Lecture 4 - Sep 15

Asymptotic Analysis

Defining Big-O using Predicate Logic
Deriving Big-O: Triangular Sum
Dynamic Arrays: Constant Increments

Announcements/Reminders

- First Class (Syllabus) recording & notes posted
- Today's class: notes template posted
- Exercises:
  + Tutorial Week 1 (2D arrays)
  + Tutorial Week 2 (2D arrays, Proving Big-O)

$\mathbb{N} = \{0, 1, 2, \dots\}$

Asymptotic Upper Bound (Big-O): Alternative Formulation

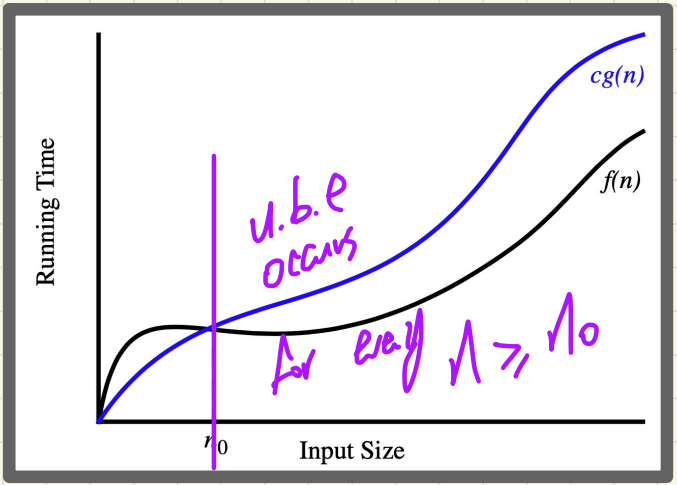
$\Rightarrow$: logical implication

Known:

$f(n) \in O(g(n))$ if there are:
- A real constant $c > 0$
- An integer constant $n_0 \geq 1$

such that:

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



$O(g(n))$

$f(n)$

Q. Formulate the definition of "**f(n)** is order of $O($**g(n)**$)$" using **logical** operator(s): $\neg, \wedge, \vee, \Rightarrow, \forall, \exists$

input size $n \in \mathbb{N}$

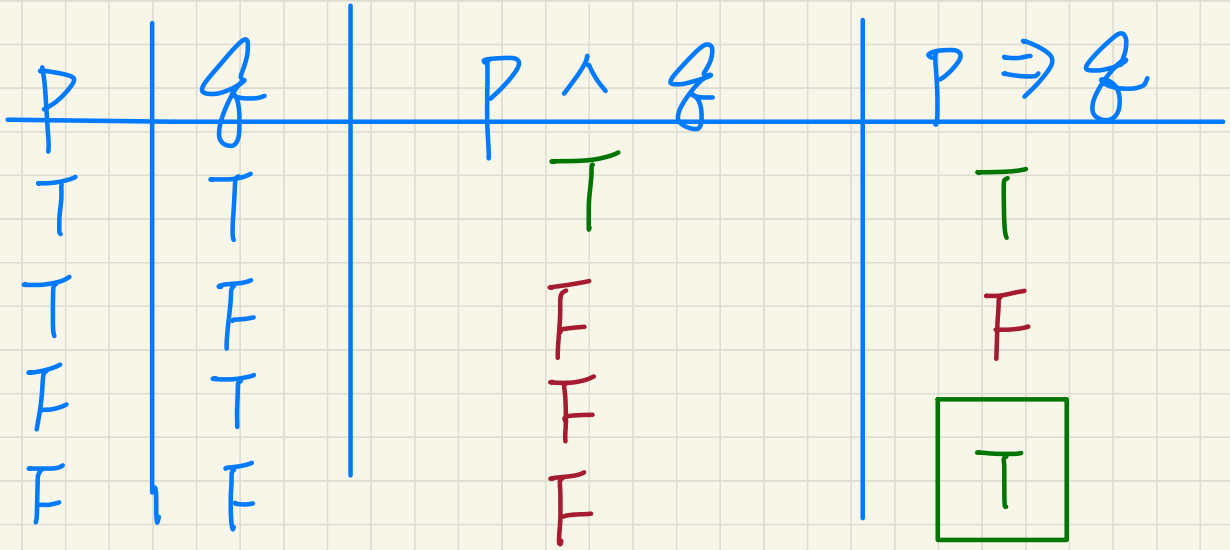
$$f(n) \in O(g(n)) \iff \exists c, n_0 \cdot c > 0 \wedge n_0 \geq 1 \wedge (\forall n \cdot n \geq n_0 \Rightarrow f(n) \leq c \cdot g(n))$$

