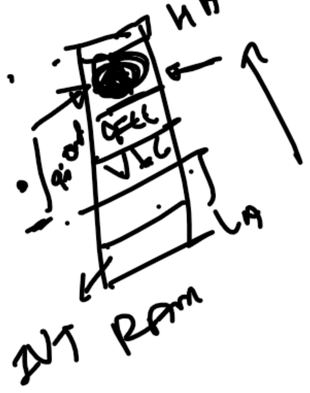
Algorithm:- a sequence of well-defined instructions to solve a well defined problem.

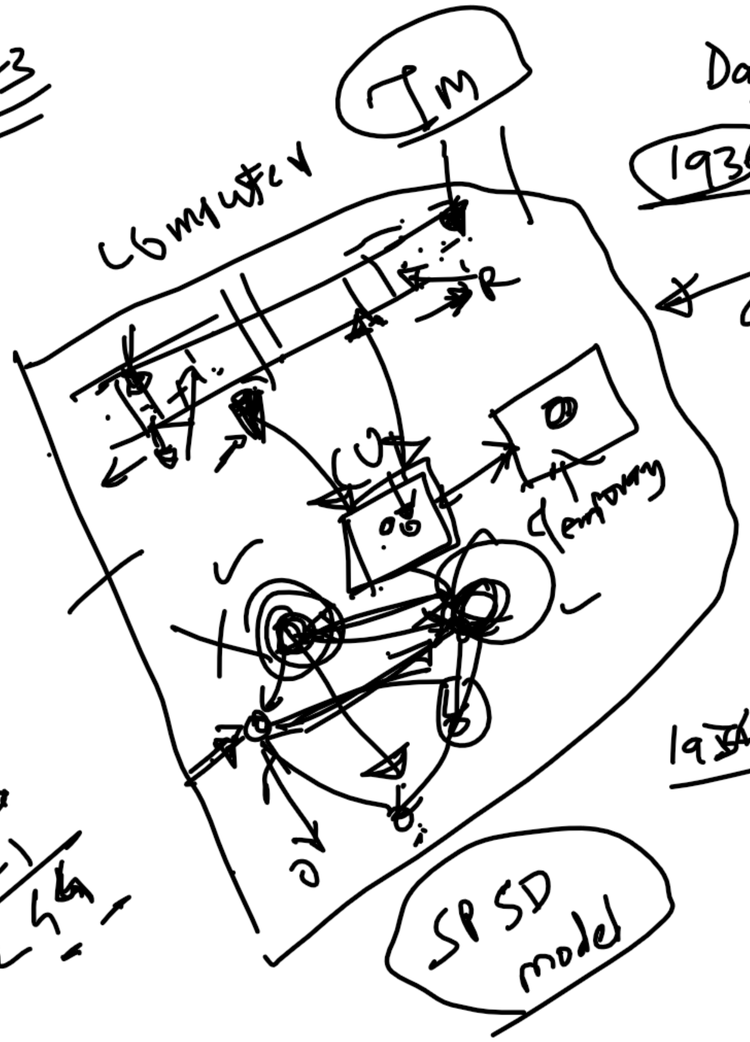
What it is?

What it is not?

Knuth 1973



123
+121
244



Historical Perspective

David Hilbert ← 1900

1936 Alan Turing → Computer

VON-Neumann

George Cantor

∞ ← $\mathbb{N}_0, \mathbb{N}_1$ ← $\mathbb{R}$

Calculus limit

Natural Numbers

1931 Gödel → Make math complete

→ Mechanical

How formal definition is?

Consensus

① Does God exist? Science X

② Do parallel universes exist? X

Genome Sequencing → ATGC

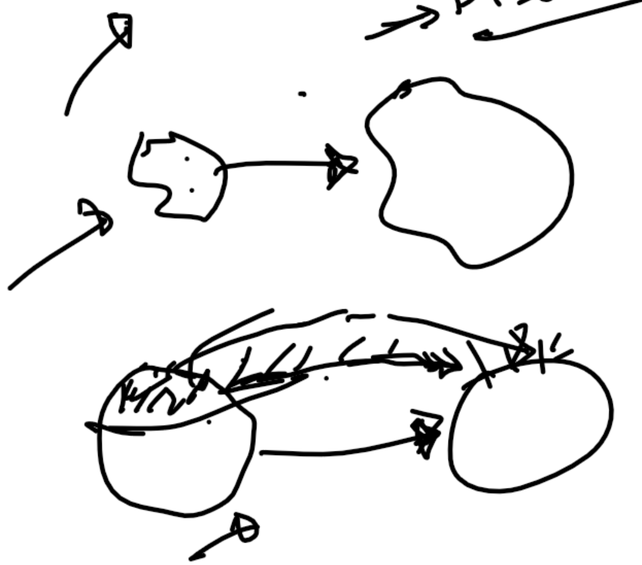
Computational Problem

Loc
FLAT

VLC =

Weather Simulation

CDE → Discretize



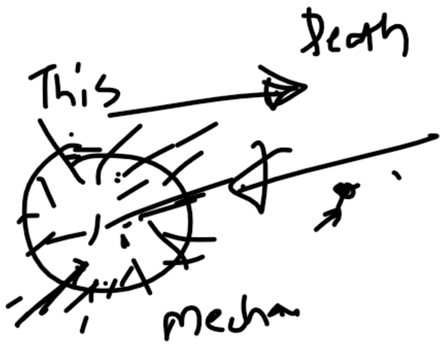
Dreaming!

CT-Scan MRI

Evolution

↓

Algorithm



mechanical Computation

↳

Characteristics of algo:

~~float~~ real r;

r = 5+2;

r = π + e;

Transcend all efforts

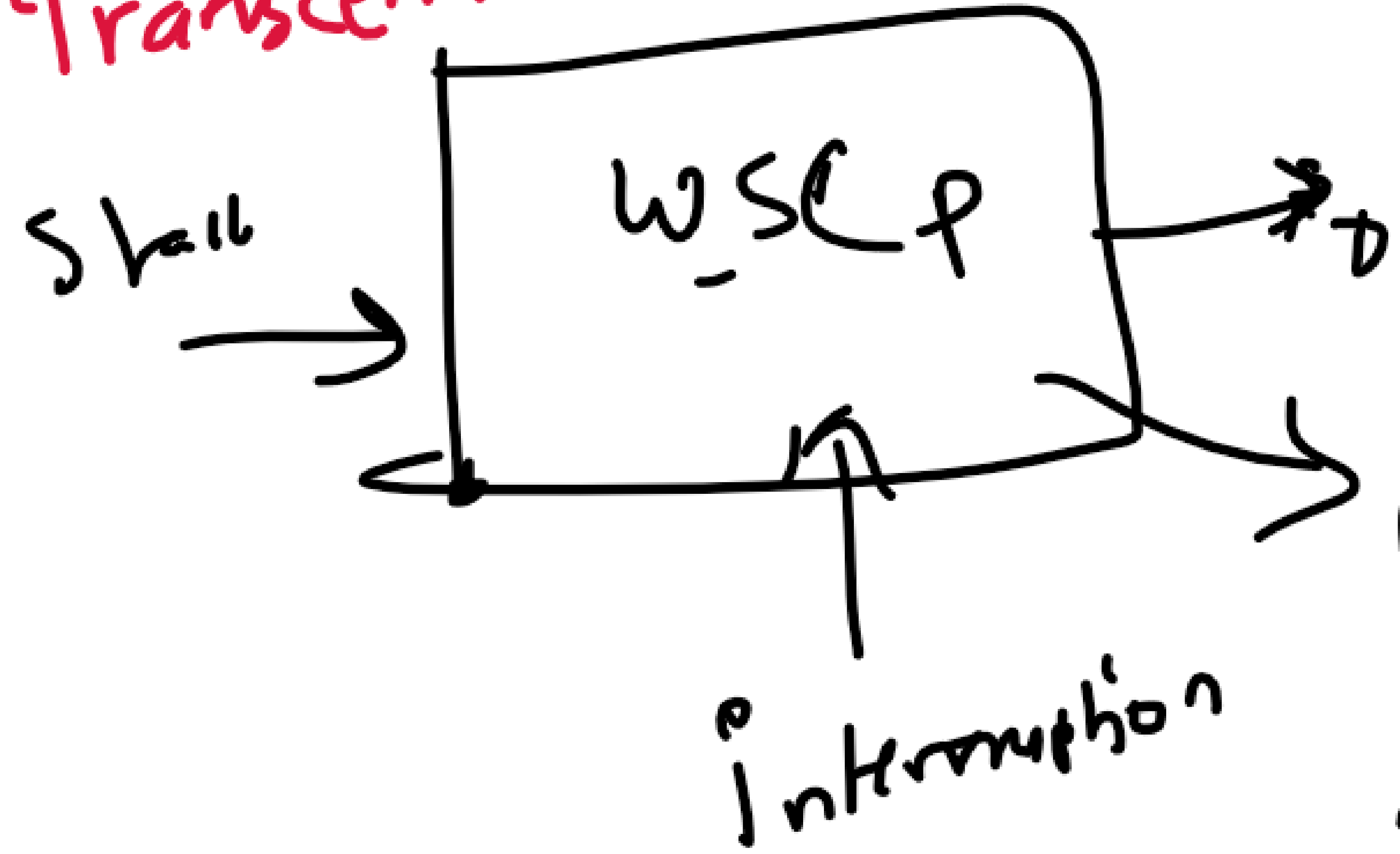
Weather simulation

10 billion year

100 yrs

1. I/P = ≥ 0 > 0
2. O/P ⇒ ≥ 0 > 0
3. No unambiguous / well-defined / definiteness
4. Finite - Time & Space ← Description — Sequence

   should be finite
5. Correctness
6. Effectiveness — feasible
7. elegance — Kolmogorov complexity



Not an algo

int a

a = 5+2×4.

(5+2)×4

5+(2×4)

```
while(1)
{
    a=a+1;
}
```

procedure

Can Infinite Time

Wallis Product for π.

Taylor expansion of e.

∅

OS - procedure / Algo. Y2K

T ≥ O.g T

Single Instruction

Add 5 to 2

or

multiply 3 to 2

(Quantitative) Analysis of Algorithm : Time and Space.

Mathematical Computer → Pseudocode ✗ English

Logic/maths

C/C++

QA of a program

- Executable
- OS
- Language
- Program
- Load of system

if → 3 7 7

Add x,y

Basic instructions

Add, multiply, Assignment, Condition check →

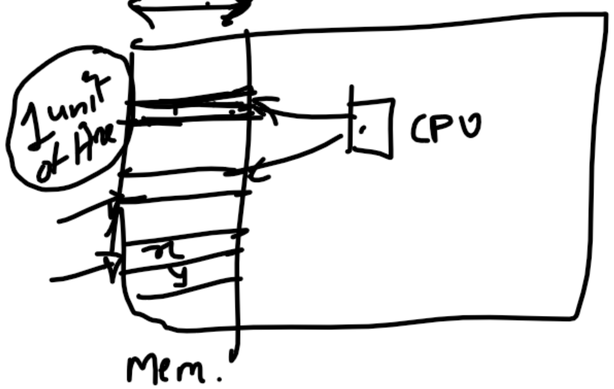
Sort ?? ✗ Not basic

32, 64, 128, ~~256~~.

Sort A, n, w n ∈ ℕ

memory

w ∈ ℕ



Theoretical Machine

w = infinite

1 unit of time

p × q arr -

p, q, r = infinite

∞ + ∞ + ∞ = ∞

10^{10}

2^{10}

Sort A, 2, 32

Sort A, 10^{10}, 32

$2^{32}-1$

32-bit

1 << 32

1 << 128

x^y ✗ x^y ✗ x^y

$x << 3$

$2^{3 \cdot n}$

Calculate 2^n

2^n

$1 \le n \le w-1$

$1 << n$

2^n

$1 << w$

Swap(x,y);

```
temp = x;
x = y;
y = temp
```

```
x = x+y
y = x-y
x = x-y
```

bitwise XOR

```
x = x ^ y;
y = x ^ y
x = x ^ y;
```

$I = \{ i_1, i_2, \ldots, i_k \}$

Assumption — Uniform-cost model

all basic instructions require same units of time = 1 unit

```
for(i=1 to n)
    x = x+1;
```

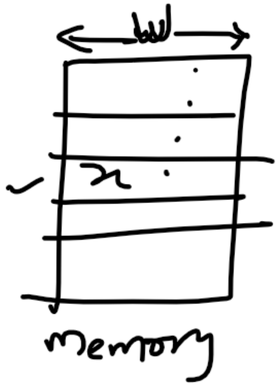
A1

$I = \{for, =, +, \leq, ++\}$ ✗

$I = \{=, +, \leq, ++\}$ $C_=, C_+, C_\leq, C_{++} > 0$

```
x = x+1;
```

A2

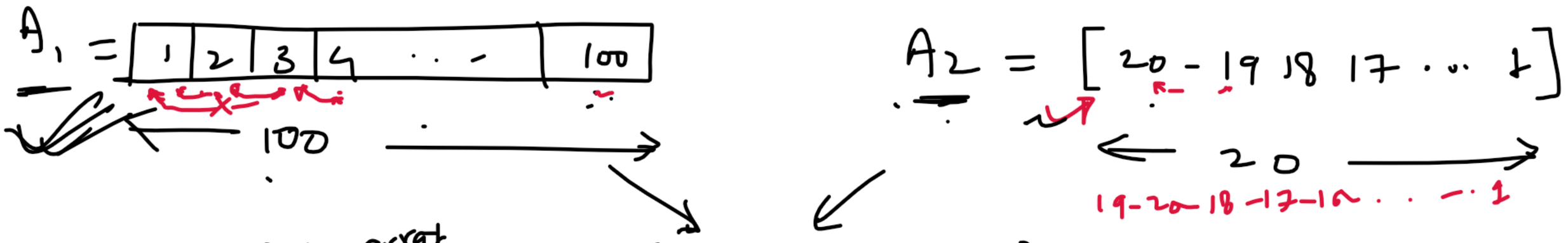


$size(x) > w$ ✗

$\frac{size(x)}{w} = m$

~~data~~ input

individual data unit is stored in a single word & that word is accessible in 1 unit of time.


A1 = 1 2 3 4 . . . 100
100
A2 = [20 19 18 17 . . . 1]
20
19-20-18-17-16 . . . 1
InsertionSort(A, n)
within array elements
# of comparisons : (A1)
= 99
within array elements
# of comparisons :- A2
1 i i+1 i+2 n
L Sorted
i+1
A[i+1] ≤ A[1] ≤ A[2] ≤ A[3] : < A[i]
(A[i], A[j])
|A1| > |A2|
No. of Comparison
99 < 210
# of Comparisons
= 20(21)/2 = 210

$A = 6-5-4-3-2-1$

(6)

$5-6-4-3-2-1$

$4-5-6-3-2-1$

$(A[i], A[j])$

$A[i] > A[j]$

$i < j$

Comp $1+2+3+4+5 = \frac{5\times6}{2}$ $\frac{n(n+1)}{2}$

Shif $1+2+3+4+5 \leq \frac{5\times6}{2}$ $\frac{n(n+1)}{2}$

$S = S_1 = 1+2+3+\cdots+k$

$S = S_2 = k+k-1+k-2+\cdots+1$

$S_1+S_2 = k+1+k+1+k+1+\cdots+k+1$

$= 2S$

$S = \frac{k(k+1)}{2}$

0

1 2 3 4

1 2 4 3 $\Rightarrow$ 1 2 3 4

1 4 2 3 $(4,2)^{\vee}$ $(4,3)^{\vee}$

4 3 2 1 $\frac{4\times3}{2}$

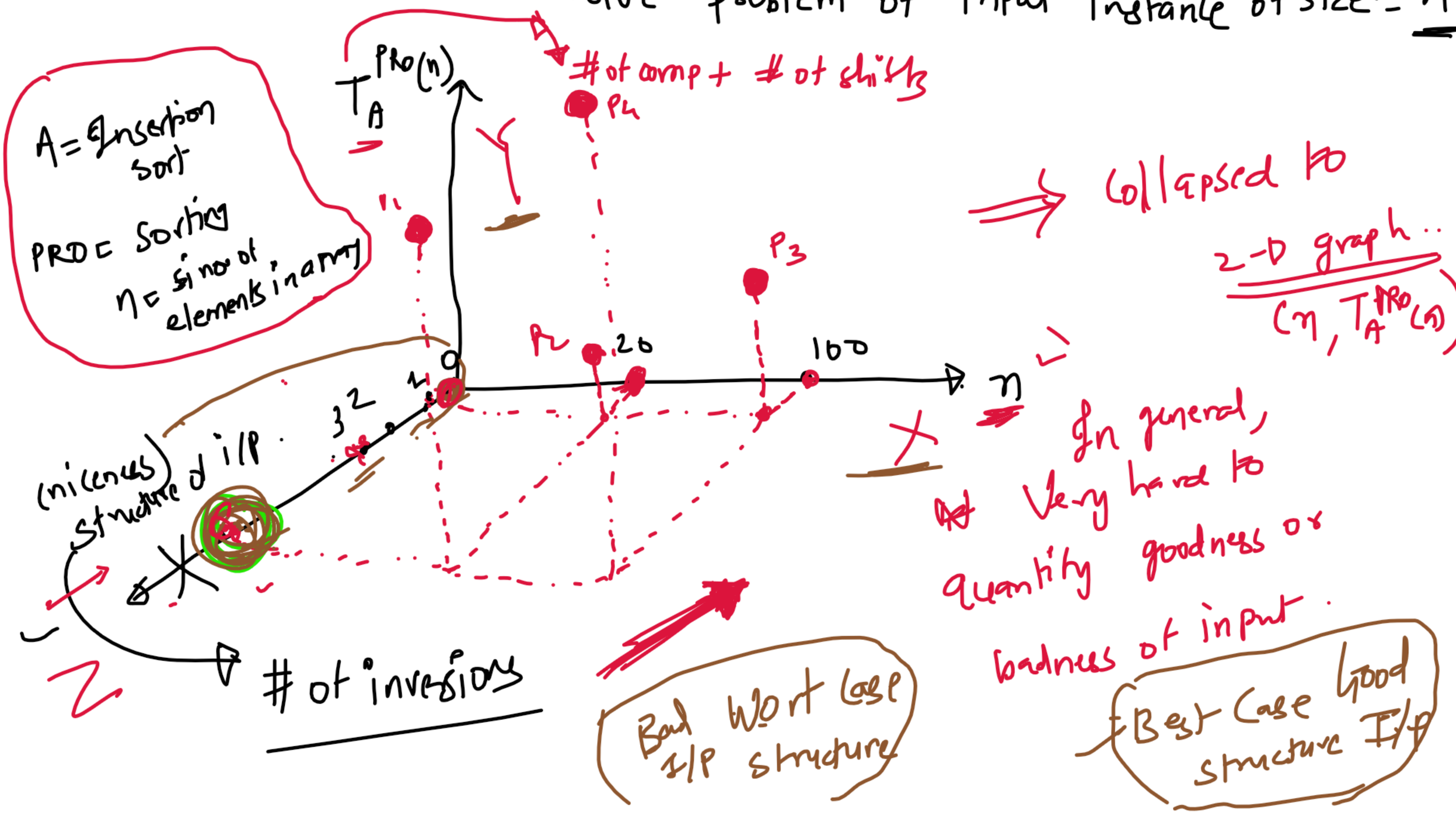
(4,3) (4,2) (4,1) (3,2) (3,1) (2,1)

$T_A^{PRO}(n)$ = ~~Tim~~ Time/steps required for algo A to solve problem of Input instance of size = n

A = Insertion sort

PRO = Sorting

n = Size of elements in array



→ Collapsed to 2-D graph.. $(n, T_A^{PRO}(n))$

In general, Very hard to quantify goodness or badness of input.

A [12 - 11 - 10 - 9] = $\frac{4 \times 3}{2}$ = 6

Inversion: (12,11) (12,10) (12,9) ✓
(11,10) (11,9) ✓
(10,9) ✓

$|A| = n$

$1 \to 2$
$2 \to 1$

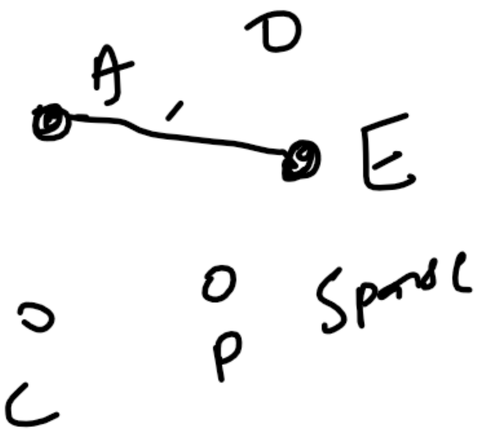
$0 \leq \#\text{Inversions} \leq \frac{n(n-1)}{2}$

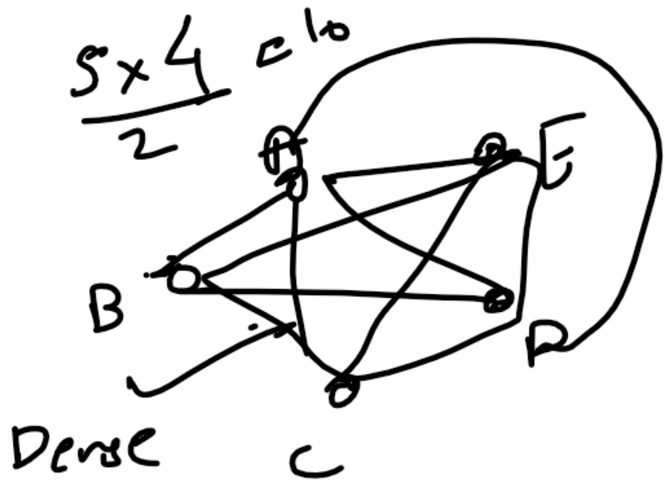
Shortest path problem:

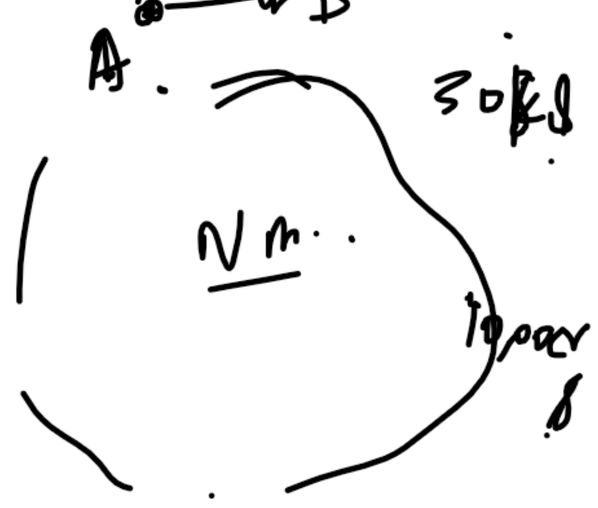
|V|, |E|, weighted edge

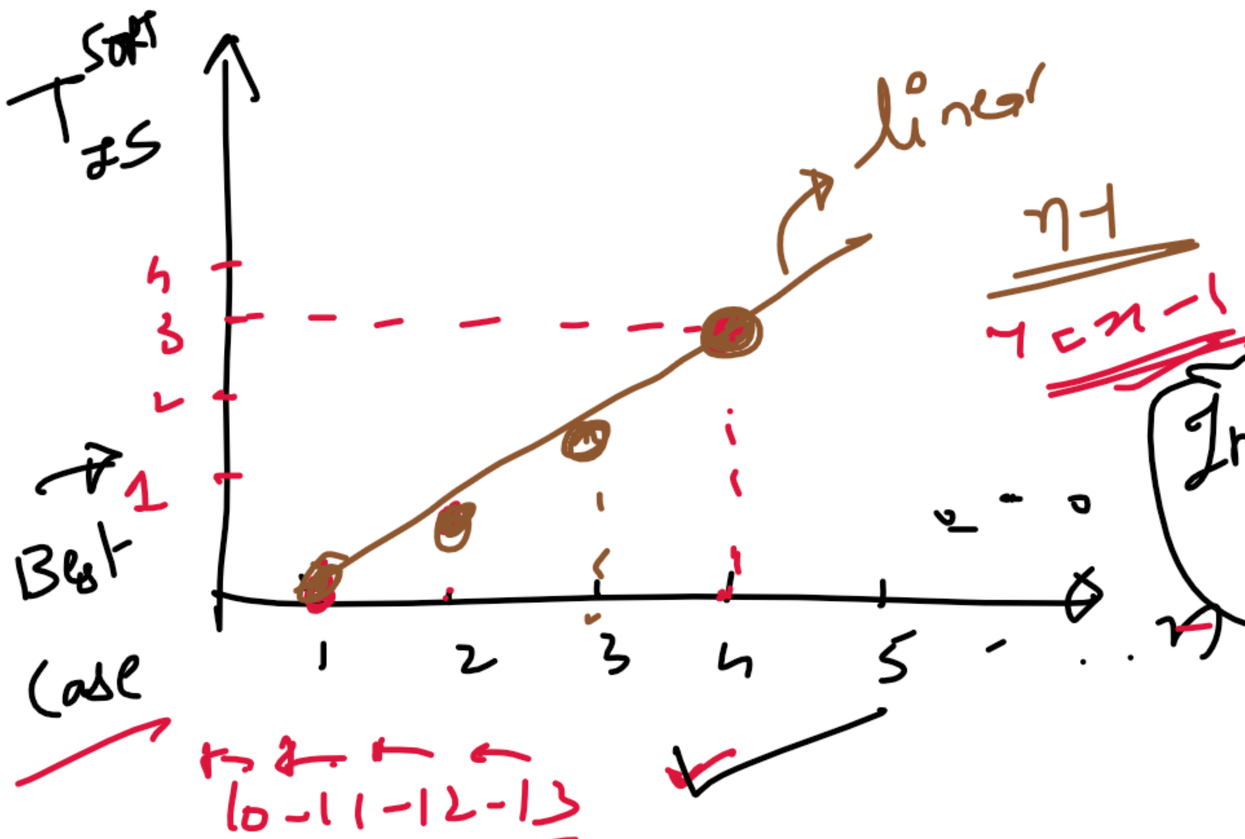
decides input size

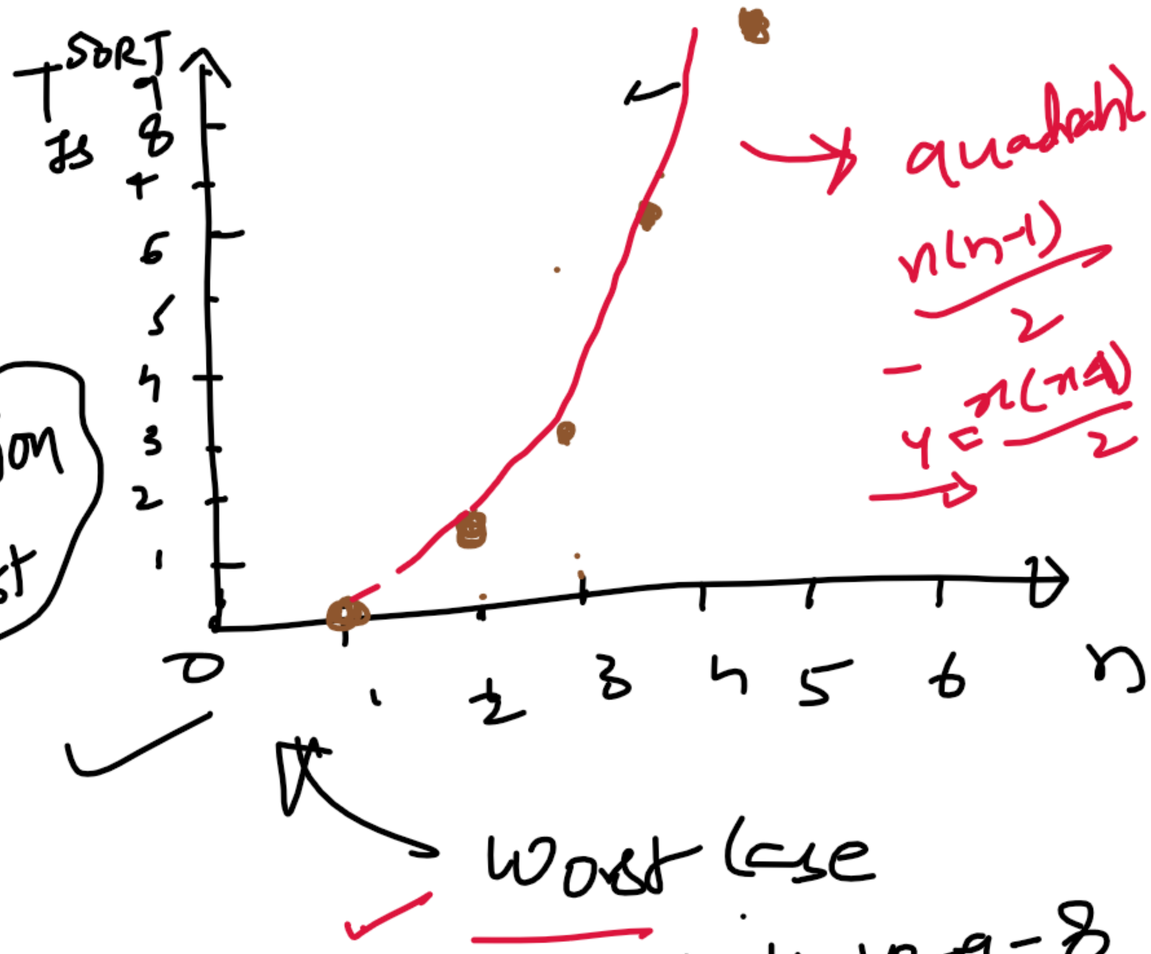
Good graph?
Bad graph?











$\frac{5 \times 4}{2}$ etc

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

Algo ← $I = \{I_1, I_2, \ldots I_{1000}\}$

I/P instance set

$T_A(I_1)$ $T_A(I_2)$ ... $T_A(I_{1000})$

$T_A(I_1) = T_A(I_{19}) = T_A(I_{100})$

$\equiv$ 5 steps

$T_{Avg}^{A} = \frac{\sum_{i=1}^{1000} T_A(I_i)}{1000 = \text{Total no. of instances}}$ → Arithmatic Average

Asymptotic Analysis:-

$\lim_{n\to\infty} T_A(n)$ → Time requirement fⁿ.

$n \to \infty$ → size of input instance

$\lim_{n\to\infty} S_A(n)$ → Algorithm → Space requirement fⁿ

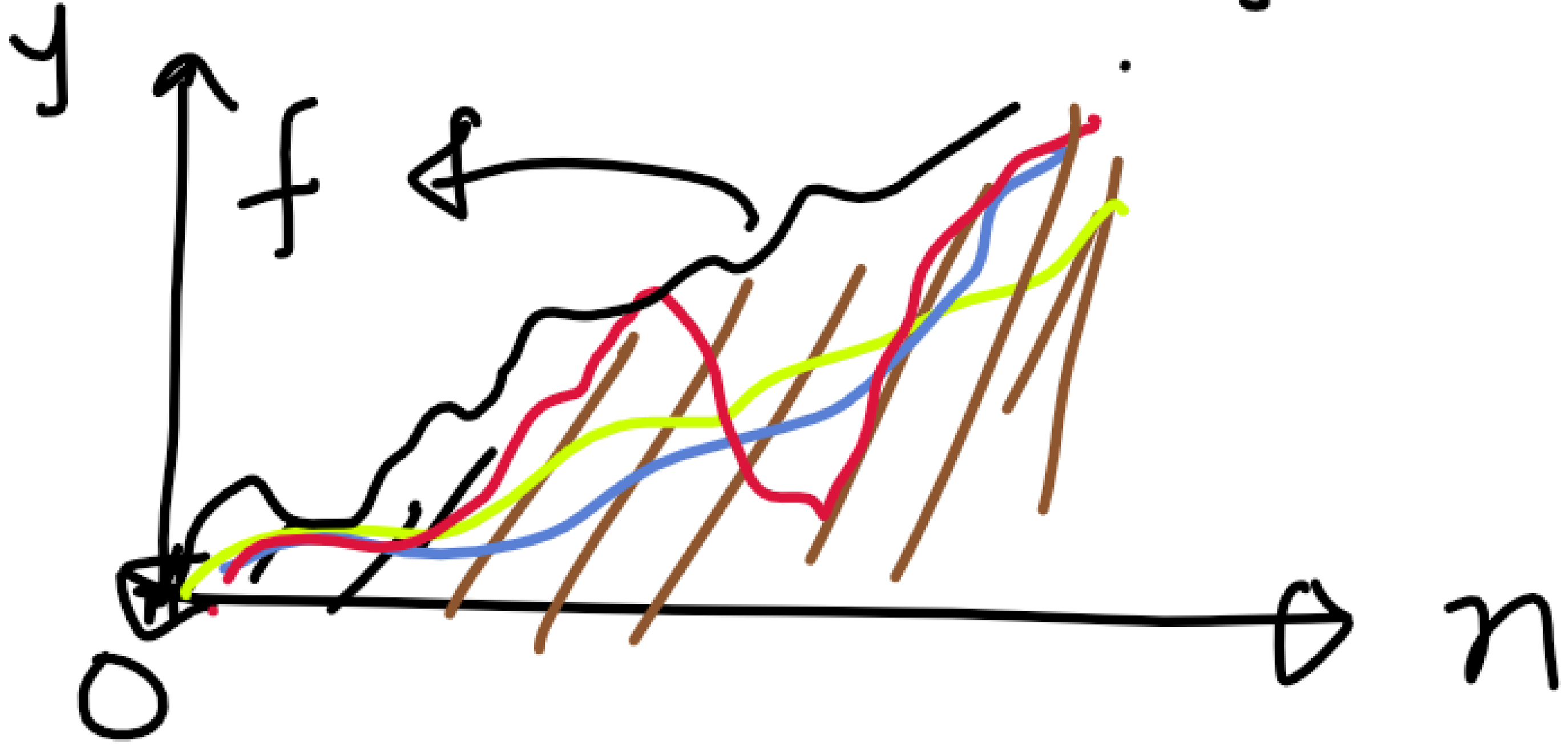
Asymptotic Operators (notations)

① O Big-Oh ② Ω Big-Omega ③ Θ Theta ④ o Small-oh ⑤ ω → Small Omega.

$f : \mathbb{N} \to \mathbb{N}$ $g : \mathbb{N} \to \mathbb{N}$ ① Big-Oh O (19/3) NOP.

∃ → There exists; ∀ → For all; x_0 → x-naught / cut-off point

$$O(f) = \{ g : \mathbb{N} \to \mathbb{N} \mid \exists x_0 \; \forall x \geq x_0 \; g(x) \leq f(x) \}$$



~~f/g = c~~ $f/h = c$

fig ① $n_0 = 1$

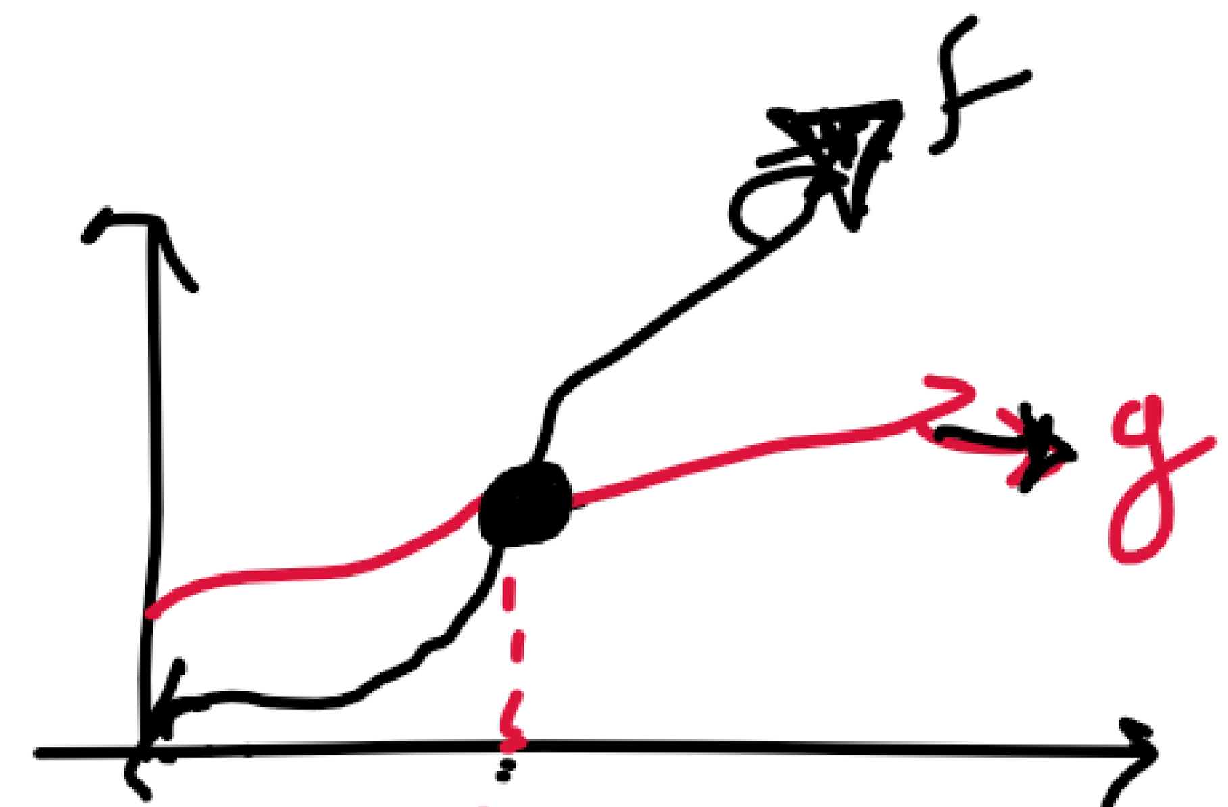


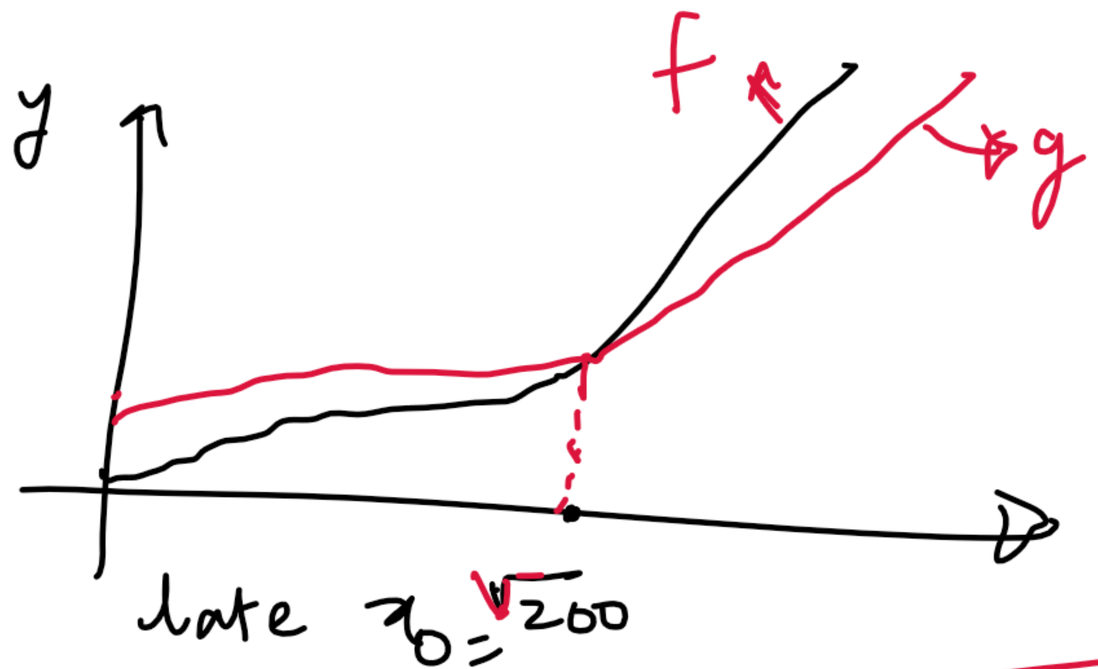
fig ② n_0 → Cut-off Threshold, n_0 // x_0

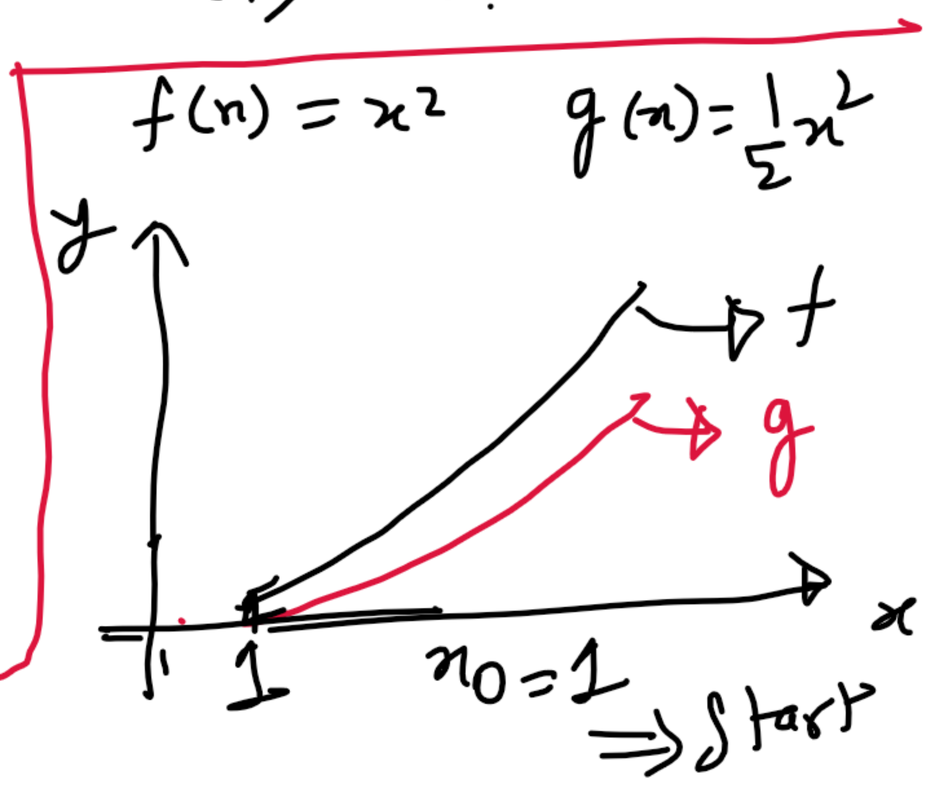
$$O(f) = \{ g : \mathbb{N} \to \mathbb{N} \mid \exists c > 0 \; \exists x_0 \; \forall x \geq x_0 \; g(x) \leq c f(x) \}$$

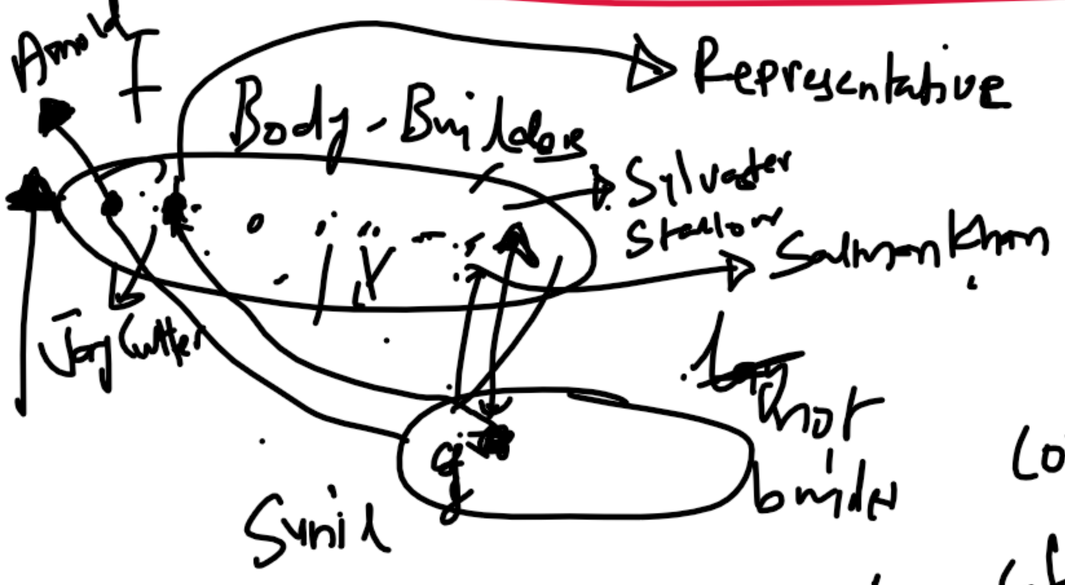
$f(n) = n^2$ $\quad g(n) = \frac{1}{2}n^2 + 100$ $\quad g(n) \leq f(n)$

$n \geq 1$

$\frac{1}{2}n^2 + 100 \leq n^2$ $\quad n^2 \geq 200$ $\quad n \geq \sqrt{200}$.







Relative of f's

Compare

Order Powers of f's :-

$\frac{f(n) = n^2}{g(n) = n^{1.5}} = \sqrt{n}$

R_1

$\frac{f(n) = n^2}{g(n) = 3n^2} = \frac{1}{3}$

R_2

← n is a variable →

if ratio (f/g) is variable one of them is powerful than other the other

if ratio (f/g) is constant then they are ~~equally~~ equivalent in Power.

$f(n) = n^2$ $\quad g(n) = \frac{1}{2}n^2 + 100$ $\quad g(n) \le f(n)$

$\frac{1}{2}n^2 + 100 \le n^2 \quad n \ge \sqrt{200}$

$g(n) \le c \cdot f(n)$

$c = 1 \quad \forall_{n \ge \sqrt{200}} \; g(n) \le 1 \cdot f(n)$

$\nexists \; g(n) \le c \cdot f(n)$ $\quad \frac{1}{2}n^2 + 100 \le c \cdot n^2 \quad (2c-1)n^2 \ge 200$

$\therefore n \ge \sqrt{\frac{200}{2c-1}}$ $\quad c = 1 \Rightarrow n \ge \sqrt{200}$

$n_0 \ge \sqrt{\frac{200}{2c-1}}$ $\quad \bullet \; n \ge 1$ $\quad \therefore 1 \ge \sqrt{\frac{200}{2c-1}} \quad \therefore 1 \ge \frac{200}{2c-1}$

$\therefore g(n) \le \boxed{100.5} f(n) \quad n \ge 1$ $\quad \therefore c = \frac{201}{2} = 100.5$

c ✓

$g(n) = \frac{1}{2}n^2 + 100 \quad f(n) = n^2$

$c \cdot f(n) \not\ge g(n)$ $\quad\longleftarrow\quad$ $cf(n) \ge g(n)$

$c = 1 \quad\longrightarrow\quad c = 100.5$

$n_0 \not\ge \sqrt{200} \quad\longleftarrow\quad n_0 = 1$

$O(f) = \{ g : \mathbb{N} \to \mathbb{N} \mid \exists c > 0 \; \exists n_0 \in \mathbb{N} \; \forall n \ge n_0 \; g(n) \le c \cdot f(n) \}$

$\exists \exists \quad \exists \forall \quad \forall \exists$

$g_1 \in O(f) \Rightarrow c_{g_1}$

$g_2 \in O(f) \Rightarrow c_{g_2}$

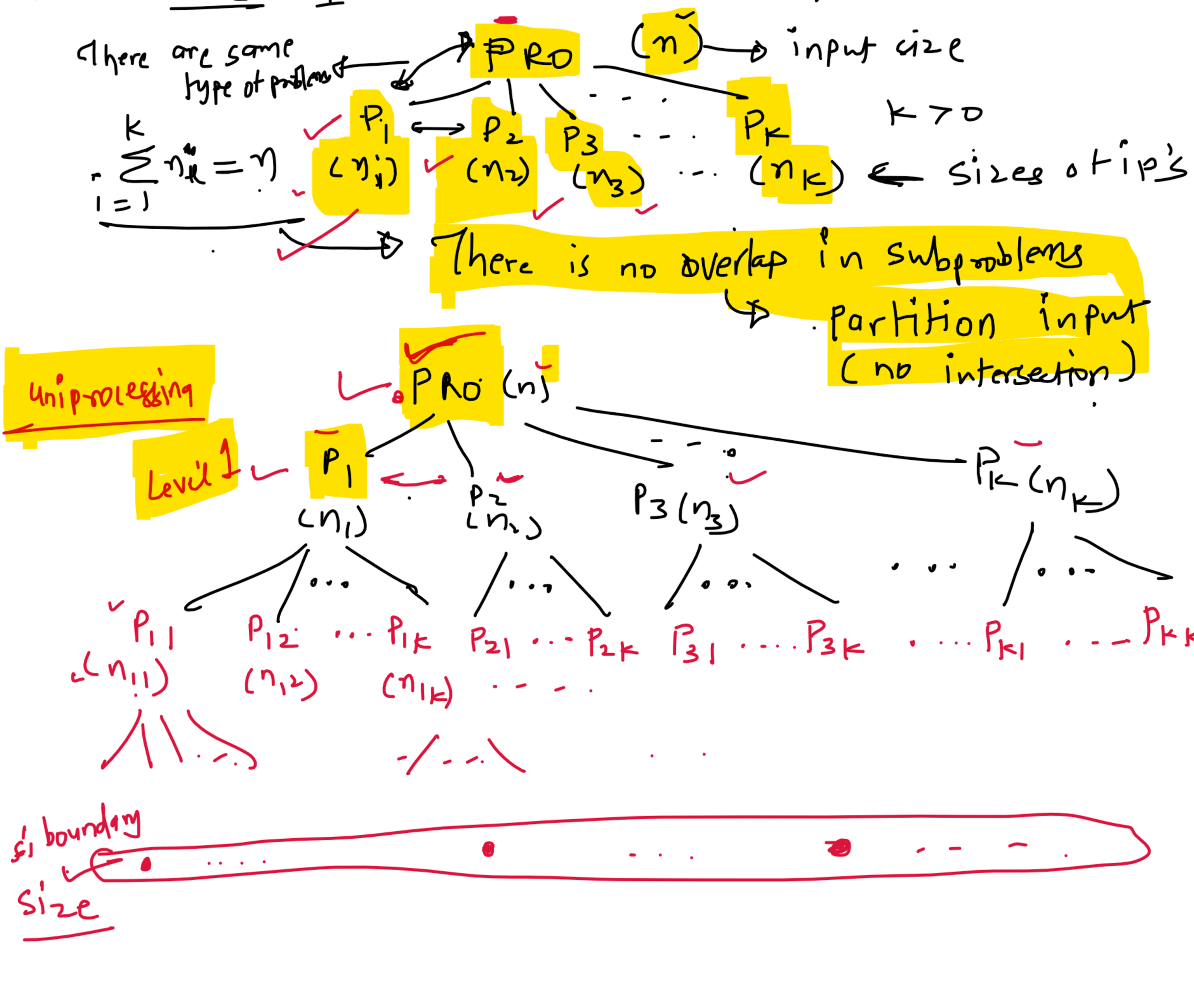
$O(f) = \{ g : \mathbb{N} \to \mathbb{N} \mid \exists n_0 \in \mathbb{N} \; \exists c > 0 \; \forall n \ge n_0 \; g(n) \le c \cdot f(n) \}$

if $\frac{1}{2}n^2 + 100 \in O(n^2)$ $\Rightarrow$ $(c = 1, n_0 = \sqrt{200})$ OR $(c = 100.5, n_0 = 1)$

True

Unit 2 :- D & C (momentarily)

Pro :- Size n of the input $n \geq 0$



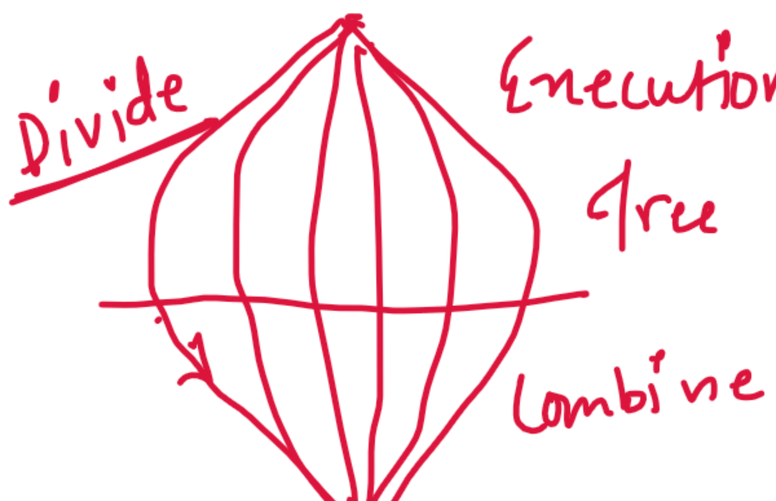
Problem: Given a random array A of size n, simultaneously find min of A & max of A. $n \in \mathbb{N}$

Straightforward :- No D&C approach: **Iterative approach**

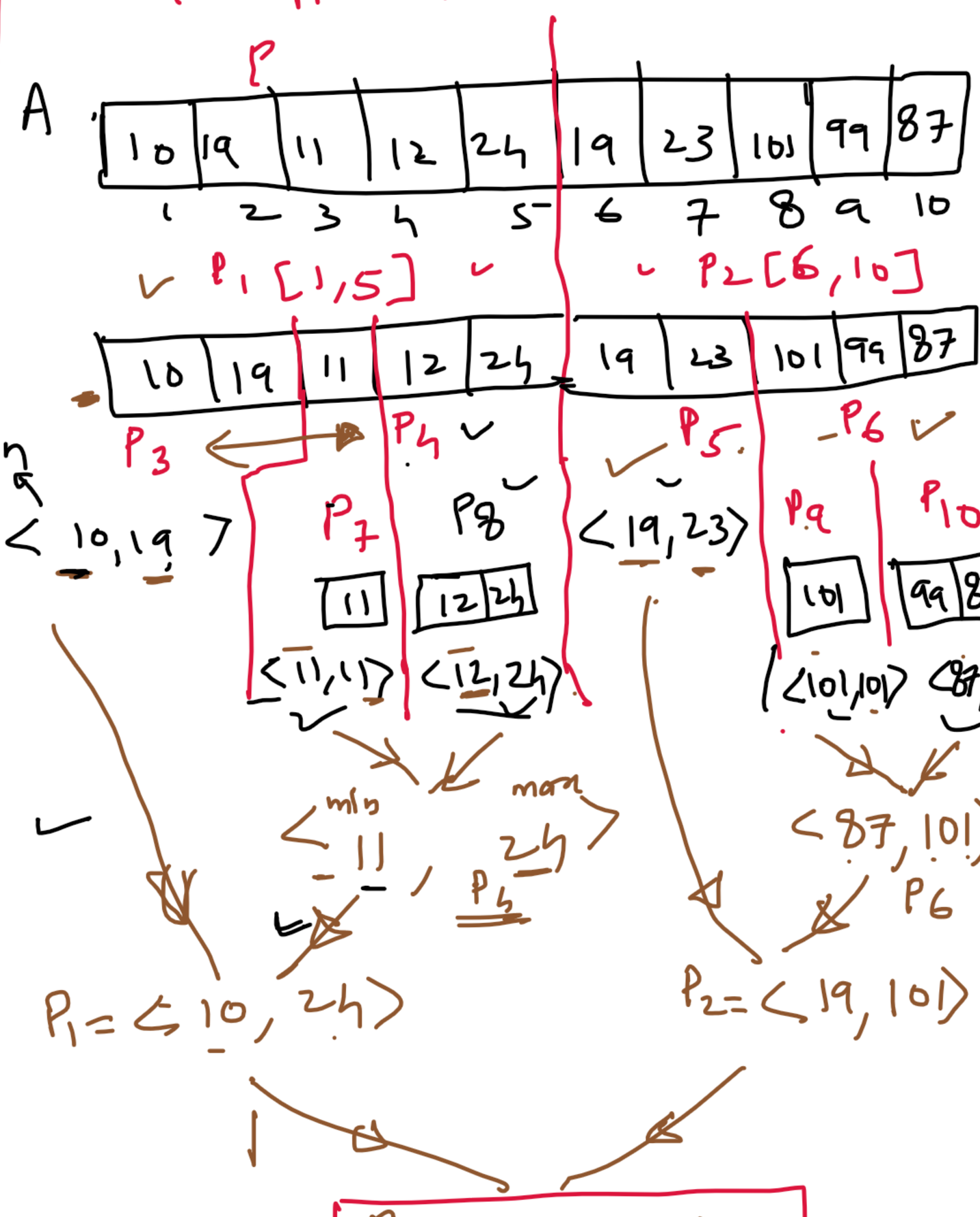
```
Algostraight(A, n)
{ min = A[1]; max = A[1];
  for(i = 2 to n)
  {
    if(A[i] < min)
      min = A[i];
    if(A[i] > max)
      max = A[i]
  }
}
```

No. of Comparisons :-

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



D&C approach.



Algo :

< min , max > DCminmax (A, low, high) → Array indices

{

if (low == high)

~ return < A[low], A[low] >;

else

if (high - low == 1)

{ if (A[low] < A[high])

~ return < A[low], A[high] >;

else

~ return < A[high], A[low] >;

}

~ mid = $\frac{low+high}{2}$; → Divide phase

<minL, maxL> = DCminmax (A, low, mid);

<minR, maxR> = DCminmax (A, mid+1, high);

if (minL < minR)

MIN = minL;

else

MIN = minR;

if (maxL > maxR)

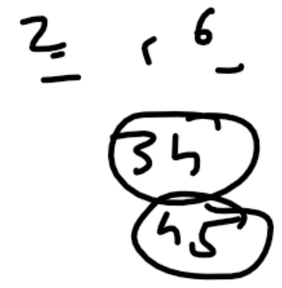
MAX = maxL;

else

MAX = maxR;

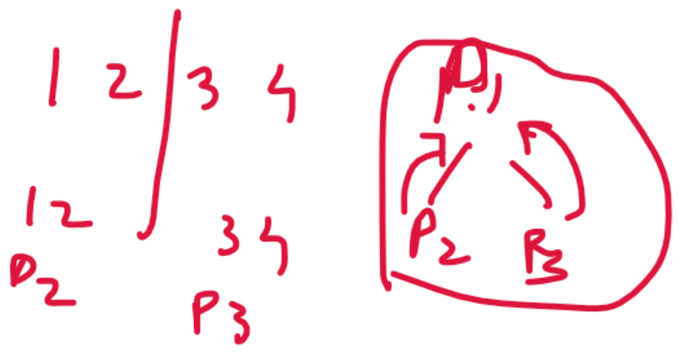
return < MIN, MAX >;

}



Stack Depth

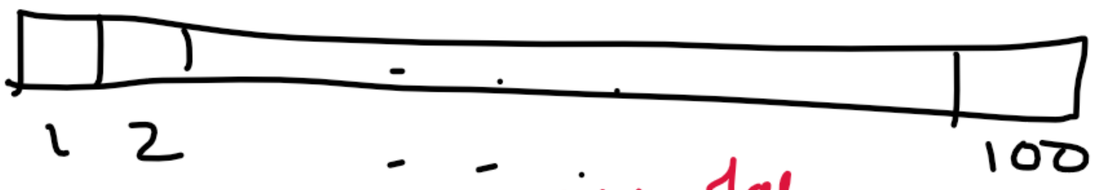
boundary condition



Solving subproblems

Algo statement → X → < min(minL, minR), max(maxL, maxR) >

Combine Operations

A = 

Solving

$x_1 + x_2 + \dots + x_n = r$

$r + (n-1) - n$

C_{n-1}

→ degree of division :- 4

99_{C_3} ways to distribute I/p to 4 sub problems

Boundary condition size ≤ 6

Division Tree

P_1 [1 → 100]

P_2 [1-25] P_3 [26-50] P_4 [51-75] P_5 [76-100]

P_6 [1-6] P_7 [7-12] P_8 [13-18] P_9 [19-25]

P_{10} P_{11} P_{12} P_{13} [26-50]

P_{14} P_{15} P_{16} P_{17} [ ]

P_{18} P_{19} P_{20} P_{21} [ ]

6 6 6 6 6 6 6 6 6

P_{22} P_{23} P_{24} P_{25}

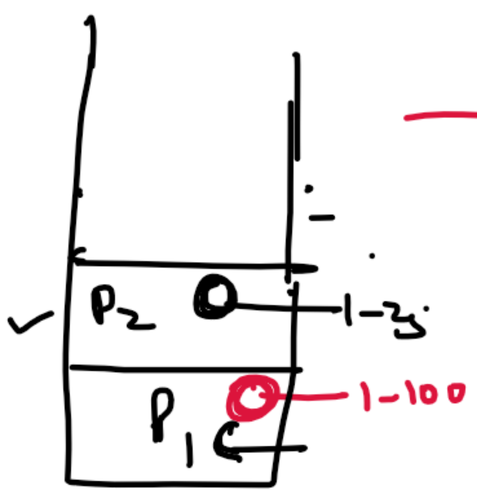
[19-19] [20-20] [21-21] [22-25]

26 27 28 29

30 31 32 33

34 35 36 37

Uniprocessing



→ Every DfC algo implemented recursively as a program/algo

It will always have a execution stack which works the same way

→ Not every DfC algo has to be implemented using recursion

Eg: MergeSort ⇒ Iterative version

Straightforward Method for Simultaneous Min & Max:

Hp: A[1 ... n]

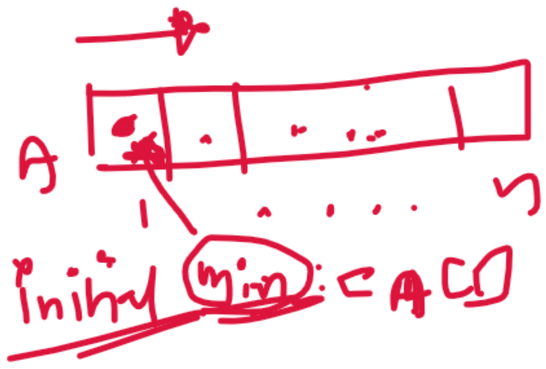
O/P: <min, max> min = MIN{A[1], ..., A[n]}
max = MAX{A[1], ..., A[n]}

(1) No Thought Method
Easiest Method:

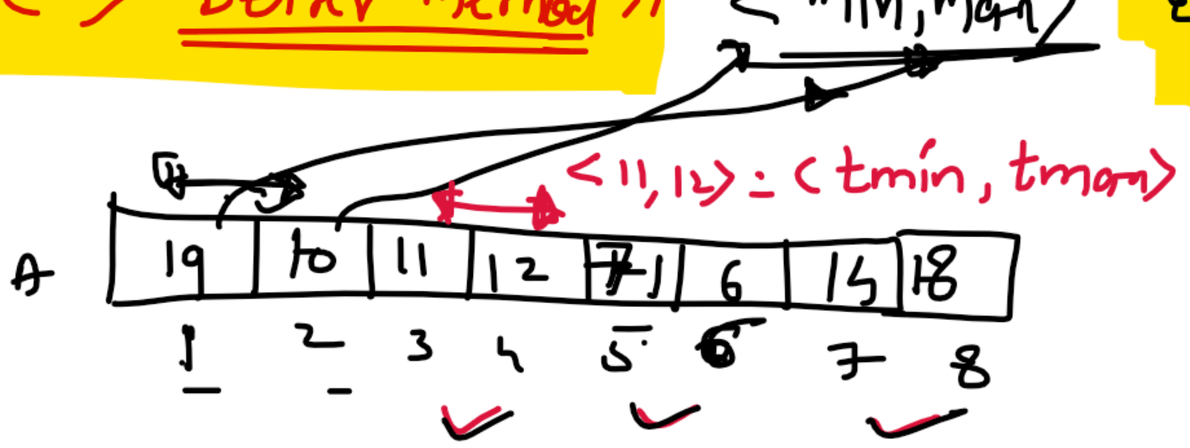
Step 1: Find min using linear search on A[1...n]
Eg: (Left to right)

Step 2: — // max — //

# of Comparisons: (within array elements) = n-1 + n-1 = 2n-2



(2) Thought Better method:



Even length

gmin = 6, gmax = 7

	pmin	pmax	Pair of indices	# of comparisons
(1)	11	12	(3,4)	1+2=3
(2)	6	71	(5,6)	1+2=3
(3)	14	18	(7,8)	1+2=3
			+ initial = 1	
			End	10

|A| = n is even

# of comparisons = $1 + 3\cdot\left(\frac{n-2}{2}\right) = \frac{3n-4}{2}$

n = odd

|A| = n is odd

$= 3\left(\frac{n-1}{2}\right) = \frac{3n-3}{2}$

$\left\lfloor \frac{3n}{2} \right\rfloor$

gmin = 11, gmax = 43, Initial = 0

	Pmin	Pmax	Pair of Indices	# of comp
(1)	11	21	(2,3)	1+2=3
(2)	12	33	(4,5)	1+2=3
(3)	25	43	(6,7)	1+2=3
(4)	23	39	(8,9)	1+2=3
				= 12

1) Easiest Method Algo A1

Array index range is 1 to n

```
A1 (A, n)
{
1   min = A[1];
2   for (i = 2 to n)
        if (A[i] < min)
            min = A[i];
3   max = A[1];
4   for (i = 2 to n)
        if (A[i] > max)
            max = A[i]
}
```

2) Better Method Algo A2

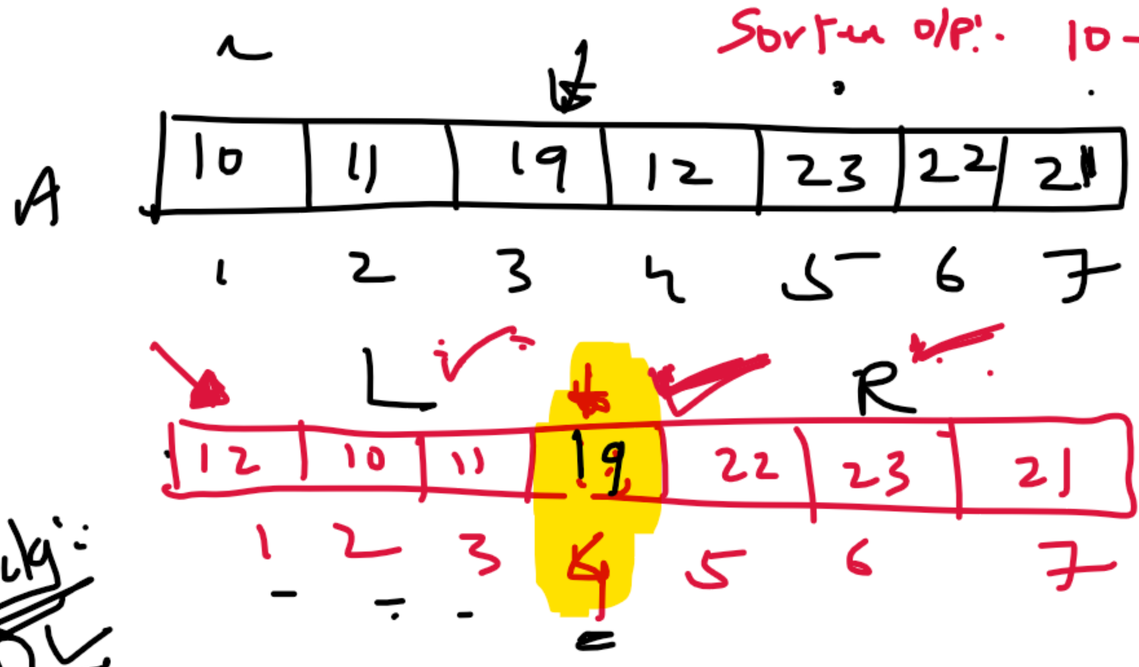
```
A2 (A, n)
{
1   if (n is odd)                    // if (n%2 == 1)
        <min, max> = <A[1], A[1]>    // i = 2
    else                             // i = 3
        if (A[1] ≤ A[2])
            <min, max> = <A[1], A[2]>
        else
            <min, max> = <A[2], A[1]>
2   while (there are pairs)          // (i ≤ n-1)
    {
        • Pick next pair <A[i], A[i+1]>
        • if (A[i] ≤ A[i+1])
              <pmin, pmax> = <A[i], A[i+1]>
          else
              <pmin, pmax> = <A[i+1], A[i]>
        • if (pmin < min)
              min = pmin;
        • if (pmax > max)
              max = pmax;
        • i = i + 2.
    }
3   return <min, max>
```

we same check before initial setting

Assumption: Array index starts from 1

Quick Sort :-

Sorted o/p: 10 – 11 – 12 – 19 – 21 – 22 – 23



Check:

① ~~Try~~ to find correct pos^{n} of 19 in final sorted array.

② ✓ make sure that once you find correct pos^{n} of 19, $x < 19$ should not be part of L

③ — " — $y > 19$ — " — part of R.

if ①, ② & ③ are made sure of then combine phase is not required.

(A, n) Merge Sort

$i < j$

① Divide A[i ... j]

② Solve subproblems

A[i ... $\frac{i+j}{2}$] → Left

A[$\frac{i+j}{2}+1$ ... j] → Right

③ Combine:

Merge (i, $\frac{i+j}{2}$, j)

→ Left & Right

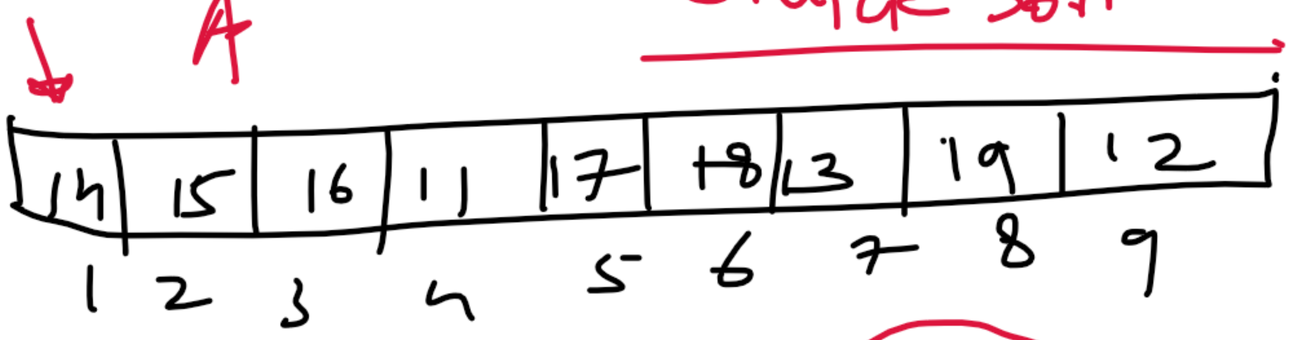
Subarrays interact with each other to get bigger array sorted.

Divide phase - less work

Solve phase :- moderate

Merge :- lot of work

Quick Sort



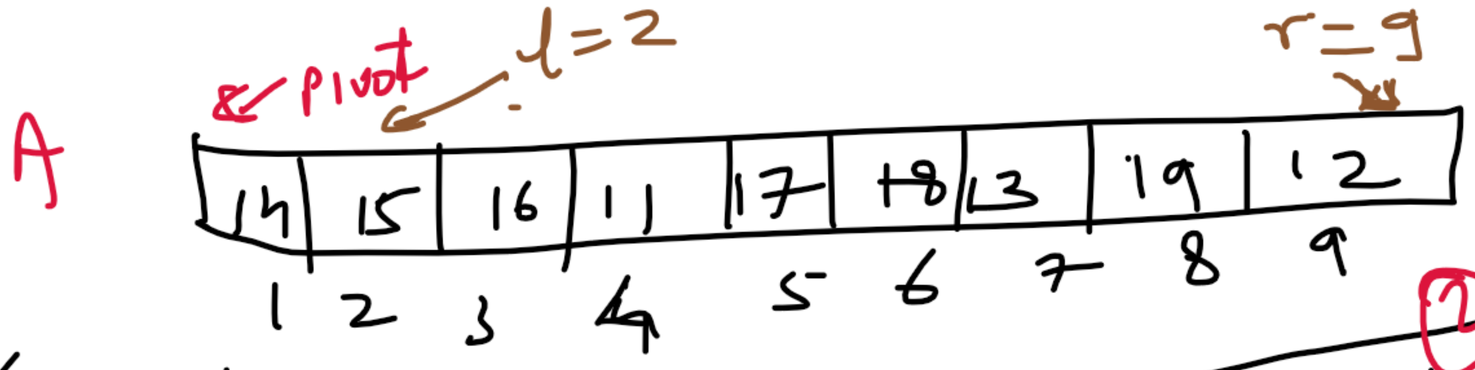
Strategy:
1. Pick up pivot element. [first element of array under consideration] In our case
2. Find correct position of pivot in array. (sorted)
   → It is a process,
3. While performing (2), make sure that all elements
   x > pivot go to right side of pivot
   y < pivot go to left side of pivot.

EA

4. This will create L = left of pivot & array R = right side of pivot.
5. Sort L using steps (1) to (4) [recursively] ... and so with R.

D & C strategy
(1) Divide into subproblems
(2) Solve subproblems
(3) Combine solutions of subproblems to get parent solution

Sedgewick's version of segregation



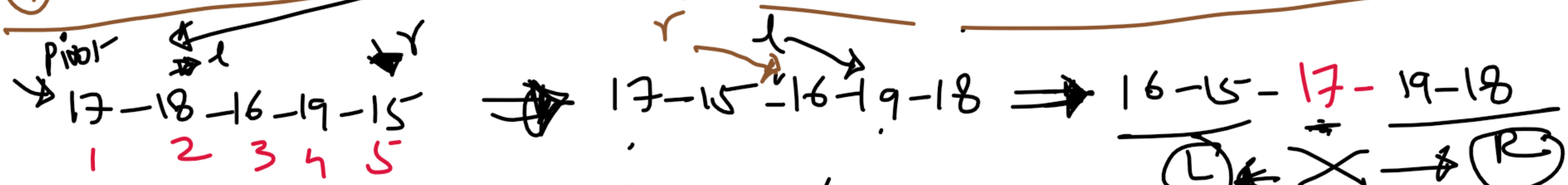
C. A. R. Hoare

(1) Find no. of elements ≤ 14
Position = index 4
→ Swap(A[1], A[4])
11 - 15 - 16 - 14 - 17 - 18 - 13 - 19 - 12

(2) what we want
i's L:
12-11-13-14
18-17-19-16-15 R:
- Sort L
- Sort R

STEPS (Example):

(1) 14 - 12-16-11-17-18-13-19-15
(2) 14-12-13-11-17-18-16-19-15
(3) 11-12-13-14-17-18-16-19-15
(4)



Sedgewick's quick sort

Merge Sort
1. Sort L, Sort R
2. Merge L & R

Quick Sort using Pivot
1. Segregate into L & R
2. Sort L & Sort R

QuickSort Pseudocode:

I/P : A |A| = n

O/P : A'[1] ≤ ... ≤ A'[n]

```
Algo QuickSort(A, low, high)
{
1   if (low < high)
    {
2     p = Partition(A, low, high);
3     QuickSort(A, low, p-1);
4     QuickSort(A, p+1, high);
    }
}
```

p is set to value y from partition procedure

Correct index of pivot element in sorted output.

Ask left to sort recursively

Ask right to sort recursively

Steps (of back page)

1 to 3 ⇒ Divide Phase

4 to 5 ⇒ Solve children subproblems

there is no combine phase.

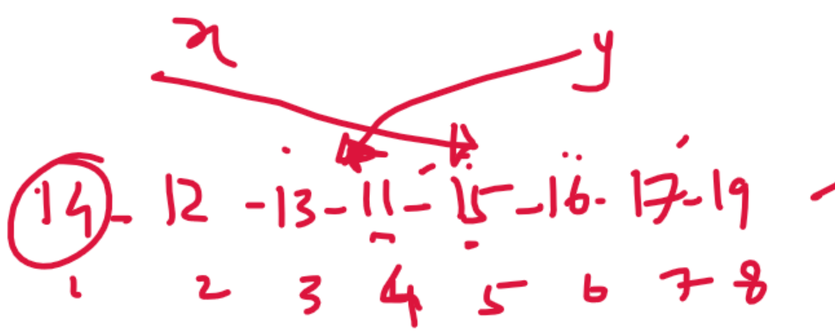
Initial call

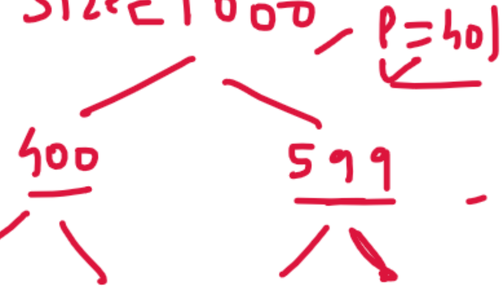
QuickSort(A, 1, n);

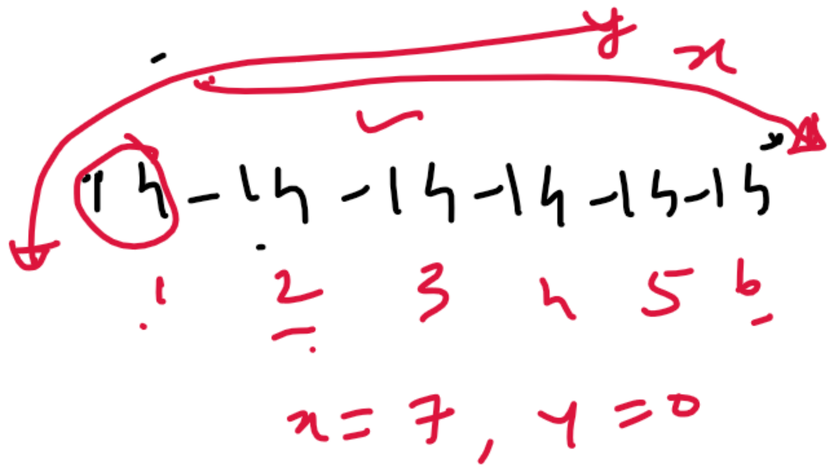
low ≤ p ≤ high ← a fact

```
Algo Partition(A, l, r)
{
1   pivot = A[l];
2   x = l+1; y = r;

3   while (x < y)
    {
4       while (x ≤ r)
5           if (A[x] ≤ pivot)
6               x++;
            else
7               break;

8       while (y ≥ l)
9           if (A[y] ≥ pivot)
10              y--;
            else
11              break;

12      if (x < y)
13          exchange/swap (A[x], A[y])
    }
14  if (x ≤ y)
        exchange(A[y], A[l]);
15  return y;
}
```

A[l] is considered as a pivot for partitioning







exchange(A[x], A[y])

x > y

x < y

l ≤ y ≤ r

x ≥ l

(Unit-1) Asymptotic Notations

Time Complexity Analysis of Algorithms:-

Every basic step takes 1 unit of time for execution.

+, =, -, *, /, ≤,

1)

```
Algo A1()
{
1   x = y + 1
}
```

$T_{A_1}() = 2$

---

2)

```
Algo A2()
{
1   for(i = 1 to n)
2   x = x + 1;
}
```

# of time units spent

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

$(2)n = 2n$

$T_{A_2}(n) = 2n+2 + 2n$

$= 4n+2$

---

3)

```
Algo A3()
{
1   for(i = 1 to n)
2   ----for(j = 1 to n)
3   x = x + 1;
}
```

# of time units Spent

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

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

$((2)n)\,n = 2n^2$

$T_{A_3}(n) = \underline{2n+2} + \underline{2n^2+2n} + \underline{2n^2} = 4n^2+4n+2$

→ Quadratic Equation (polynomial of degree 2)

---

4)

```
Algo A4()
{
1   for(i = 1 to n)
2   ---for(j = 1 to n)
3   ------for(k = 1 to n)
4   x = x + 1;
}
```

time units Spent

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

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

$((1 + (n+1) + n)n)\,n = 2n^3+2n^2$

$(((2)n)n)\,n = 2n^3$

$T_{A_4}(n) = \underline{2n+2} + \underline{2n^2+2n} + \underline{2n^3+2n^2} + \underline{2n^3} = 4n^3+4n^2+4n+2$

→ Cubic equation / polynomial of degree 3

5) Algo $A_5()$

{

1 for (i = 1 to n) ⟹ $1+(n+1)+n = 2n+2$

2 for (j = 1 to i) ⟹ $1+(i+1)+i = 2i+2 = t_{2i}$

3 $x = x+1$ ⟹ $((2)i)n = t_{3i}$

}

time units Spent

← MISTAKE

$\sum_{i=1}^{n} t_{2i}$

$\sum_{i=1}^{n} t_{3i}$ MISTAKE

$T_{A_5}(n) = 2n+2 + \sum_{i=1}^{n}(2i+2) + \sum_{i=1}^{n} 2in = 2n+2 + (n+1)\sum_{i=1}^{n} 2i + 2\sum_{i=1}^{n} 1$

$= 2n+2 + (n+1)\,n(n+1) + 2n =$

---

6 Algo $A_6()$

{

1 for (i = 1 to n) → $1+(n+1)+n = 2n+2$

2 for (j = 1 to a) → $(1+(a+1)+a)n = (2a+2)n$

3 $x = x+1$ → $((2)a)n = 2an$

}

time units

$T_{A_6}(n,a) = 2n+2 + 2an+2n + 2an$

$= 4an + 4n + 2$

Asymptotic Notations : (operators) $O, \Omega, \Theta, o, \omega$

Big-oh (O): f and g are functions $f: \mathbb{N} \to \mathbb{N}$ $g: \mathbb{N} \to \mathbb{N}$

$$O(f) = \{ g \mid \exists_{c>0} \text{ (constant)} \; \exists n_c \; \forall_{n \geq n_c} \; g(n) \leq c \cdot f(n) \}$$

$$O(f) = \{ g \mid \exists_{c>0} \exists n_c \; (\forall_{n \geq n_c}) \; g(n) \leq c \cdot f(n) \}$$

Ex 2 :- $f(n) = n^2$ $g(n) = 2n^2$ ✓

$g \in O(f)$? True

$2n^2 \leq c \cdot n^2$, Put $c \geq 2$, $n_c = 1$ $\quad (c \geq 2, \; n_c = 1)$ → Proof

Ex 2 : $f(n) = n^2$ $g(n) = \frac{1}{100} n^2$ ✓

$g \in O(f)$?

$\frac{1}{100} n^2 \leq c \cdot n^2$ $\quad c \geq \frac{1}{100}$ $\quad (c \geq \frac{1}{100}, \; n_c = 1)$ → Proof

Ex 3 $f(n) = n^2$ $g(n) = n^{2.1}$

$g \in O(f)$?

False

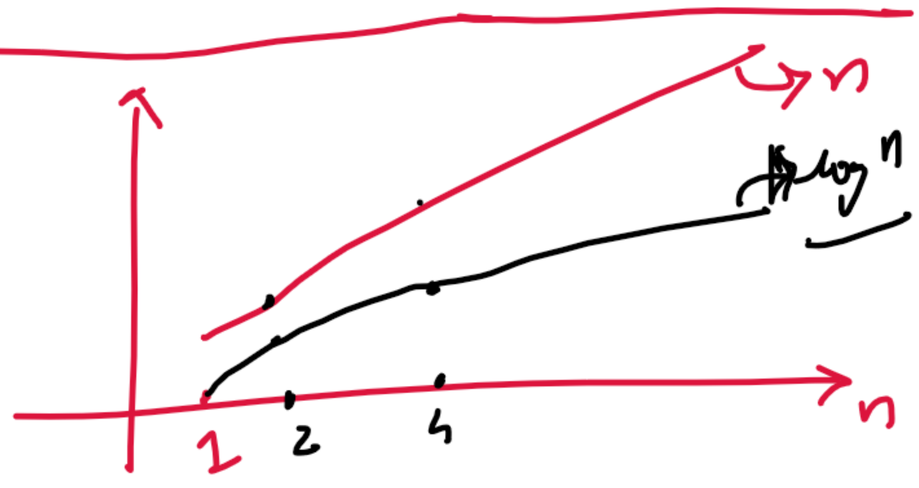
$n^{2.1} \leq c \cdot n^2$ $\quad c \geq n^{0.1}$ $\quad c^{10} \geq n$ $\quad n \leq c^{10}$ → Constant

It Violates $\forall n \geq n_c$ condition of definition

Ex 4: $f(n) = n$ $g(n) = \log n$

$f \in O(g)$?

$f \notin O(\log n)$ → Answer



Ex 5 : $f(n) = n$ $g(n) = \log(n^2)$

$f \in O(g)$?

False

If $f \in O(g)$ then $n \leq c \cdot \log(n^2)$ → going to be false after a certain point

$n \leq 2c \cdot \log(n)$

$\therefore \; 2^{n-2c} \leq n$ $\quad (c=1) \Rightarrow 2^{n-2} \leq n$ $\quad n \geq 5$

$2^n \leq 2^{2c} \cdot 2^{\log n} = 2^{2c} \cdot n$

$X \to Y$

$\sim Y \to \sim X$

$a^{\log_a b} = b^{\log a}$

Ex6 $f(n) = \log(n)$ $g(n) = \log(n^2)$ $n \in \mathbb{N}$ → is a natural number variable.

$f \in O(g)$?? → True

$\log n \leq c \cdot \log(n^2)$ $\therefore \log n \leq c \cdot 2 \log n$ $c = 1$, $n_c = 1$

Ex7 $f(n) = \log(n)$ $g(n) = \log(\sqrt{n})$

$f \in O(g)$?? → True

$\log n \leq c \cdot \log(n^{1/2})$ $\therefore \log(n) \leq c \cdot \frac{1}{2} \log(n)$ $c \geq 2$, $n_c = 1$