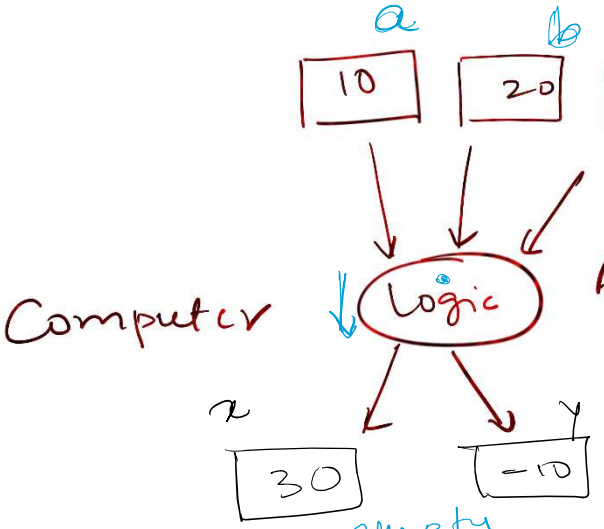
Variable

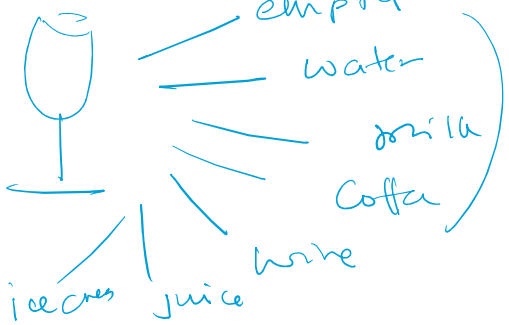


Act on values

to transforming

one set of values

to another set of values



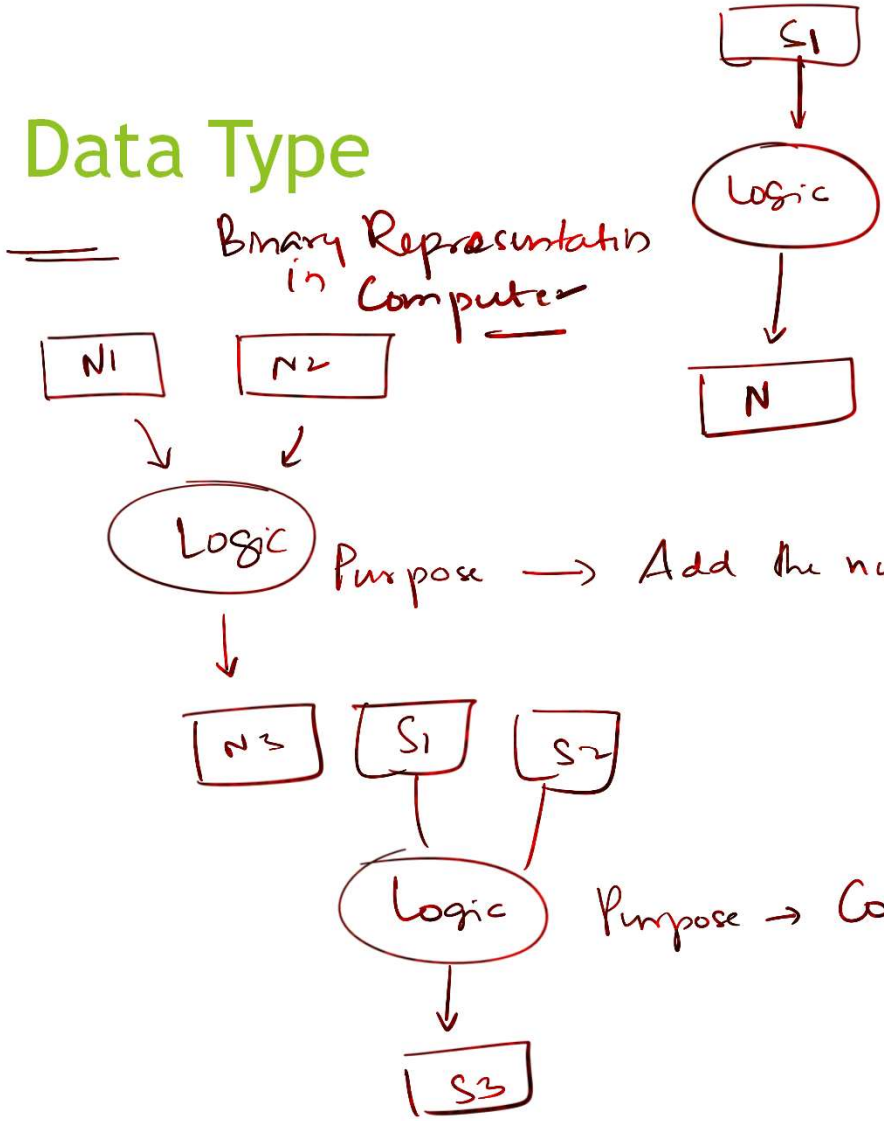
Contents of the glass is variable

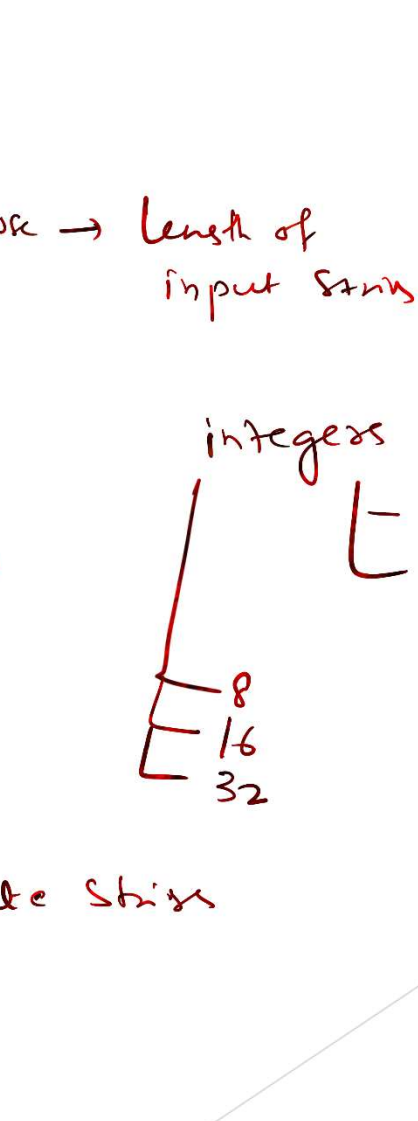
(a) → pointer to the location that contains value