$c \in \mathbb{Z}$, $c \in \mathbb{N}$

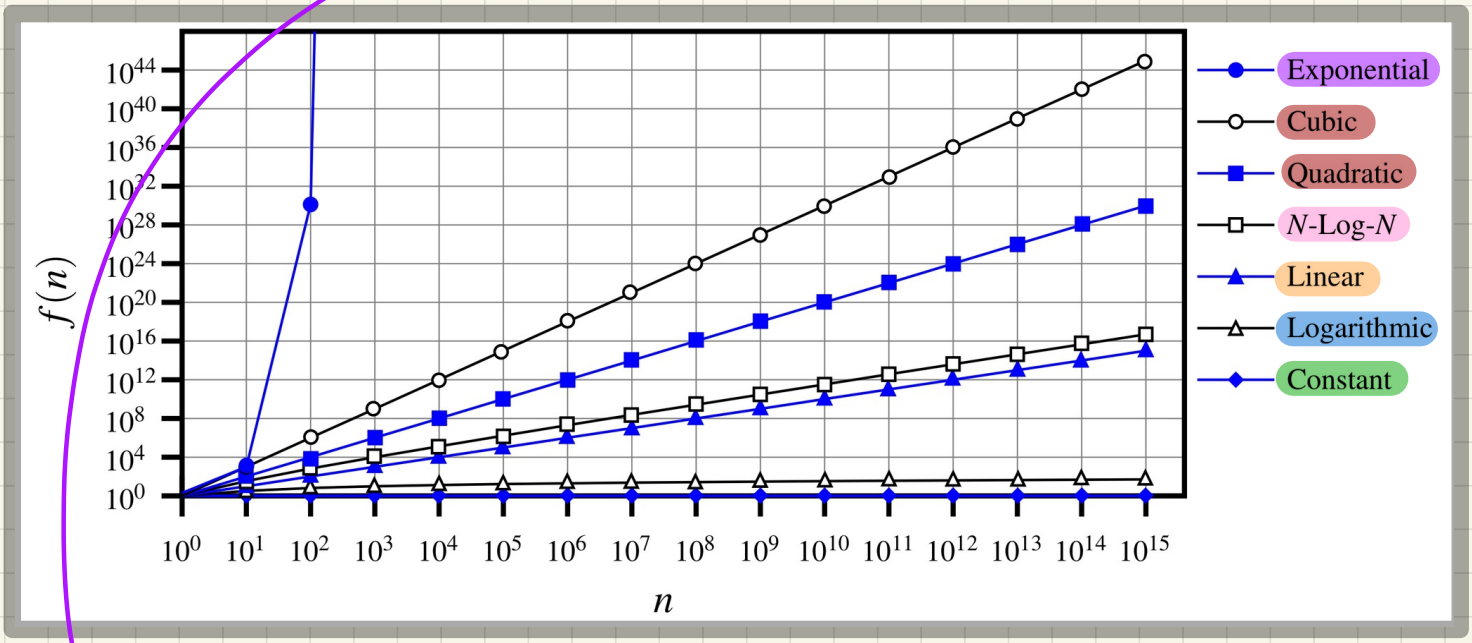
u.b.e.

Q. Why $\Rightarrow$ as opposed to $\wedge$

Hint Consider the truth tables



RT Functions: Rates of Growth (w.r.t. Input Sizes)



the slower (flatter) relative to input size more asp, the more efficient.

Size of integer interval

$[a, b] \rightsquigarrow a, a+1, a+2, \dots, b$ — how many?

$[a, b] = b - a + 1$

size $= b - a + 1$

closed end ↳ end value included

e.g. $[34, 100] = 100 - 34 + 1 = 67$

array

$[0, x]$ = array size

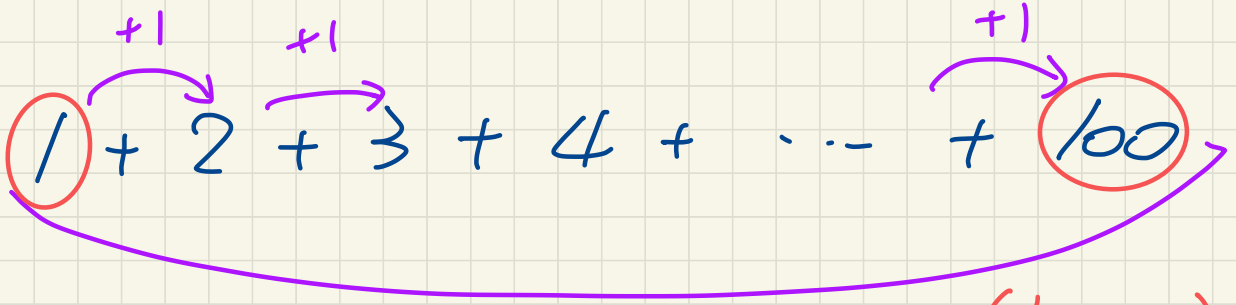
min index: 0

max index: x

$= (x - 0) + 1$

$= x + 1$

Asymptotic Upper Bound: Arithmetic Sequence/Progression



100 terms

$\frac{(1+100)*100}{2}$

n terms

common difference

$\underbrace{i}_{\text{start term}} + \underbrace{(i+c)}_{\text{1st term}} + \underbrace{(i+2\cdot c)}_{\text{2nd term}} + \underbrace{(i+3\cdot c)}_{\text{3rd term}} + \cdots + \underbrace{(i+(n-1)\cdot c)}_{n\text{th term}}$

