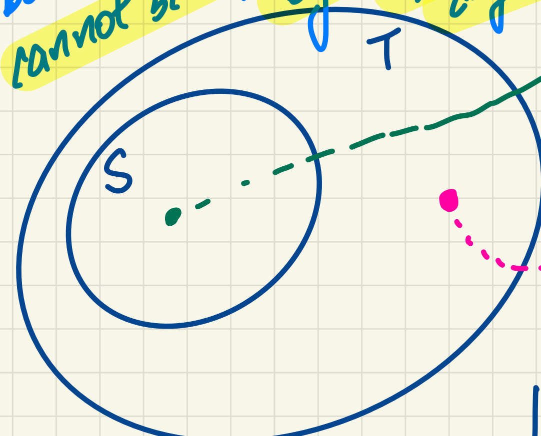
$n \ge n_0 = 1$

$$(n \ge 1) \quad f(n) = a_0 \cdot n^0 + a_1 \cdot n^1 + \cdots + a_d \cdot n^d$$
$$\le a_0 \cdot n^d + a_1 \cdot n^d + \cdots + a_d \cdot n^d \le (|a_0| + |a_1| + \cdots + |a_d|) \cdot n^d$$

Proper Subset

$S \subset T$

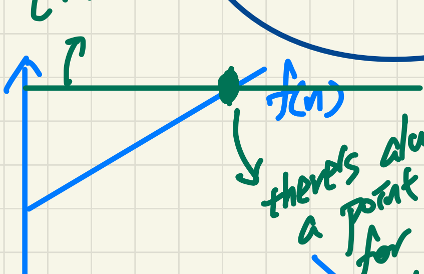
e.g. $f(n) = 3n+1$ is upper bounded by n, but it cannot be upper-bounded by n^0 for any c and n_0.



1. Every member of S is also a member of T.
2. There is at least one member of T that's not a member of S.

If a function is upper-bounded by n^x (e.g. n^1)
can that function be upper-bounded by n^y (e.g. n^0) $(y < x)$?
Not necessarily.

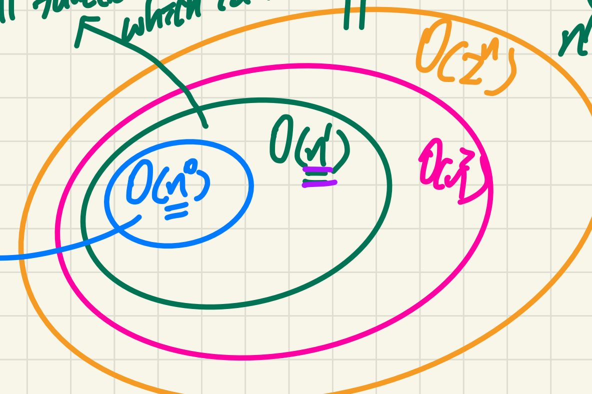
$|S| < |T|$



there's always a point for $f(n) \le c \cdot n^0$ to be false.

all functions which can be upper-bounded by n^0

all functions which can be upper-bounded by n^1

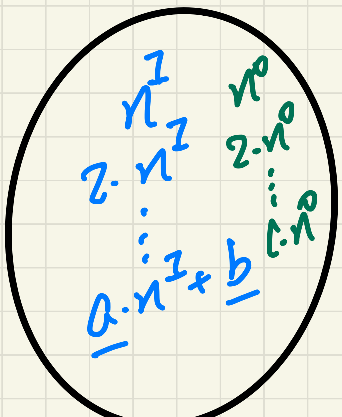


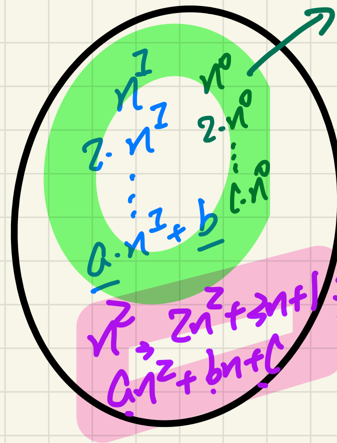
O(g(n)): A Set of Functions

Each member f(n) in O(g(n)) is such that:

Higest Power of f(n) <= Highest Power of g(n)

O(n) ^{1} ⊂ O(n^2)





O(n)

cannot be upper-bounded by $c \cdot n$.

Lecture 1

Part D

Asymptotic Upper Bounds of Math Functions

Asymptotic Upper Bounds: Example (1)

$$5n^2 + 3n \cdot \log n + 2n + 5 \text{ is } O(n^2)$$

$O(n^2)$

$c = |5| + |3| + |2| + |5| = 15$

$n_0 = 1$

$f(n) \leq 15 \cdot n^2 \quad \text{for} \quad n \geq 1$

Asymptotic Upper Bounds: Example (2)

$$20n^3 + 10n \cdot logn + 5 \text{ is } O(n^3)$$

$O(n^3)$

$c = |20| + |10| + |5| = 35$

$n_0 = 1$

$f(n) \leq 35 \cdot n^3$ for $n \geq 1$

Asymptotic Upper Bounds: Example (3)

$$\boxed{3 \cdot \log n + 2 \text{ is } O(\log n)}$$

$O(\underbrace{\log n}_{g(n)})$

$c = |3| + |2| = 5$

$n_0 = \cancel{1} \; \boxed{2}$

$\log 1 = 0 \quad \because 2^0 = 1$

$\log 2 = 1$

$\log n$

$2 * n^0$

$f(n) = 3 \cdot \log n + 2$

$\boxed{f(1)} \leq \boxed{5 \cdot \log 1}$

$\boxed{3 \cdot \log 1 + 2}$ 2   0

False

$f(2) \leq \underbrace{5 \cdot \log 2}_{5}$

$\boxed{3 \cdot \log 2 + 2}$

5

True

Asymptotic Upper Bounds: Example (4)

$$\boxed{2^{n+2} \text{ is } O(2^n)}$$

$$2^{n+2} = \boxed{2^2} \cdot 2^n$$

$$4$$

$$c = |4| = 4$$

$$n_0 = 1$$

Asymptotic Upper Bounds: Example (5)

$2n^1 + 100 \cdot \log n$ is $O(n)$

$O(n)$

$c = |2| + |100| = 102$

$n_0 = 1$

Exercise: Check if the upper-bound effect starts to hold when $n = n_0 = 1$:

$$\boxed{f(1) \le 102 \cdot n}$$