→ Container of values

→ its value can vary over the run time

Data Type





integers → 8
- Signed → 2^{-7} to 2^{7}
- Unsigned 0 to 2^{8}

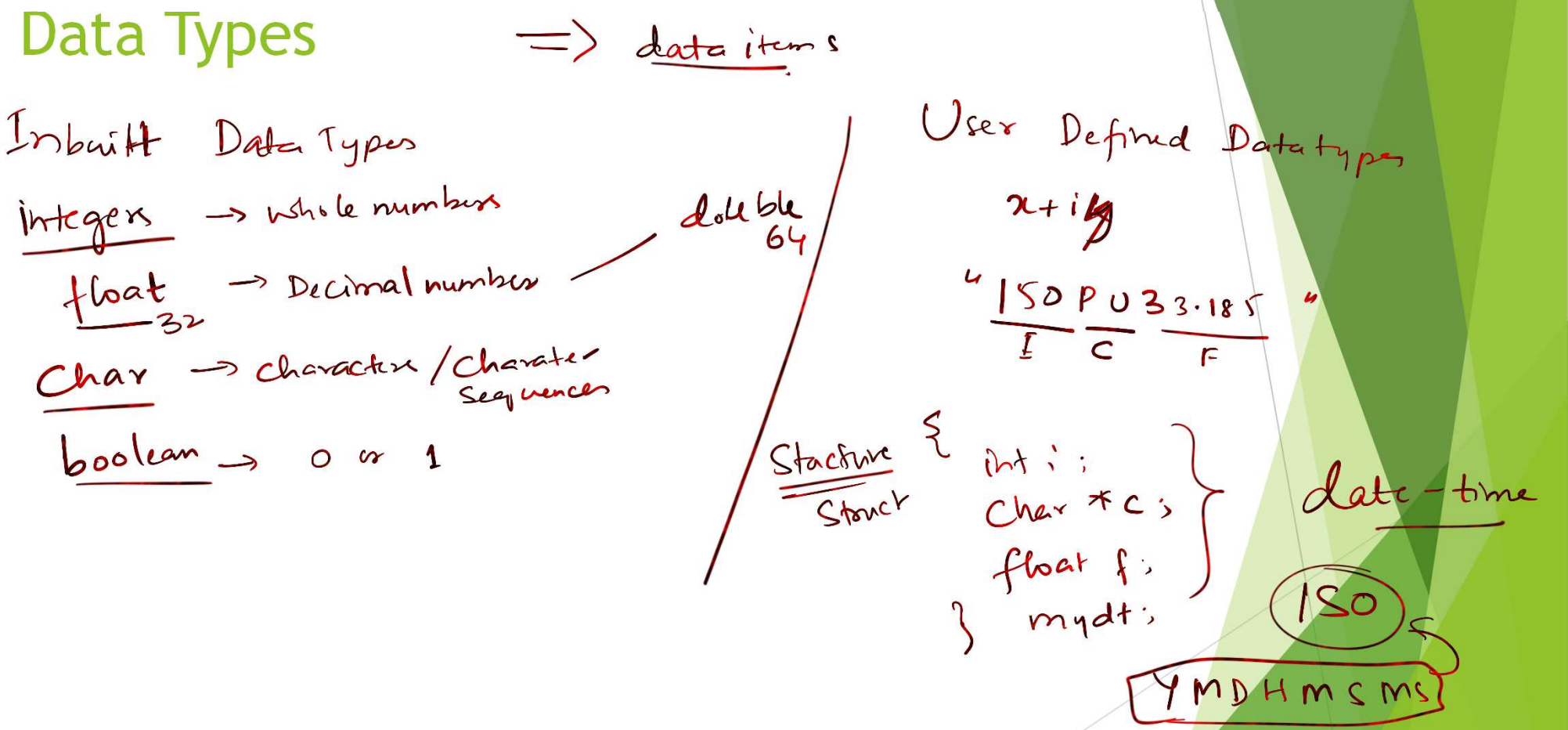
8
16
32

−127 to 128
0 to 256

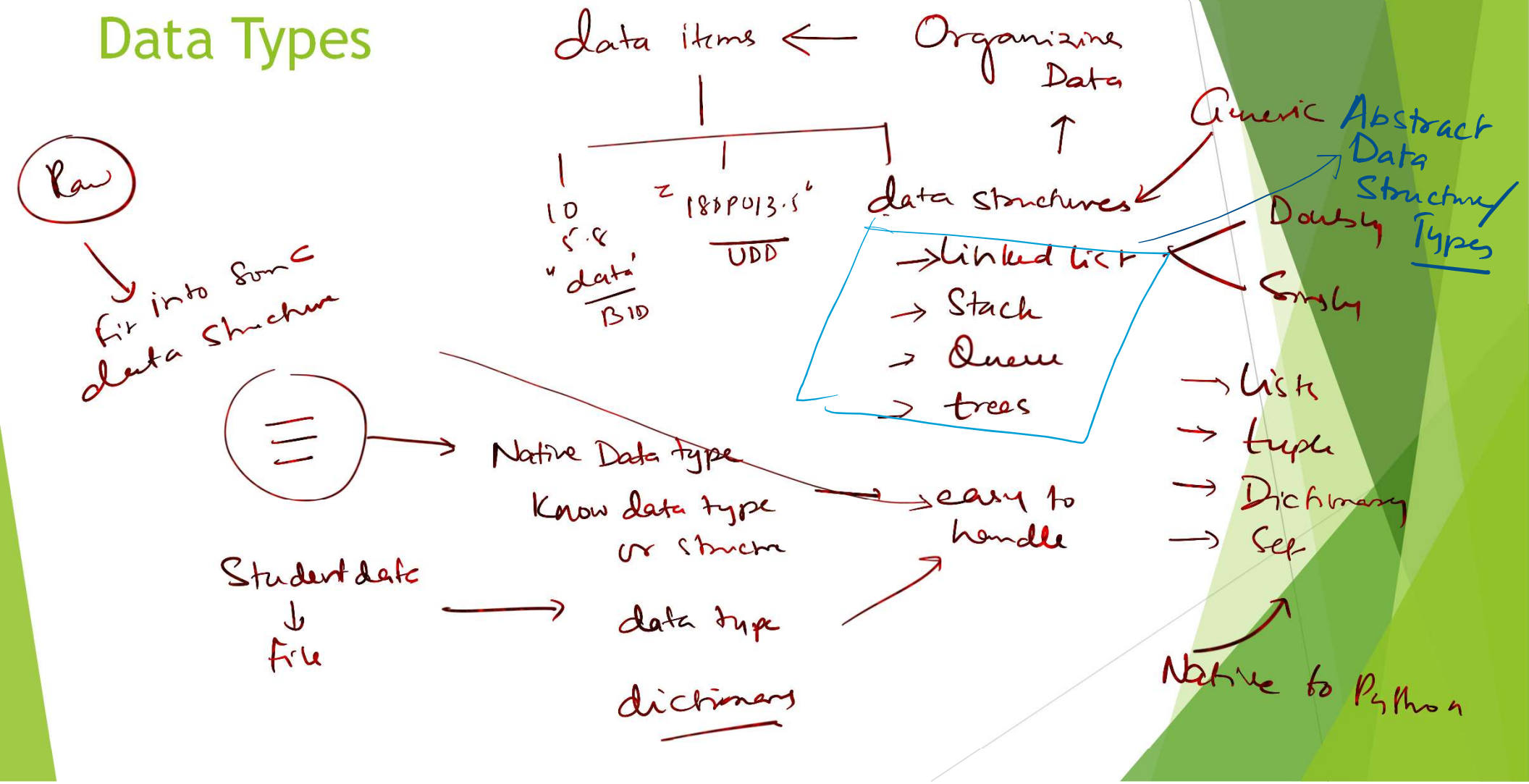
uint8
int8

uint8 i;

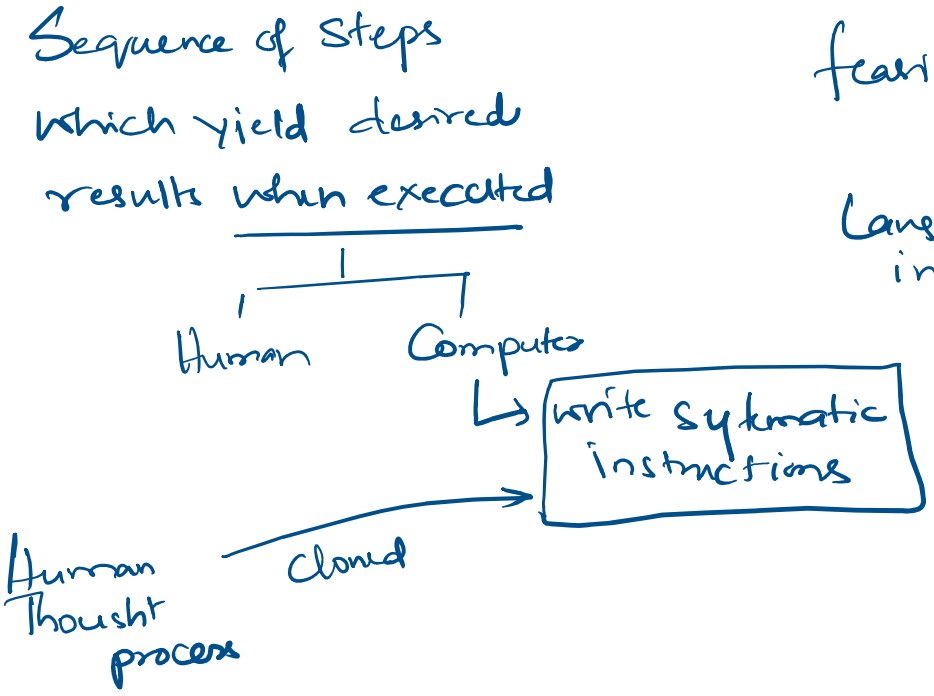
Data Types

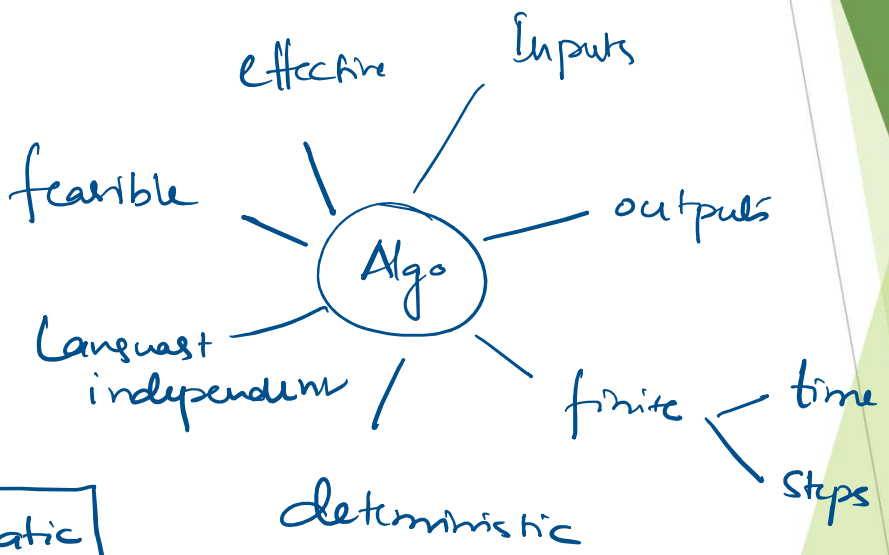

=> data items
Inbuilt Data Types
integers → whole numbers
float → Decimal numbers
32
double
64
Char → characters / Charater sequences
boolean → 0 or 1
User Defined Datatypes
x+iy
"ISO PU 33.185"
I C F
Structure
Struct
{
int i;
char *c;
float f;
} mydt;
date-time
ISO
YMDHMSMS

Data Types

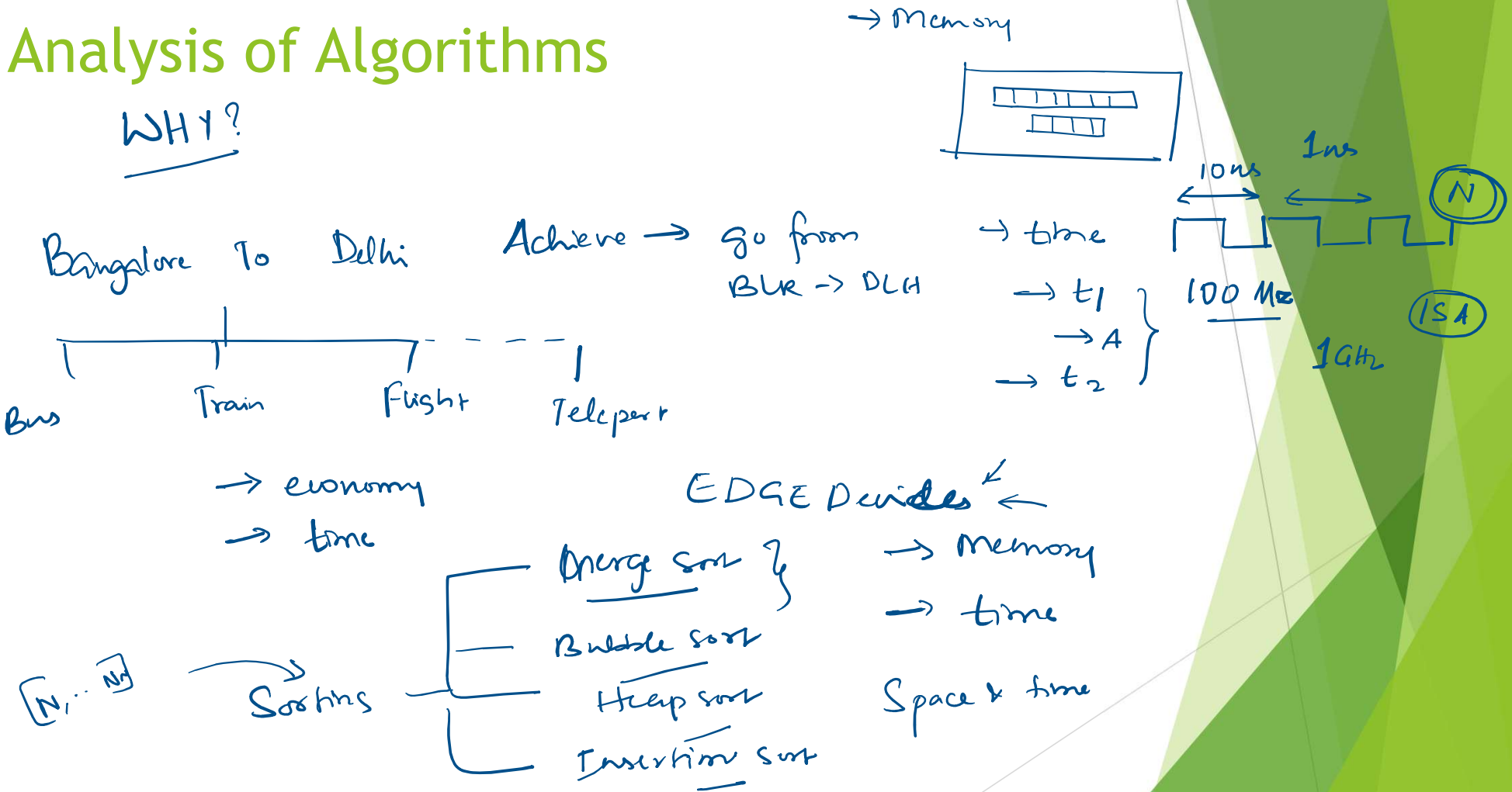

data items
Organizing Data
Raw
fit into some data structure
ID
"data"
BID
"18APO13.S"
UDD
data structures
Generic Abstract Data Structures
Doubly
Types
Singly
linked list
Stack
Queue
trees
Native Data type
Know data type or structure
easy to handle
Student data
file
data type
dictionary
lists
tuple
Dictionary
Set
Native to Python

Algorithm


Sequence of steps
which yield desired
results when executed
Human
Computer
write systematic
instructions
Human
Thought
process
cloud


effective
Inputs
feasible
Algo
outputs
Language
independent
deterministic
finite
time
Steps

Analysis of Algorithms



Goal of Analysis

→ time

→ memory

→ effort of developer

Time Analysis

How processing time increases as problem size increases?

What is running time Analysis?

→ t_1

→ Algo

→ t_2

$t_2 - t_1$ = time taken to run Algorithm

→ Size of the input array

→ Polynomial degree

$n+2$

$n^2 + n + 2$

$n^3 + 2n^2 + n + 2$

↓ increasing

→ Number of elements in a matrix

→ Number of bits

→ vertices / edges

How to compare algorithms?

→ Execution time ?

if on same computer ✓

but not an ideal thing,

as are different in different computers

→ Number of Statements ?

→ Sorting

C → 100

Py → 2

→ Run time ≐ f(input size) ✓ → Clock cycles

→ f(input size) (Benchmark)

independent of machine time, coding style

Rate of Growth

Increase input size ∝ Run time

Person — Car 80 Lakhs + bicycle 8000 INR

= 80 8000 1-

≈ 80L

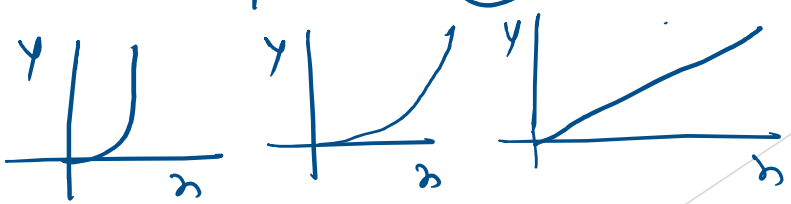
$n = [1, 2, 3, 4, 5 \ldots]$

$n^4 + 3n^3 + 2n^2 + 1 = X$

$\approx n^4$ → highest rate growth

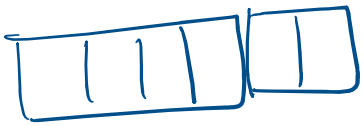
Commonly Used Rate of Growth :

2^{2^n} , $n!$, 2^n , n^2 , n , $n \log n$, $2^{\log n}$, $\log^2 n$, $\sqrt{\log n}$, 1



Common Algorithms and RoG

Time Complexity



→ 1

Search [Sorted Array] → log n

Search [Unsorted Array] → n

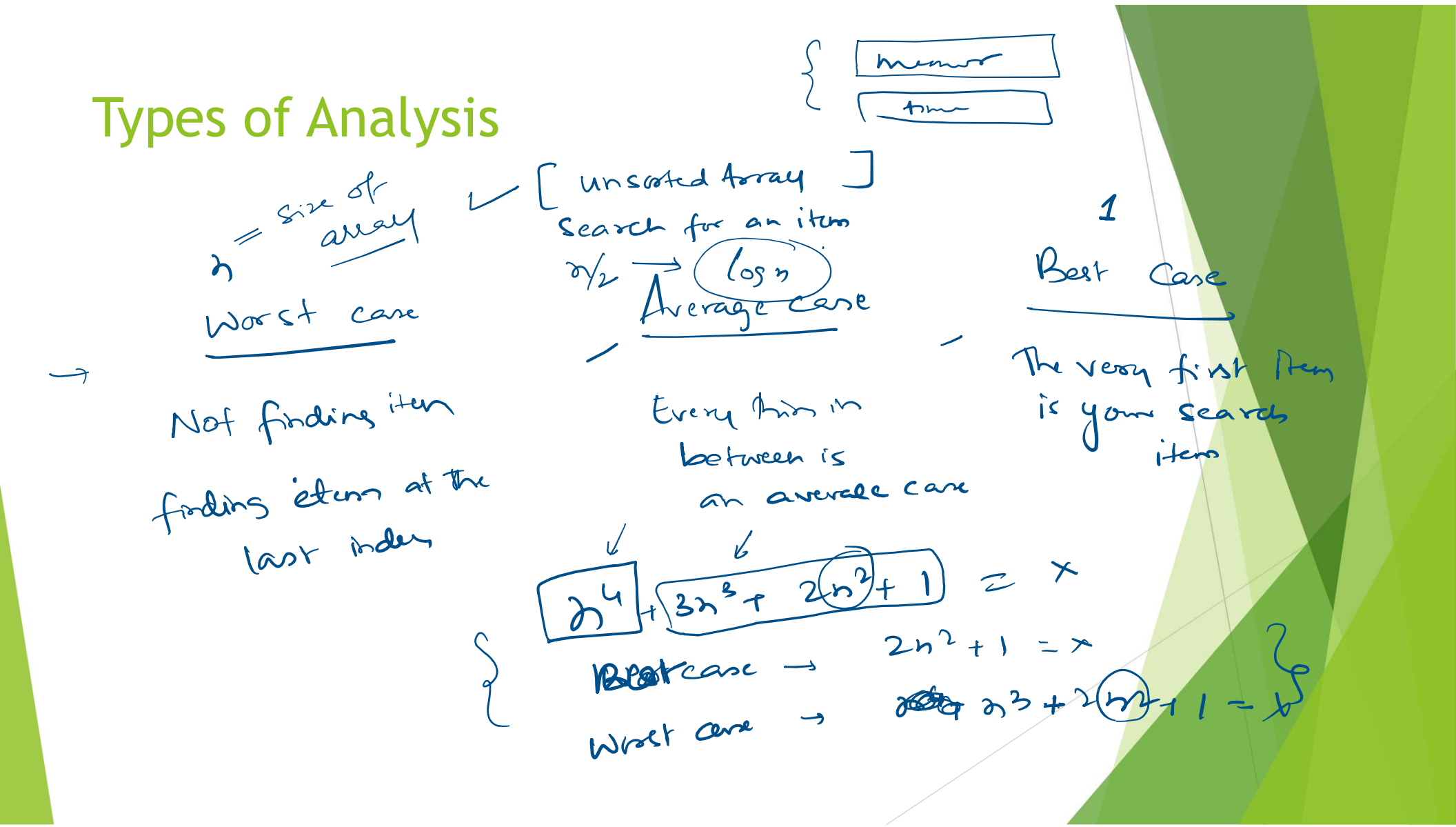
[Sorting | Merge] → n log n

→ n^2

Matrix Multiplication [ ] * [ ] → n^3

Tower's of Hanoi → 2^n

Types of Analysis



Asymptotic Notation

Lunch Break: 1:25 – 2:05
Assignments: 2:05 – 2:25

$\Omega \rightarrow$ Tight Lower Bound

$\Theta \rightarrow$ Upper Bound – Lower Bound

Big-O Notation: Tight Upper Bound of Algorithm

$n^4 + 100n^2 + 10n + 50 = Y$ $\quad g(n) = n^4$ $\quad 2^4 \cdot 3^4 \cdot 100^4$

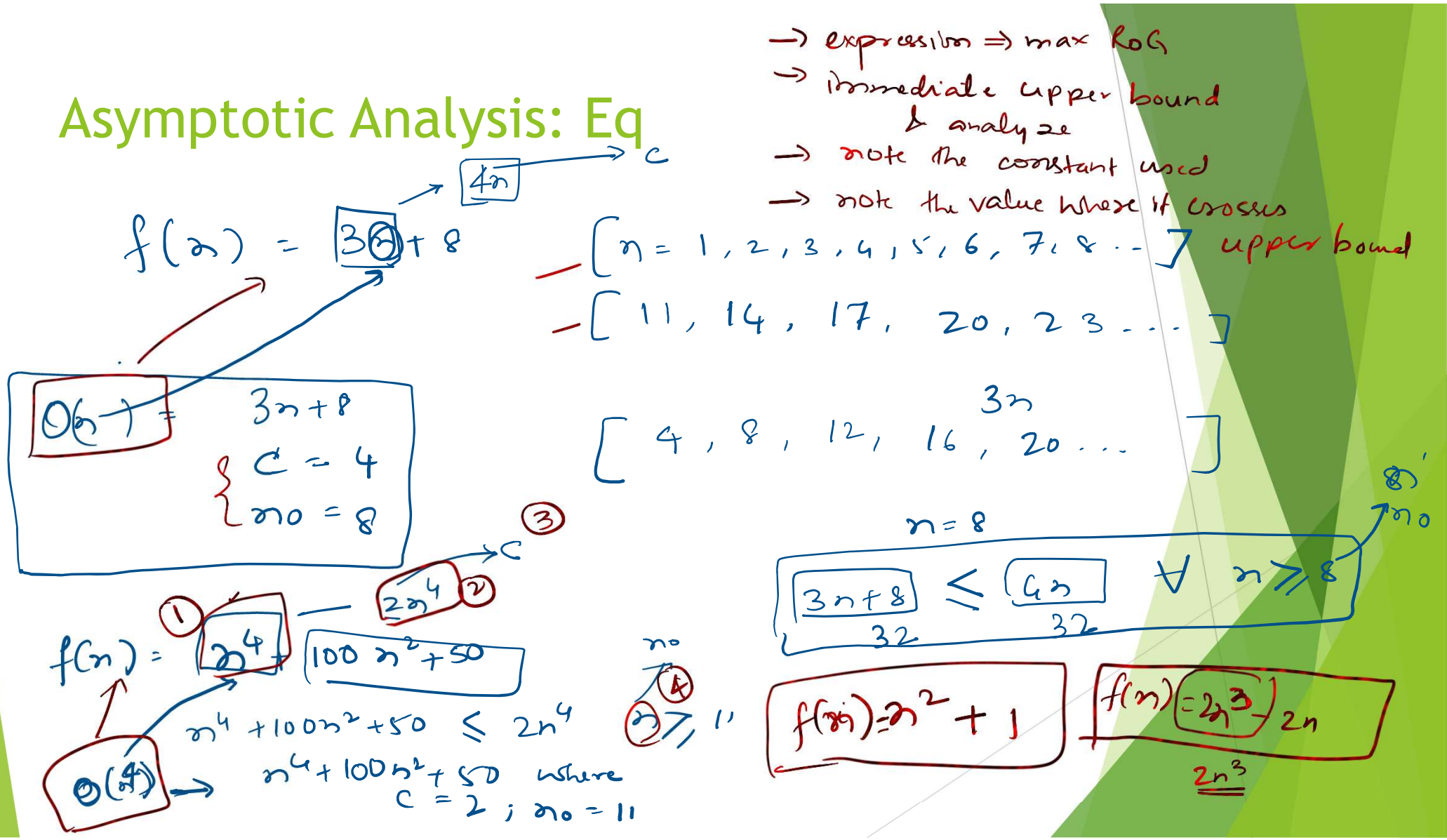
$O(g(n)) = f(n) \rightarrow$ there are positive constants 'c' & $(n_0)\, n'$ such that $0 \leq f(n) \leq c \cdot g(n)$

$\forall \; n \geq n' \Rightarrow n_0$

$n^0 \rightarrow \Rightarrow$ discard all lower values of n

$100, 1.2 \Rightarrow O(1)$ $\quad 3n + 100 \Rightarrow O(n)$ $\quad n^2 - 2n + 1 \Rightarrow O(n^2)$

Asymptotic Analysis: Eq



Asymptotic Analysis: Code

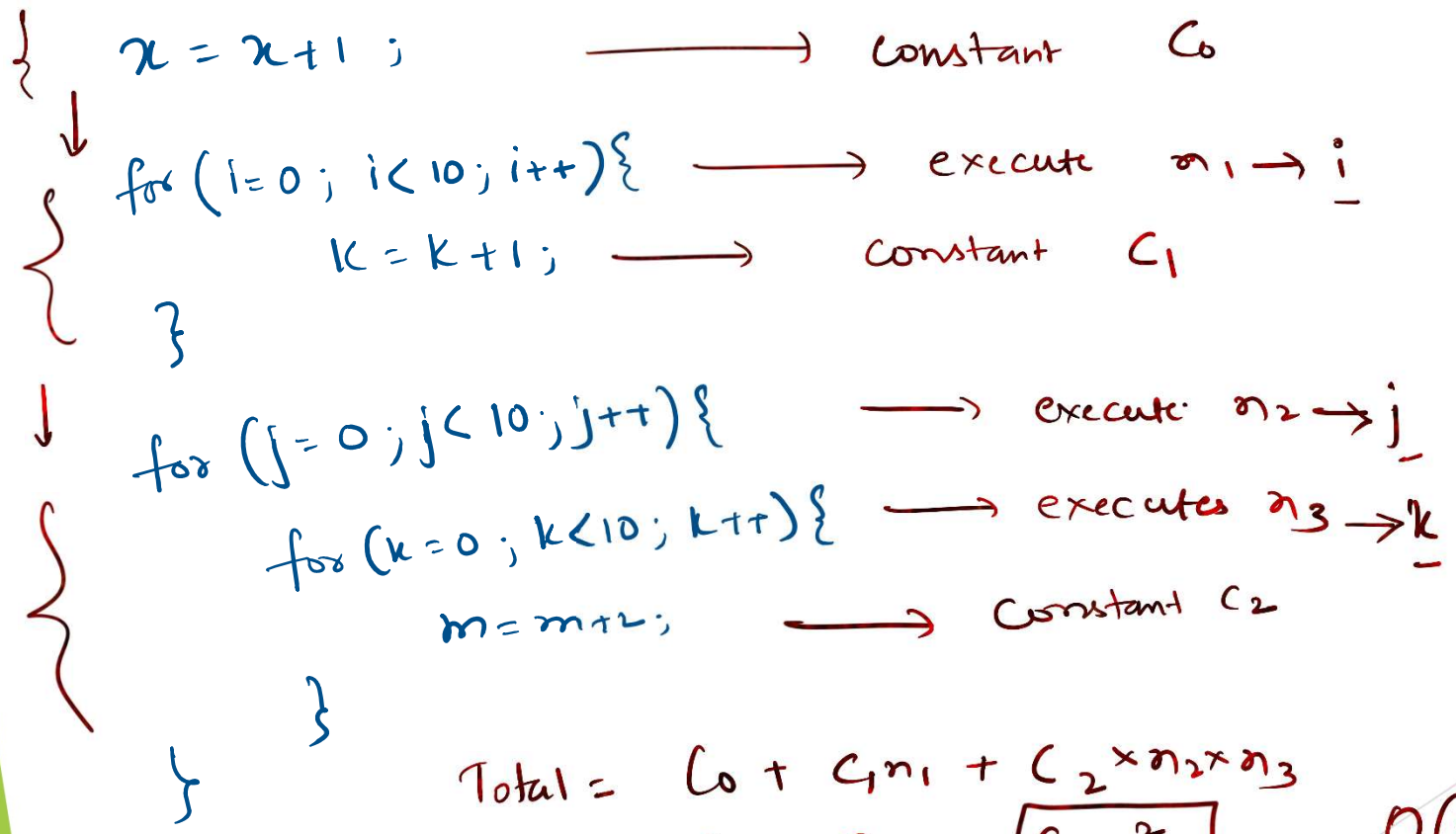
```
for(i=1; i<=n; i++){          → executes for n times
    m = m+2;                  → Constant time, c
}
```

Total = $c \times n = O(n)$

```
for(i=1; i<=m; i++){          → execute for m times
    * for(j=1; j<=n; j++){    → executing for n times
        **x = x+1;            → Constant
    }
}
```

Total = $c \times \frac{m}{n} \times \frac{n}{n} = O(n^2)$

Asymptotic Analysis: Code

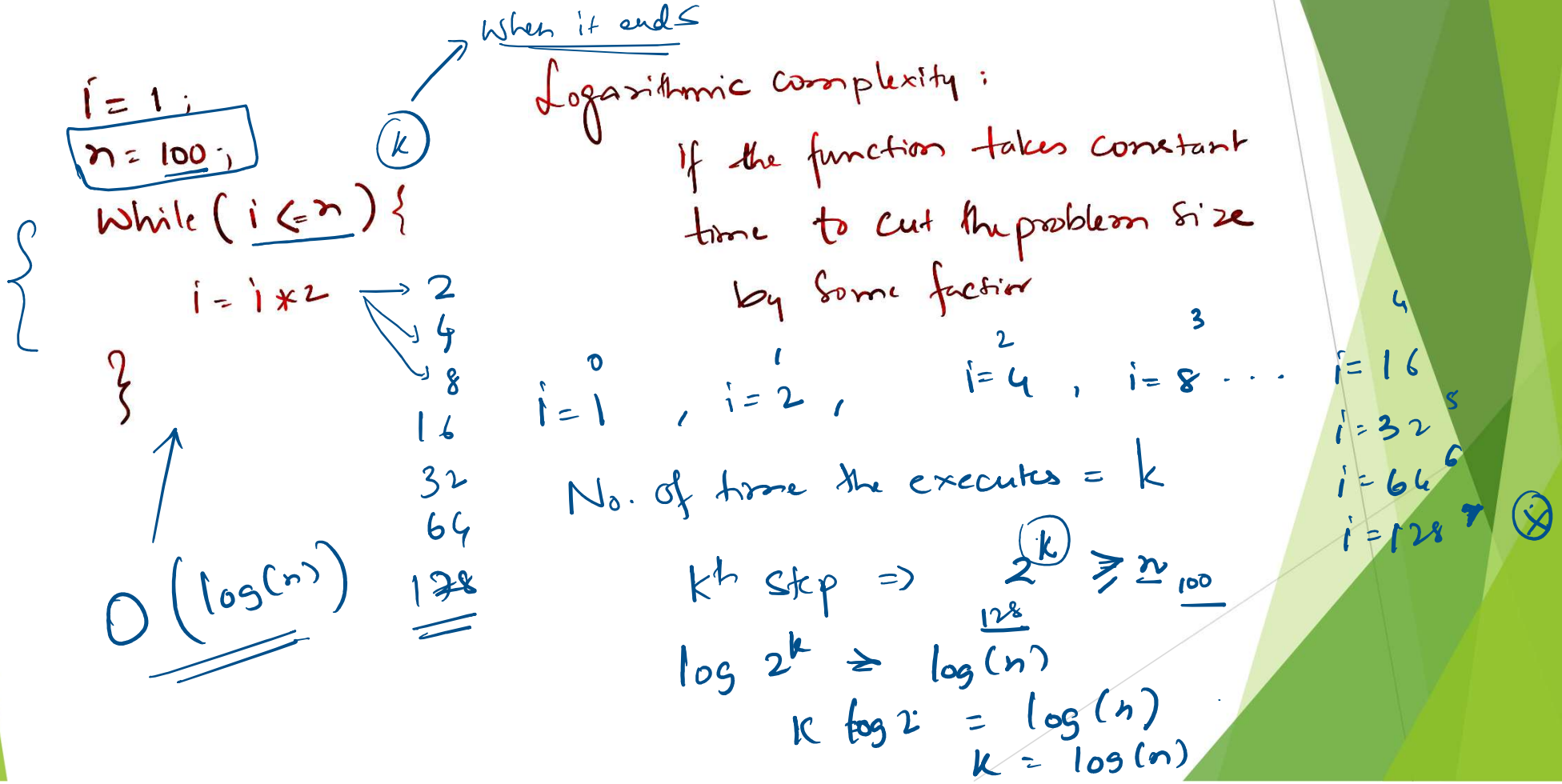


Asymptotic Analysis: Code

```
void myfunc () {
  if (a == 0)                          → constant → C0
    return false;                      → C3
  else {
    for (n=0; n < a; n++) {            → execute n times
      if ( x[n] != y[n] ) {            → constant → C1
        return false                   → constant → C2
      }
    }
}
```

Total = $C_0 + (C_1 + C_2) * n$ $\Rightarrow$ $O(n)$

Asymptotic Analysis: Code



Asymptotic Analysis: Code

```
Void function(int n){
    int i, j, k, count=0;                          → constant
    for(i=n/2; i<=n; i++)                          → execute n/2 time
        for (j=1; j+n/2=n; j=j++)                  → executes n/2 times
            for (k=1; k<=n; k=k*2)                 → executes log n times
                count++;
}
```

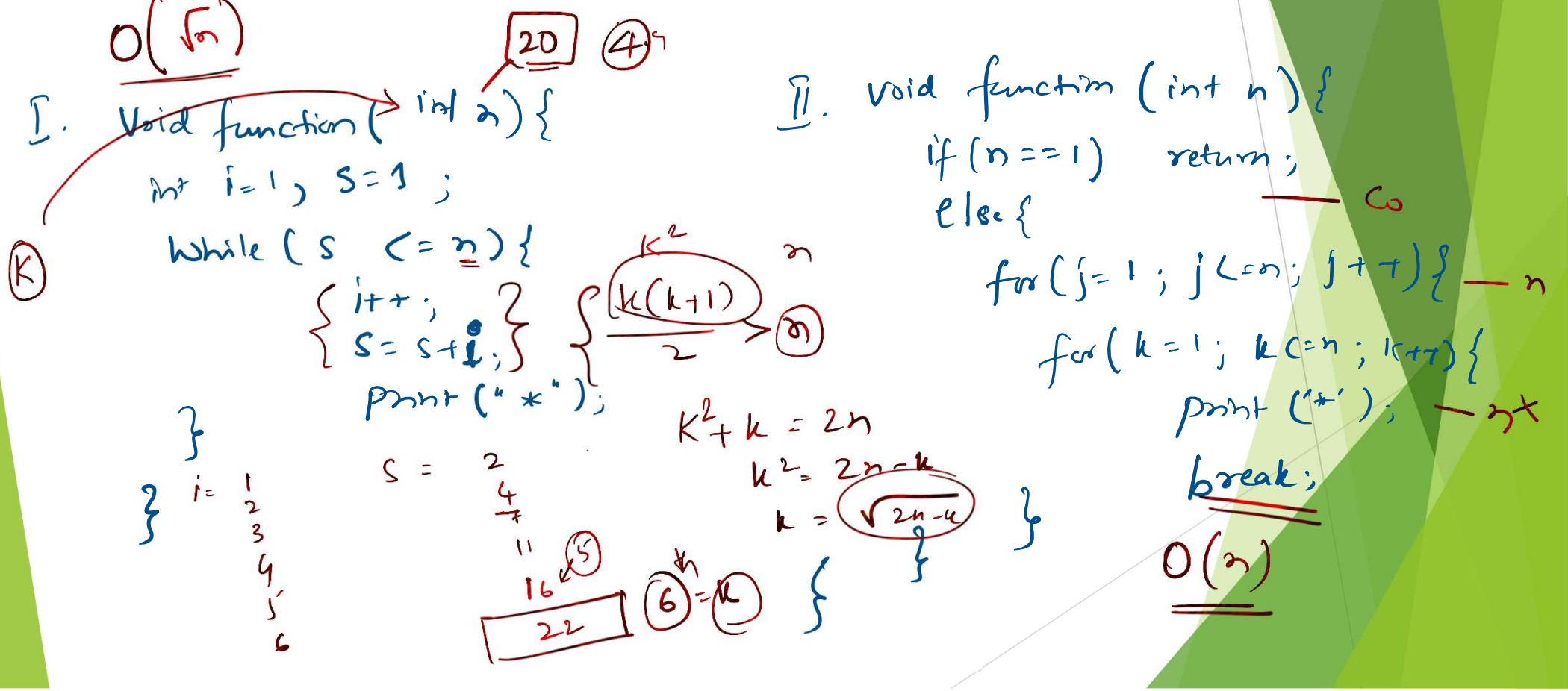
Cost $\frac{n}{2} * n/2 \times \log n$ $O(n^2 \log n)$

Cost $\frac{n^2}{2} \log n$

Cost $1/2\, n^2 \log n$

Asymptotic Analysis: Exercise

3:45 – 4:05 Exercise



Asymptotic Analysis: Recursive Functions

K iteration when the algo stops N

5

```
factorial (n) {
    if (n = 0)
        return 1
    else
        return n * factorial(n-1)
}
```

$k = f(n)$

Get the value of k

Solve for k

5 * 4 * 3 * 2 * 1

factorial(4) 4 * 3 * 2 * 1

factorial (3) 3 * 2 * 1

factorial (2) 2 * 1

factorial (1)

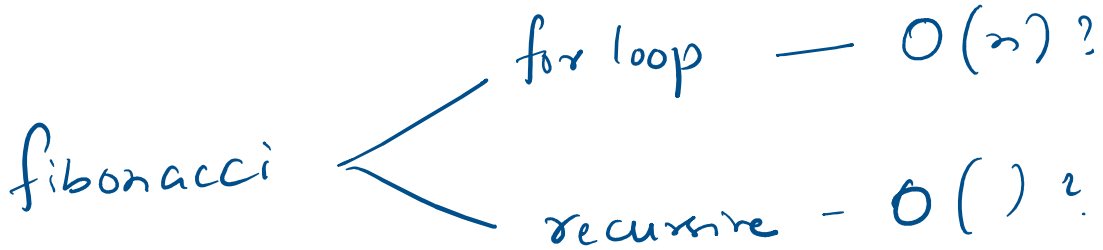
$k = c_0 * n$

stop

$O(n)$

Assignment

- Recursive Function and Efficiency Analysis - Write a recursive function pseudocode and calculate the nth Fibonacci number and use Big O notation to analyze its efficiency.
- Compare this with an iterative approach and discuss the pros and cons in terms of space and time complexity.

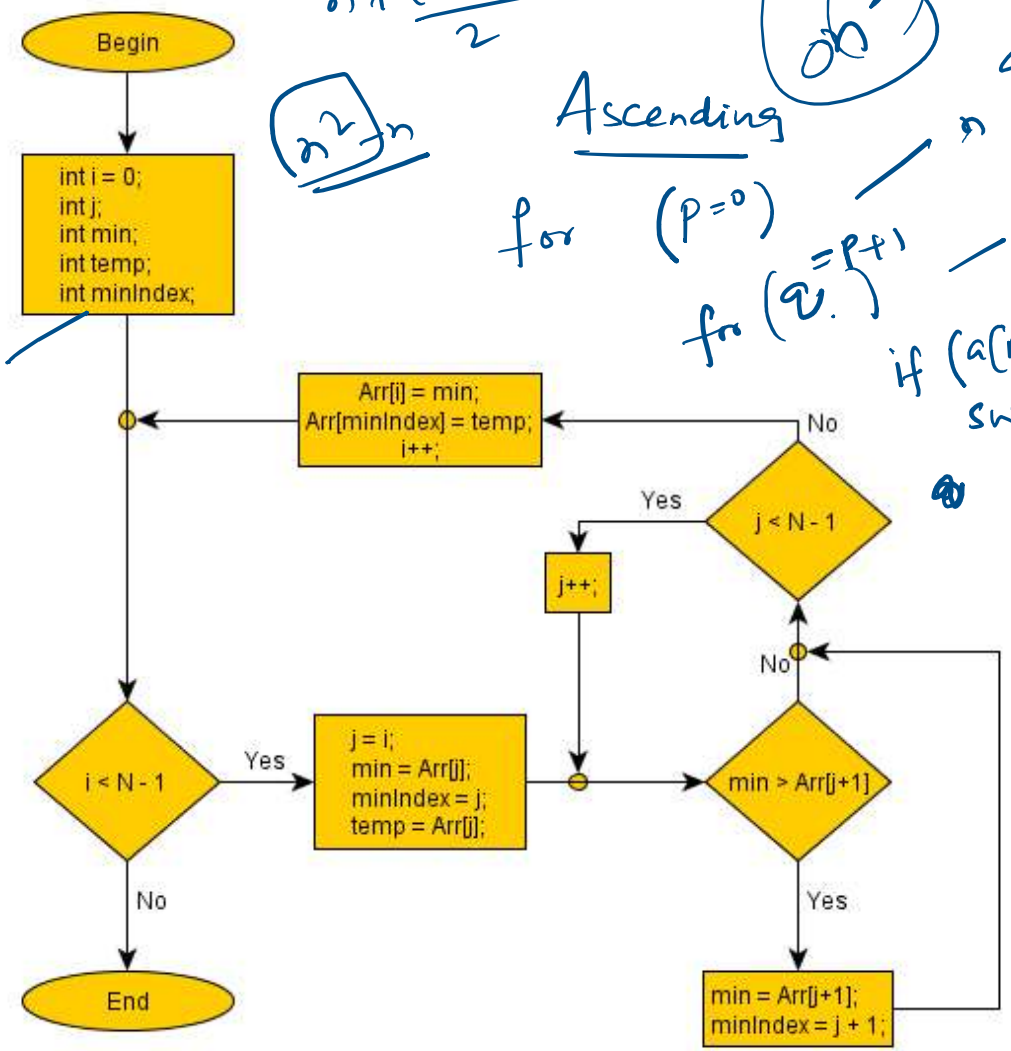


Assignment

- Pseudocode and Flowchart for Sorting Algorithm - Write pseudocode and create a flowchart for a bubble sort algorithm. Provide a brief explanation of how the algorithm works and a simple array of integers to demonstrate a dry run of your algorithm.

Bubble Sort → Flowchart [5, 1, 7, 2]

5:00 – 5:30 Meeting with Techademy



$n*\frac{(n-1)}{2} = O(n^2)$

$n^2 - n$

Ascending

for (p=0) — n

for (q=p+1) — m

if (a[p] > a[q]) swap()

a = [6 0 3 1] ⇒ 0 1 3 6

n-1

$n*\frac{(n-1)}{2}$

q = p+1

{ a[p] > a[q] Swap
else noswap

q ≠ q+1

6 0 3 1 → 0 6 3 1 → 0 6 3 1

6 | 3 1 → 0 3 6 1 → 0 1 6 3 → 0 1 3 6