Remember

$\frac{(\text{1st term} + \text{last term}) * \#\text{ terms}}{2}$

$= \frac{(i + (i+(n-1)\cdot c)) * n}{2} = \frac{c\cdot n^2 + (2i-c)n}{2}$

is $O(n^2)$

Determining the Asymptotic Upper Bound (3)

```
1  int triangularSum (int[] a, int n) {
2    int sum = 0;                          1
3    for (int i = 0; i < n; i ++) {
4      for (int j = i; j < n; j ++) {
5        sum += a[j];                      1 } }
6    return sum; }                         1
```

$O(n^2)$

Each primitive op. takes $O(1)$ time

Each combination of i and j corresponds to an exec. of L5.

Patterns of (i, j)

i \ j	0	1	2	...	n-1
0	0	1	2	...	n-1
1		1	2	...	n-1
2			2	...	n-1
⋮					
n-1					n-1

# of i: $[0, n-1] = n$

combination of (i, j) = (0, 2)

$[0, n-1] = n$

$[1, n-1] = n-1$

$[2, n-1] = (n-1)-2+1 = n-2$

⋮

$[n-1, n-1] = 1$

Executions of L5:

$= n + (n-1) + (n-2) + \dots + 1$

$= \frac{(n+1) \cdot n}{2}$ is $O(n^2)$

$O(1 + n^2 + 1) = O(n^2)$

L2 ~ L3~L5 ~ L6

Implementing Stack / Queue

1. Using an array with some capacity MAX

s.push(..) s.push(..) - - - s.push

MAX pushes

s.push

(MAX+1)th push

↳ precondition violation (StackFullExcep.)

2. Using a dynamic array with "adapting" cap.

↑ grow the size of the array when necessary

s.push(...)
s.push(...)
⋮

no worry about stack full.

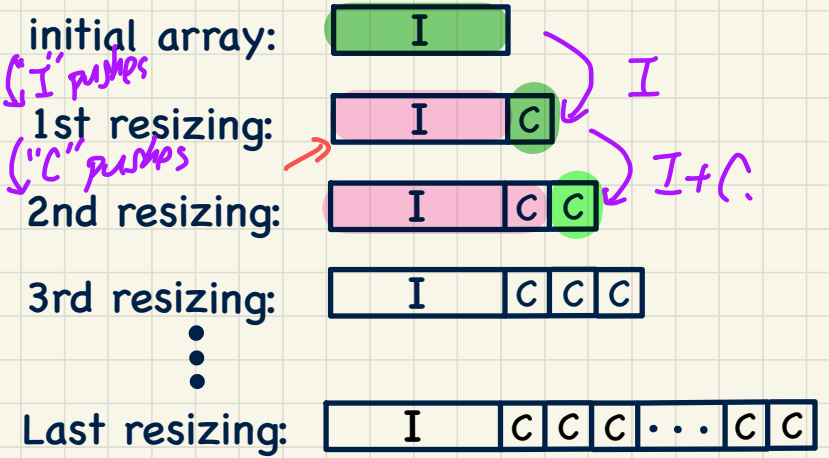
2.1 constant increments
2.2 doubling

the one that demands less frequent resizing is asymptotically more efficient

n pushes

Amortized Analysis: Dynamic Array with Const. Increments

```
1  public class ArrayStack<E> implements Stack<E> {
2    private int I;
3    private int C;
4    private int capacity;
5    private E[] data;
6    public ArrayStack() {
7      I = 1000; /* arbitrary initial size */
8      C = 500; /* arbitrary fixed increment */
9      capacity = I;
10     data = (E[]) new Object[capacity];
11     t = -1;
12   }
13   public void push(E e) {
14     if (size() == capacity) {
15       /* resizing by a fixed constant */
16       E[] temp = (E[]) new Object[capacity + C];
17       for(int i = 0; i < capacity; i ++) {
18         temp[i] = data[i];
19       }
20       data = temp;
21       capacity = capacity + C
22     }
23     t++;
24     data[t] = e;
25   }
26 }
```



W.L.O.G, assume: n pushes

and the **last** push triggers the **last** **resizing** routine.

Amortized/ Average RT: