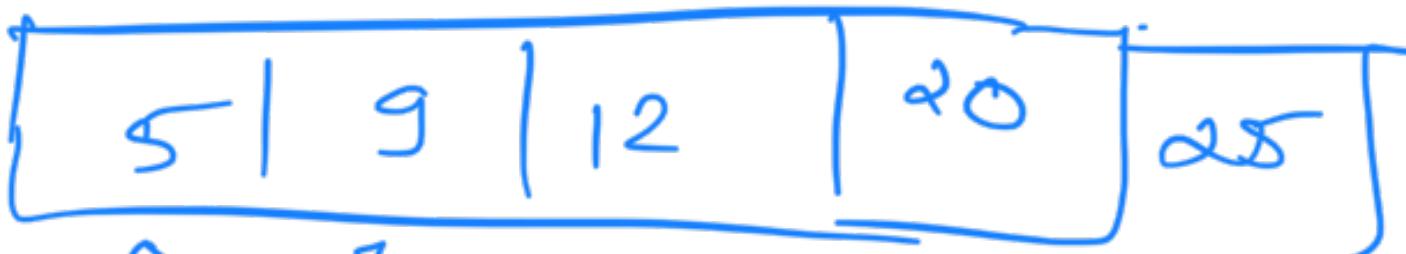


Linear Search

or Sequential Search



Algus 9
↑
i

Seq Search (A, i, x, m)

size
of
array

array index element

you want
to find

① Set $i = 0$

② If $i > n$

point ele. Not found

$O(N)$

③ If $A[i] = \infty$

point element Not found

④ $i = i + 1$

⑤ Exit

$i = 0$	If $i > 5$	$A[i] = 5$
$i = 1$	If $i > 5$	$A[i] = 9$
$i = 2$	If $i > 5$	$A[i] = 12$

↓
Found

⑥ Binary Search

→ p

Time Complexity $\Theta(\log n)$
Based on
divide & conquer
Data items should be in sorted
manner

5	12	15	20	25	30	50
0	1	2	3	4	5	6

Algo :

① While ($low \leq high$)
 $mid = low + high$

$$\text{mid} = \frac{\text{low} + \text{high}}{2}$$

② if $x = A[\text{mid}]$ return mid

③ if $x > A[\text{mid}]$

$$(\text{low}) = \text{mid} + 1$$

else

$$\text{high} = \text{mid} - 1$$

④ exit

find 12

A = [5	12	15	20	25	30	50]
	0	1	2	3	4	5	6

↑
low

high ← 10

~~(f)~~ $\text{mid} = \frac{\text{low} + \text{high}}{2} = \frac{0+6}{2} = 3$

$x = A[3] = 20 \neq 12$

so,

if $x > A[\text{mid}]$ ~~(x)~~ } $\text{low} = \text{mid} + 1$
 $20 > 20$ ~~(x)~~

else

$x < A[\text{mid}]$

$\text{high} = \text{mid} - 1$

$\text{high} = 3 - 1$
= 2

~~(g)~~ $\text{mid} = \frac{\text{low} + \text{high}}{2} = \frac{0+2}{2} = 1$

$$x = A[1] = 12$$

return 12



Bubble Sort

→ Each pair of Adjacent is compared
and elements are swapped - if they
are not in order

Worst time complexity = $\Theta(n^2)$

worst case

average

\forall element (\rightarrow N swap
 N times)

Algo 3

① While (list not get sorted)
 if (list [i] > list [i+1])
 swap (list [i], list [i+1])
 end i : R

end

exit

Flame: Companion
With
neigh hour



8		5		A		3		6
---	--	---	--	---	--	---	--	---



2		4		5		3		6
---	--	---	--	---	--	---	--	---

2		4		3		5		6
---	--	---	--	---	--	---	--	---



2		4		3		5		6
---	--	---	--	---	--	---	--	---

2		3		4		5		6
---	--	---	--	---	--	---	--	---

④

Selection Sort

→ In this sorting technique list is divided into two parts



Worst case
time complexity

$\Theta(n^2)$

Algo:

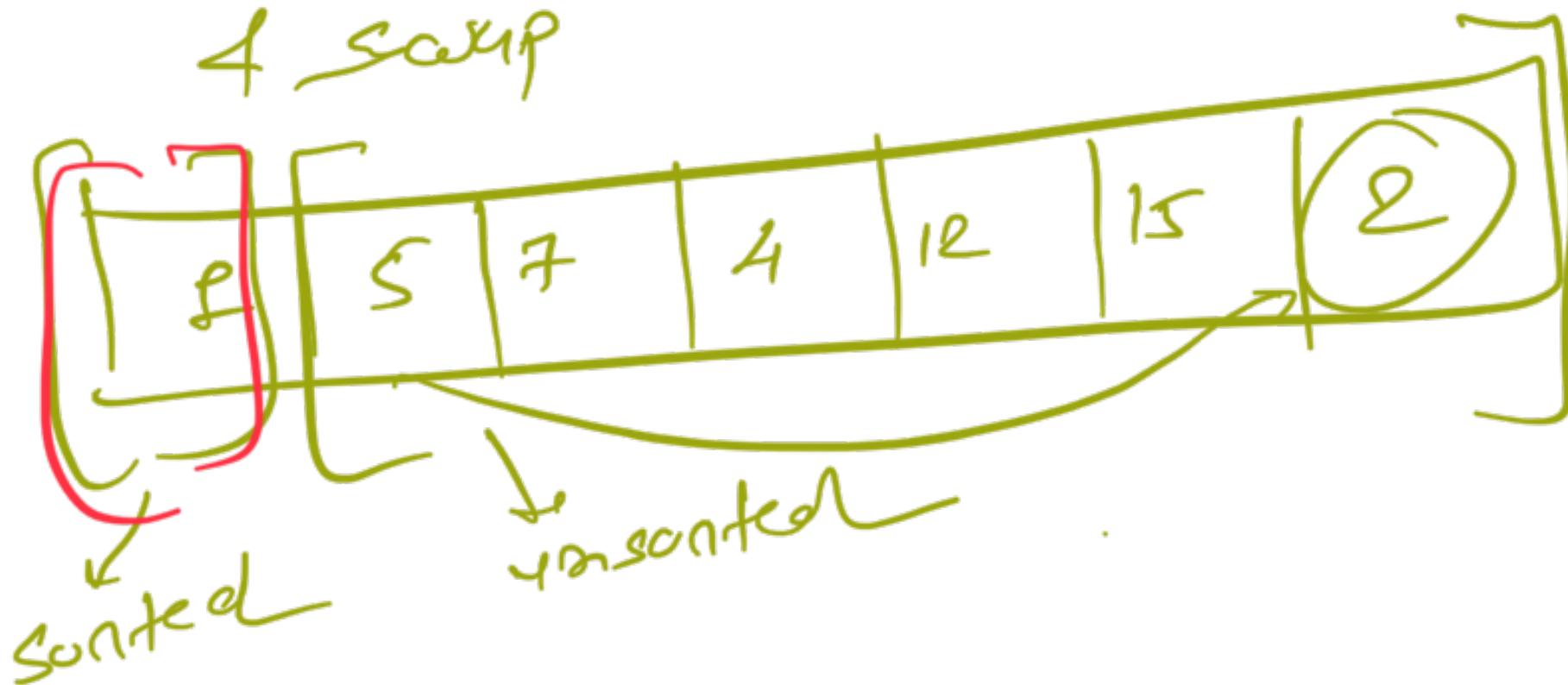
- ① set $\min = 0^{\text{th}}$ index element
- ② Search minimum element in the list
- ③ Swap with value at location \min
- ④ Increase \min to point to next element
- ⑤ Repeat until list is sorted





find minimum element

4 swap

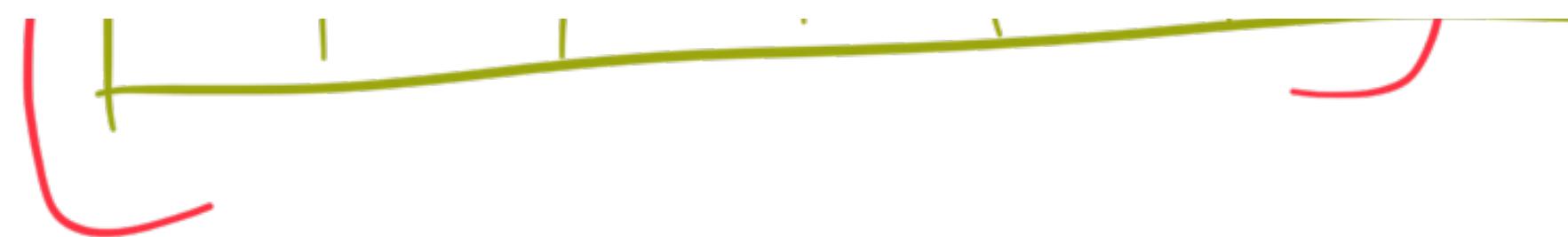


1	2	4	7	12	15	5
---	---	---	---	----	----	---

1	8	4	5	the	15	7
---	---	---	---	-----	----	---

1	12	4	5	7	15	12
---	----	---	---	---	----	----

1	2	4	5	7	12	15
---	---	---	---	---	----	----

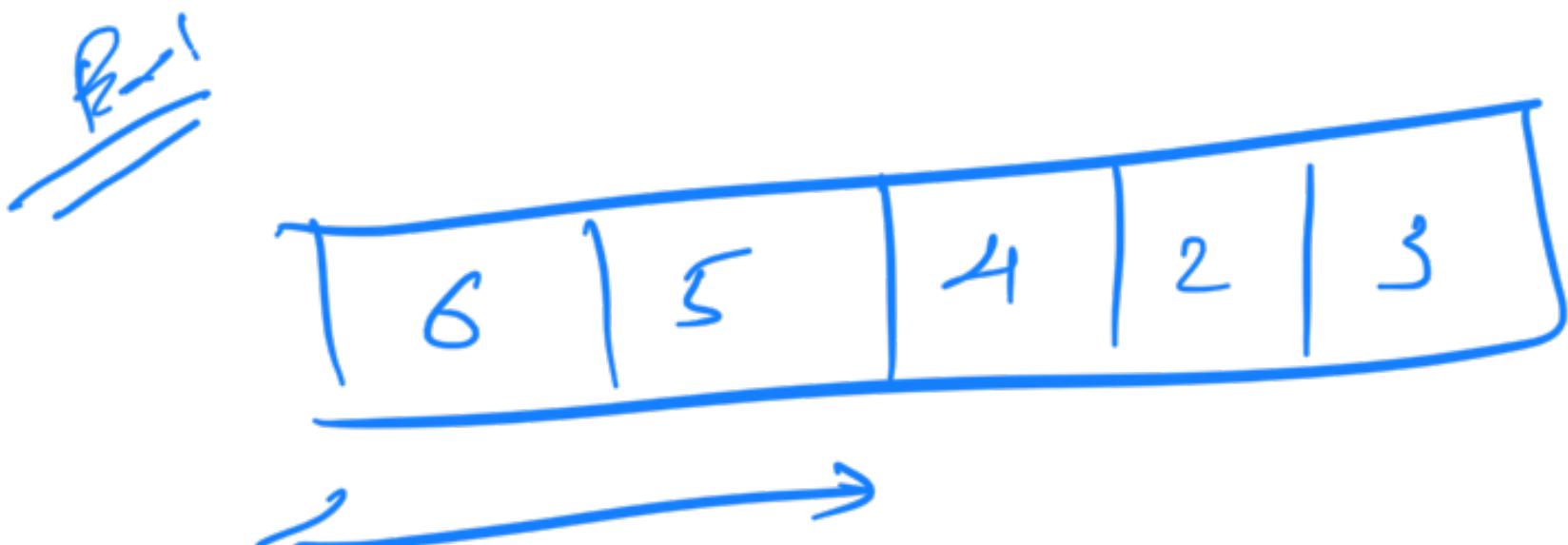


Insertion Sort

→ A sublist is
[as sorted Array] is maintained
which is always sorted

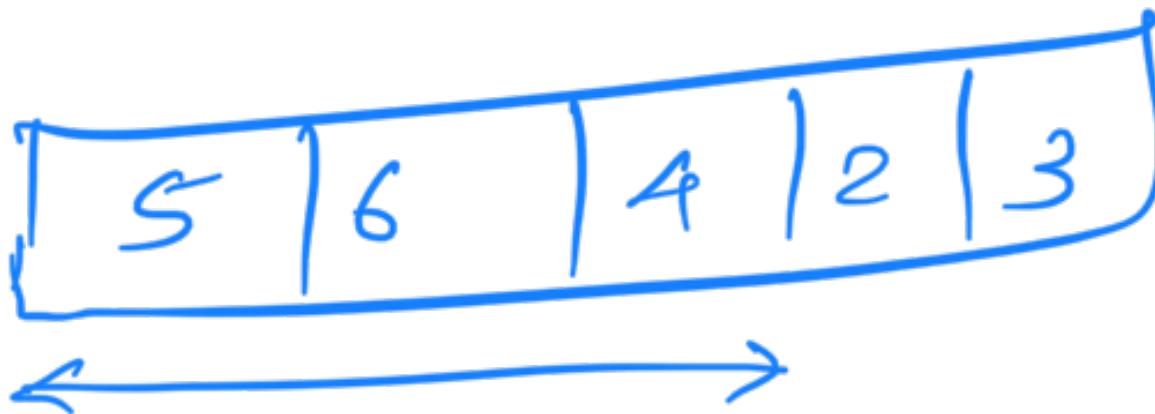
- Not for large Array
- time Complexity $\Theta(n^2)$

- $(n-1)$ pass are req. to sort n elements.
- In each pass we insert current element at appropriate place so that element in current range are in order

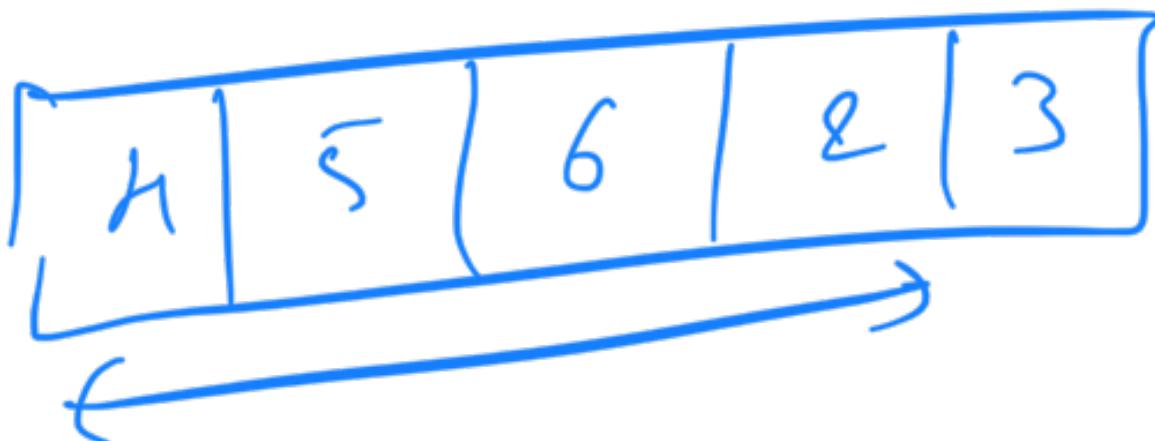


$\nearrow R$

pass : 1



pass : 2



pass : 3





* Merge Sort :-

- Based on Divide + Conquer
- time complexity $\boxed{\Theta(n \log n)}$
- If divides the array into halves
then moves them into sorted

mannet.

Alg0: Beg : start of Array

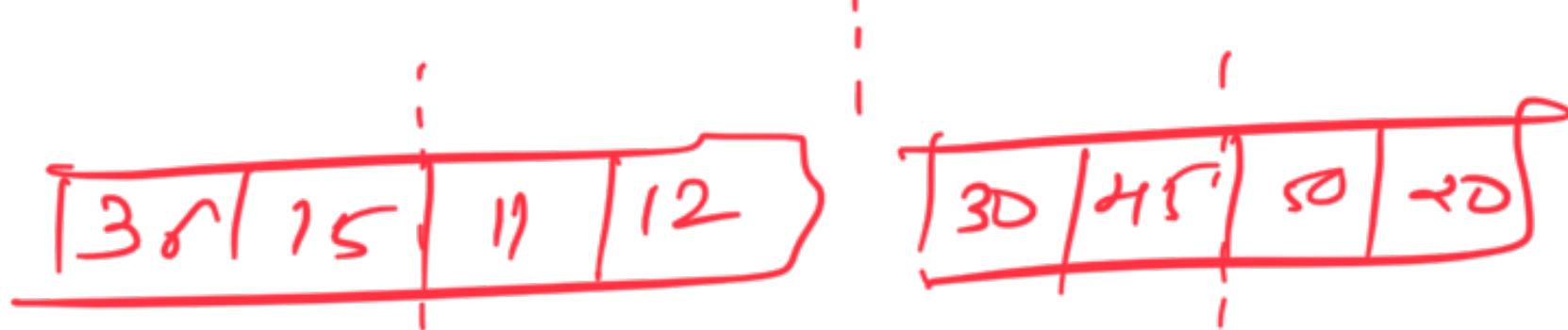
end : End of Array

if (beg < end)

then mid = $\left(\frac{\text{beg} + \text{end}}{2} \right)$

Ex:

36	15	11	12	30	45	50	20
----	----	----	----	----	----	----	----



first mid last

36 | 15 | 11 | 12 30 | 45 | 20 | 50

[36] [15] [11] [12] [30] [45] [50] [20]

Compare

among

15 | 36

11 | 12

30 | 45

20 | 50

11 | 12 | 15 | 36

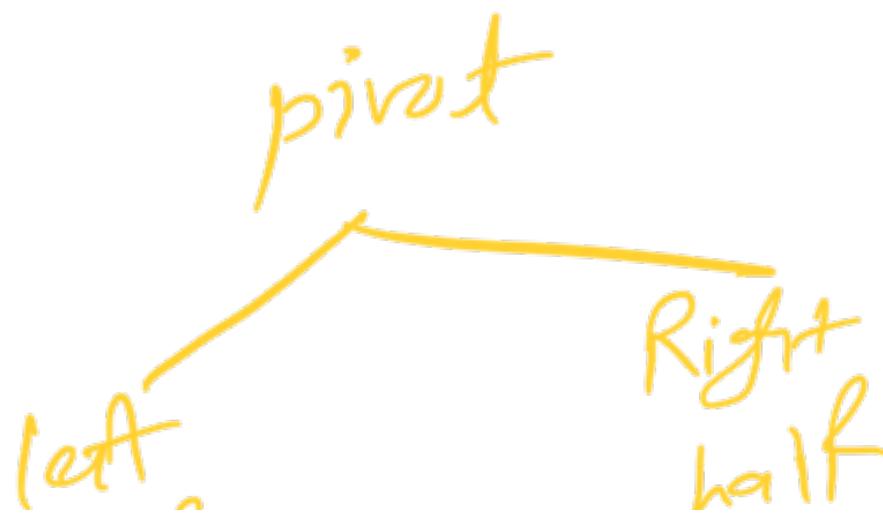
20 | 30 | 45 | 50

11 | 12 | 15 | 20 | 30 | 36 | 45 | 50

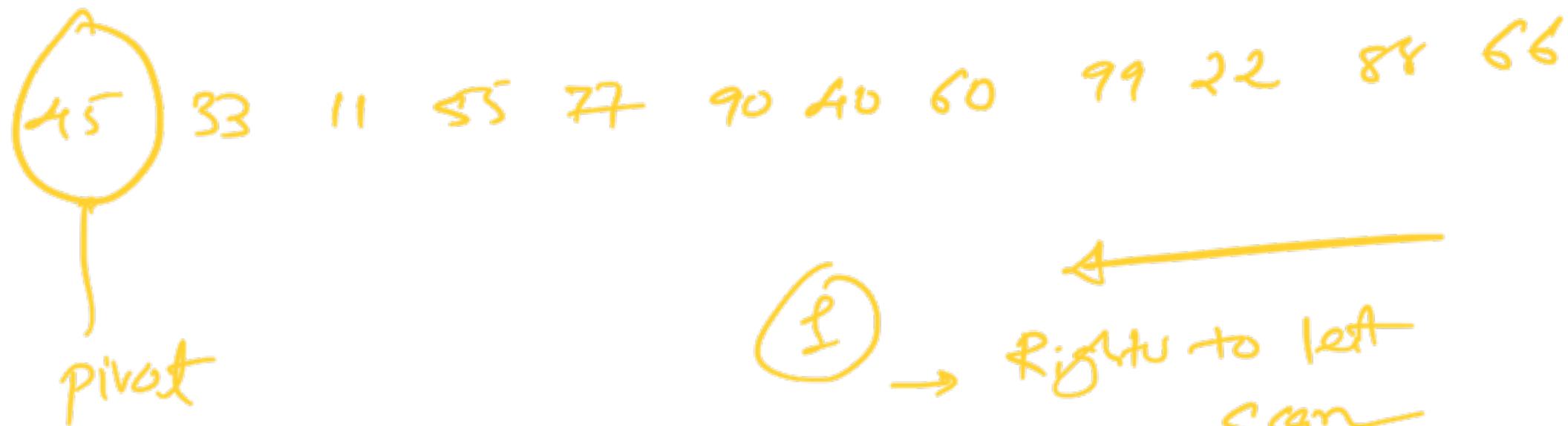
(*) Quick sort

- Based on partitioning of array of data in smaller group
- divide + conquer
- efficient for large size data

$$\boxed{\Theta(n \log n)}$$



half



l

→ Right to left

scan

→ if smaller than
pivot then
exchange

r

→ Left to Right scan

→ if Bigger than pivot
swap

~~44~~ 33 11 55 77 90 40 60 70 ~~22~~ 88 66

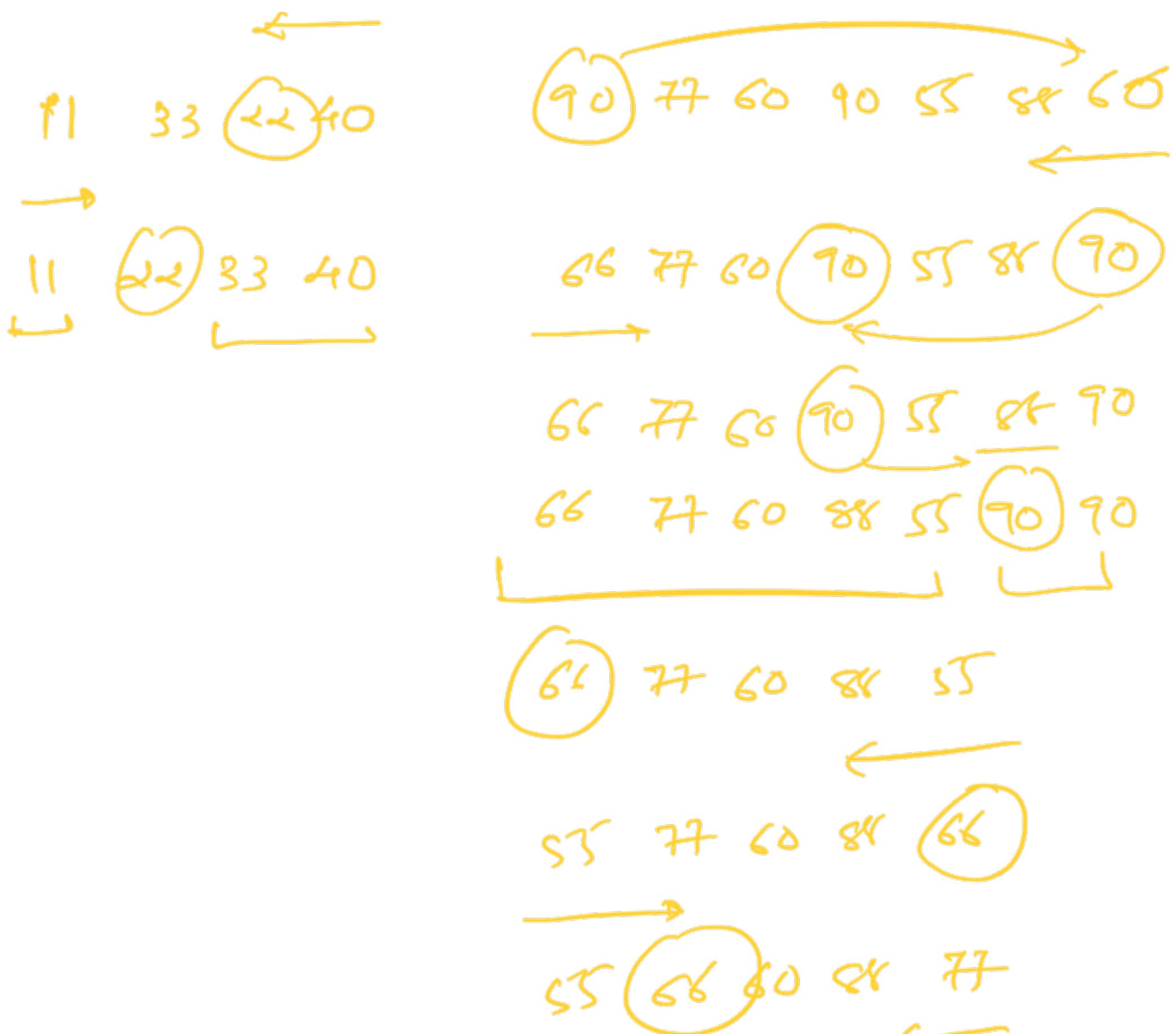
22 33 11 55 77 90 40 60 70 ~~44~~ 88 66

22 33 11 ~~44~~ 77 90 ~~40~~ 60 70 55 88 66

22 33 11 40 ~~77~~ 90 ~~44~~ 60 70 55 88 66

~~22~~ 33 11 40
Sublist

44 90 77 60 90 55 88 66
Sublist



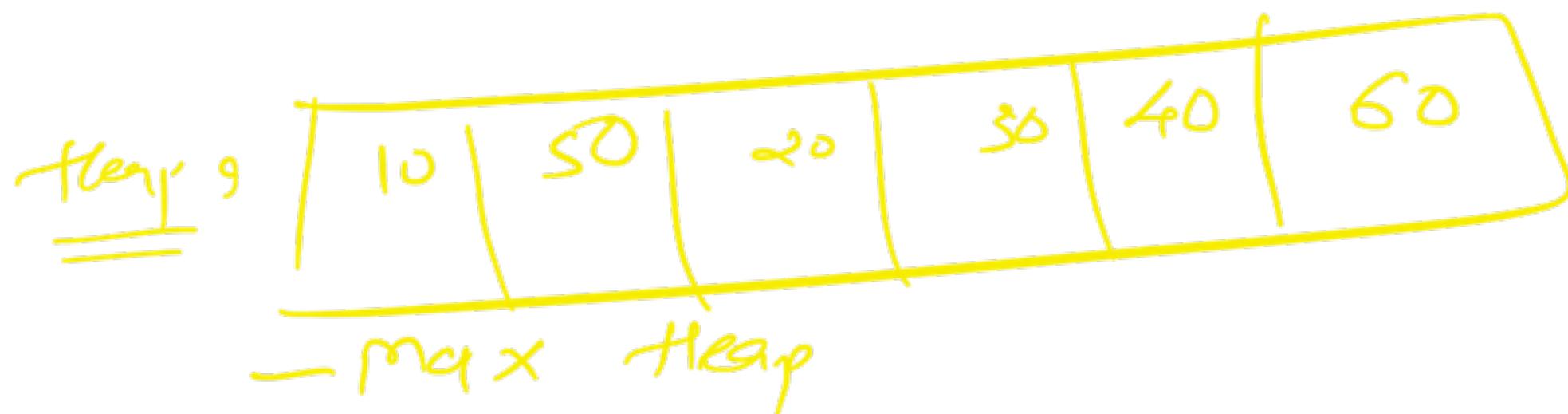


Heap Sorting

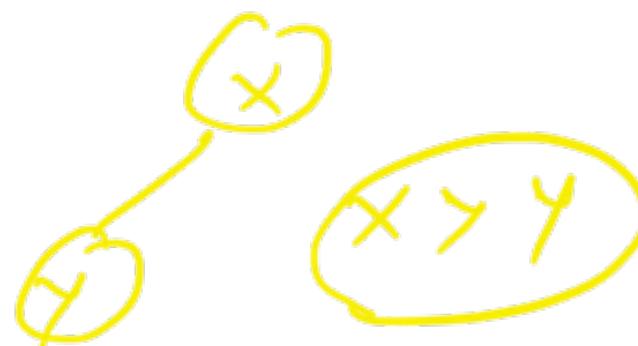
→ The sorting is done by first creating a heap tree from the given, i.e., arranging it. — max heap.

array + then sort 0

- Min Heap



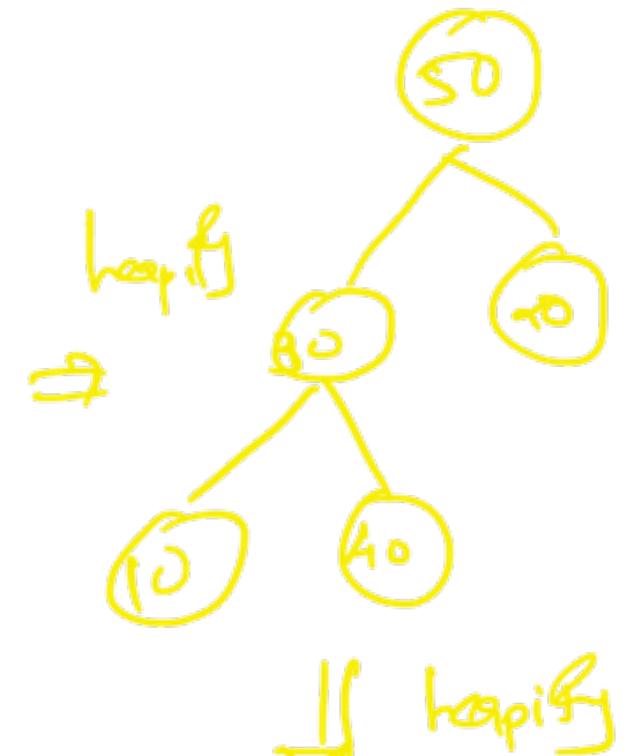
↳ parent heap > child heap



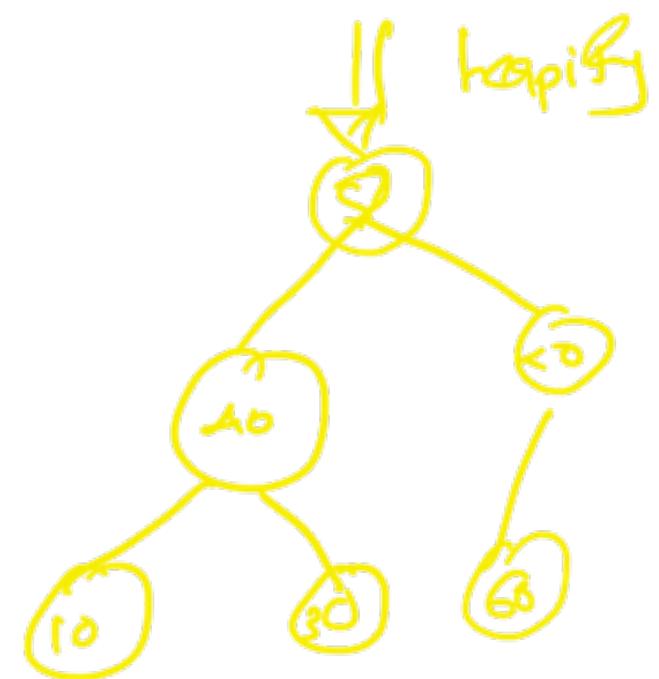
(50)



(10)

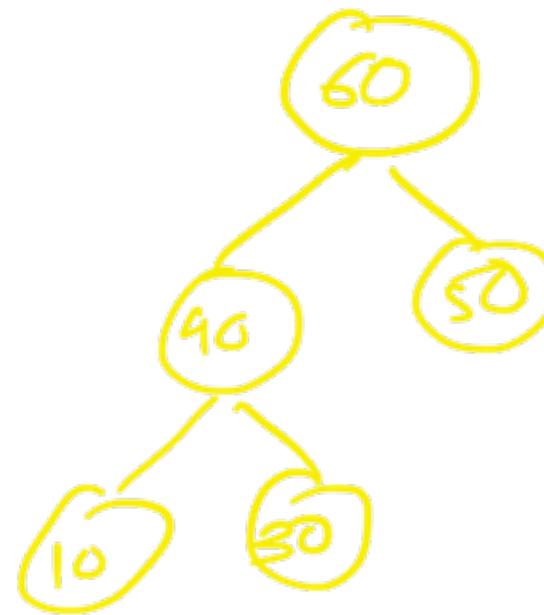
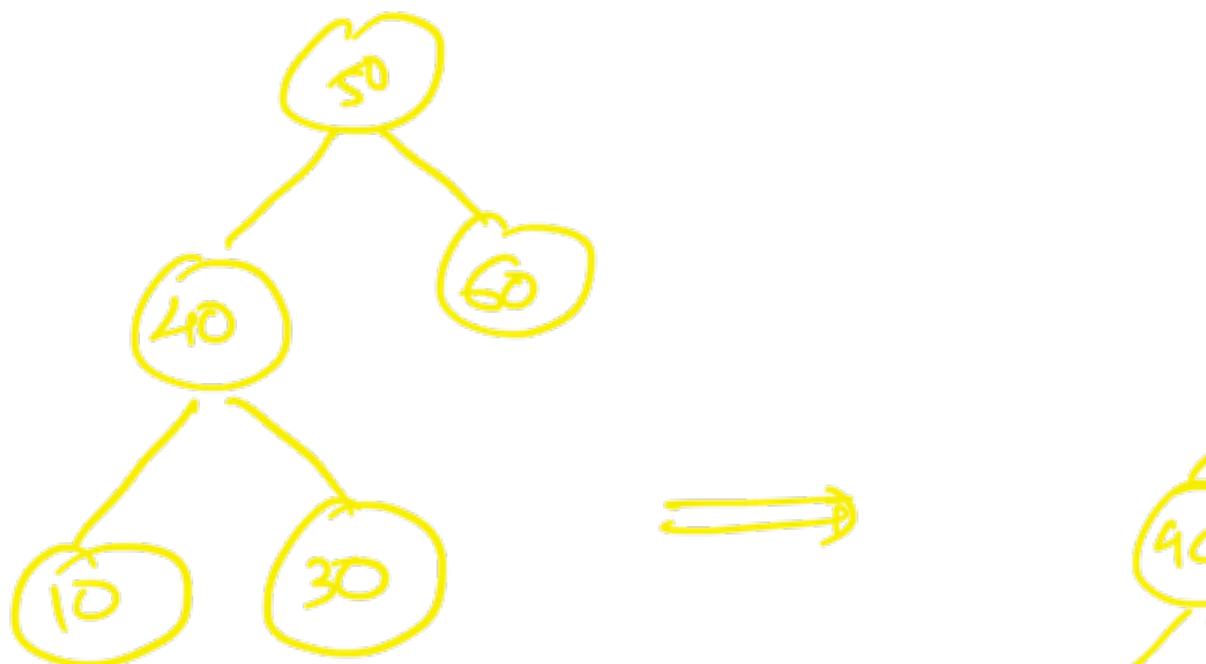


heapify

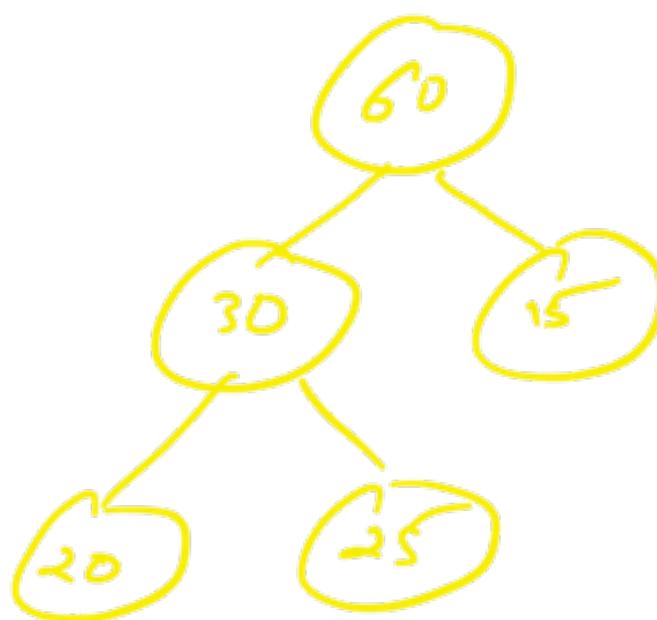


heapiify



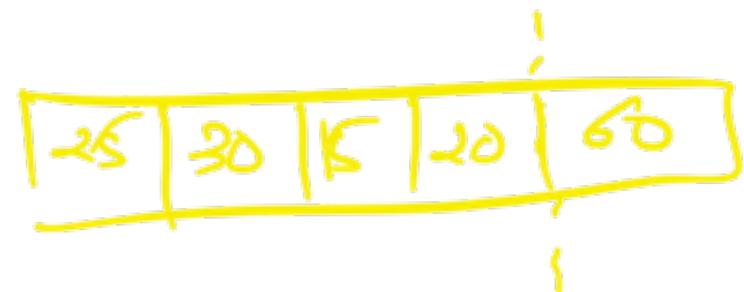
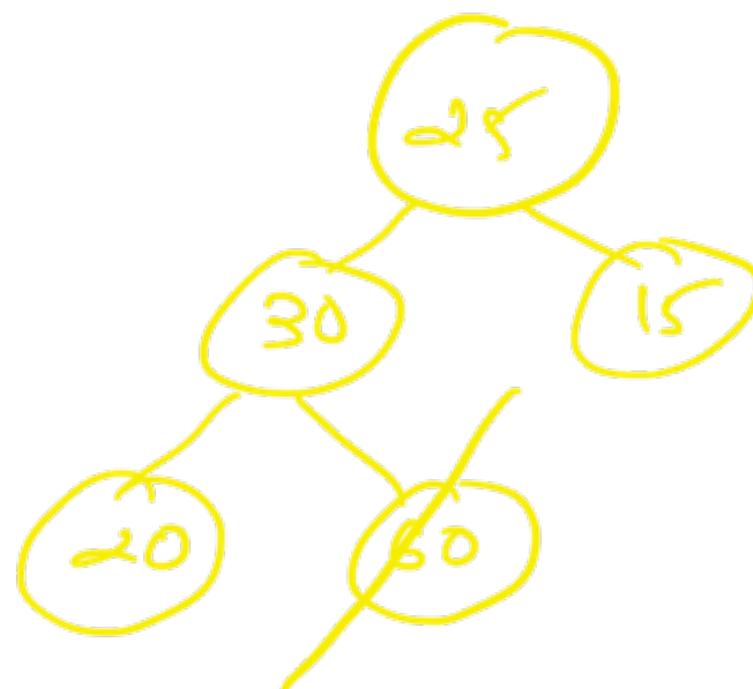


Ex:



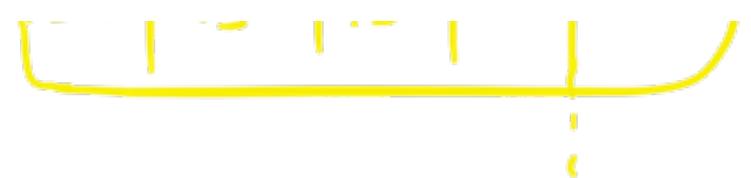
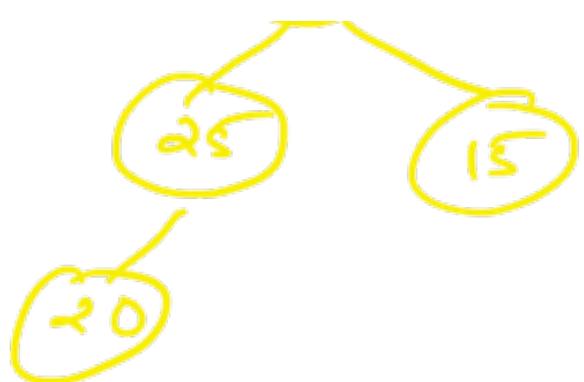
max heap tree

→ Interchange the largest element
with parent Node

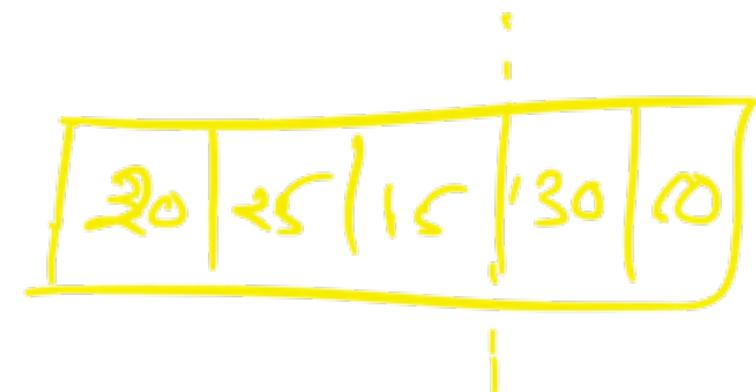


↓
heapify





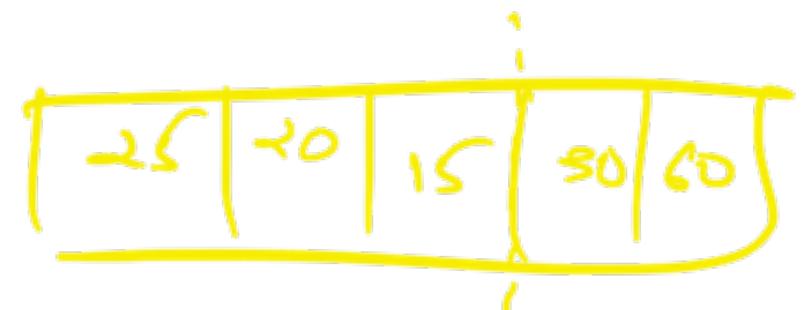
\Downarrow largest el. interchange



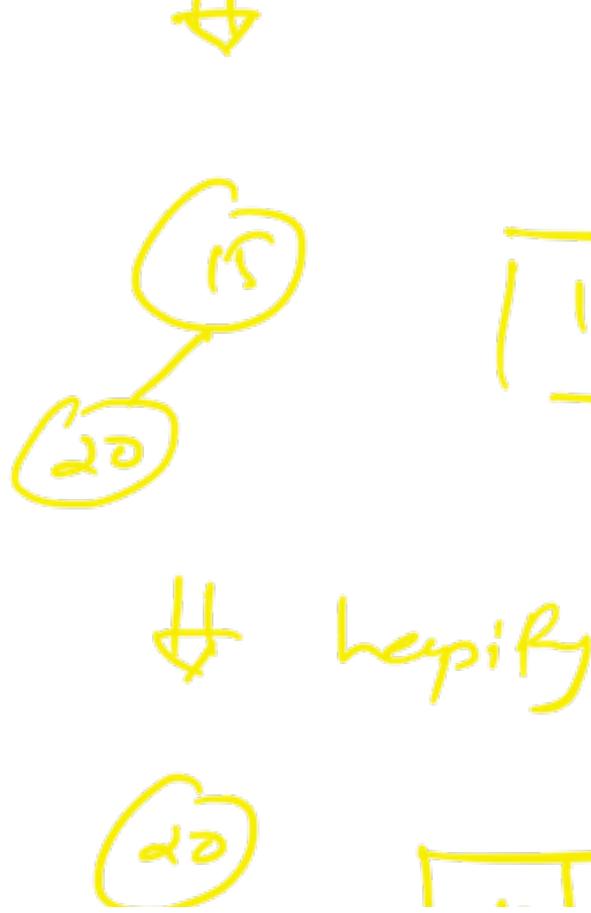
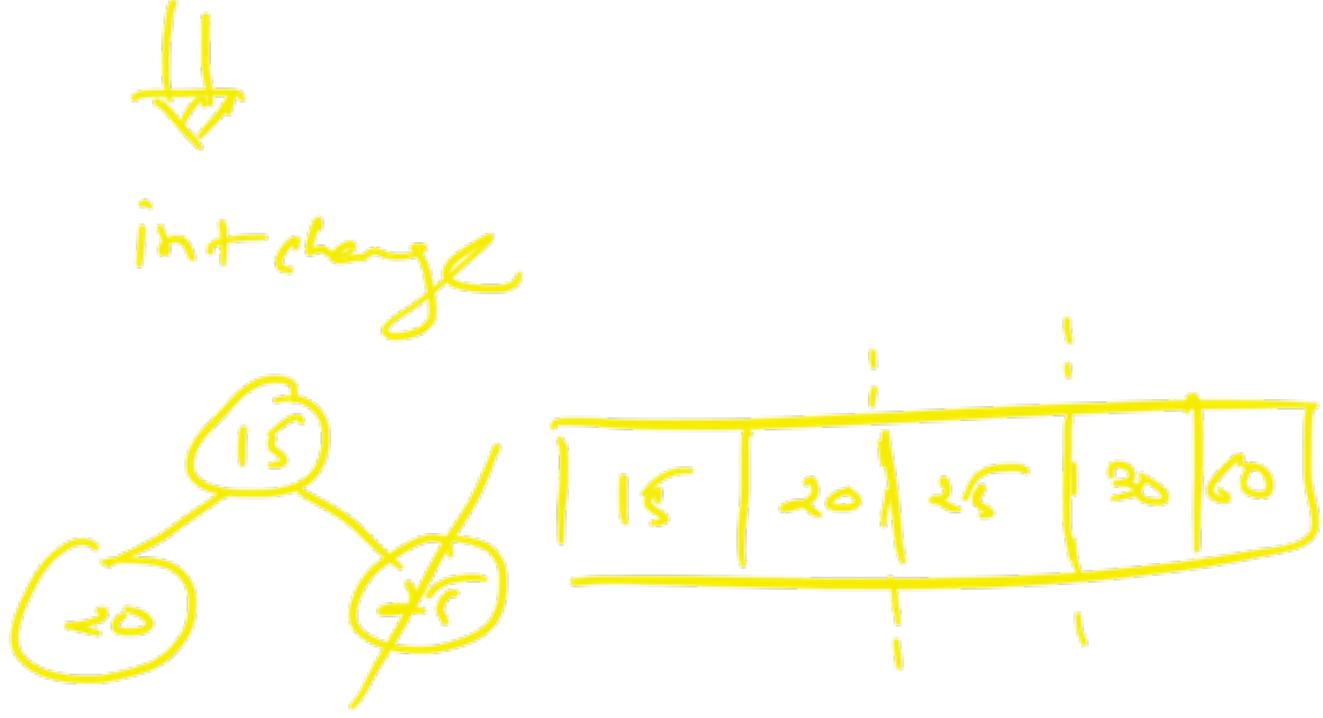
\Downarrow interchange

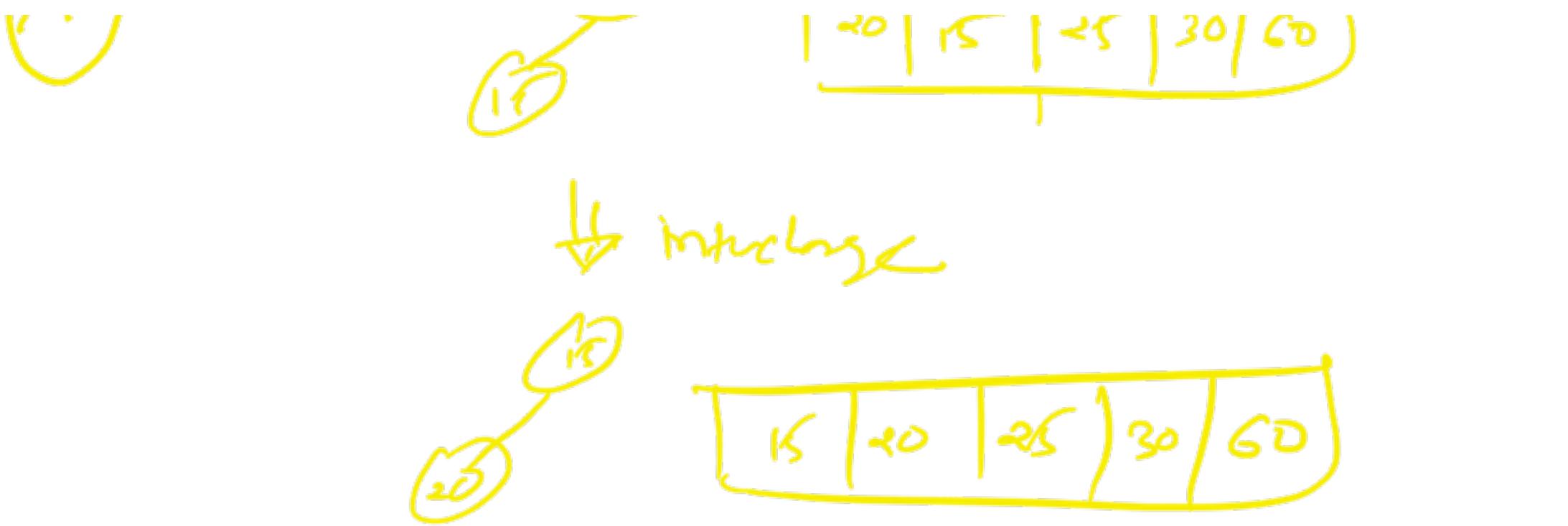


Verify



(2) ~~Latex~~
heaps
introduction





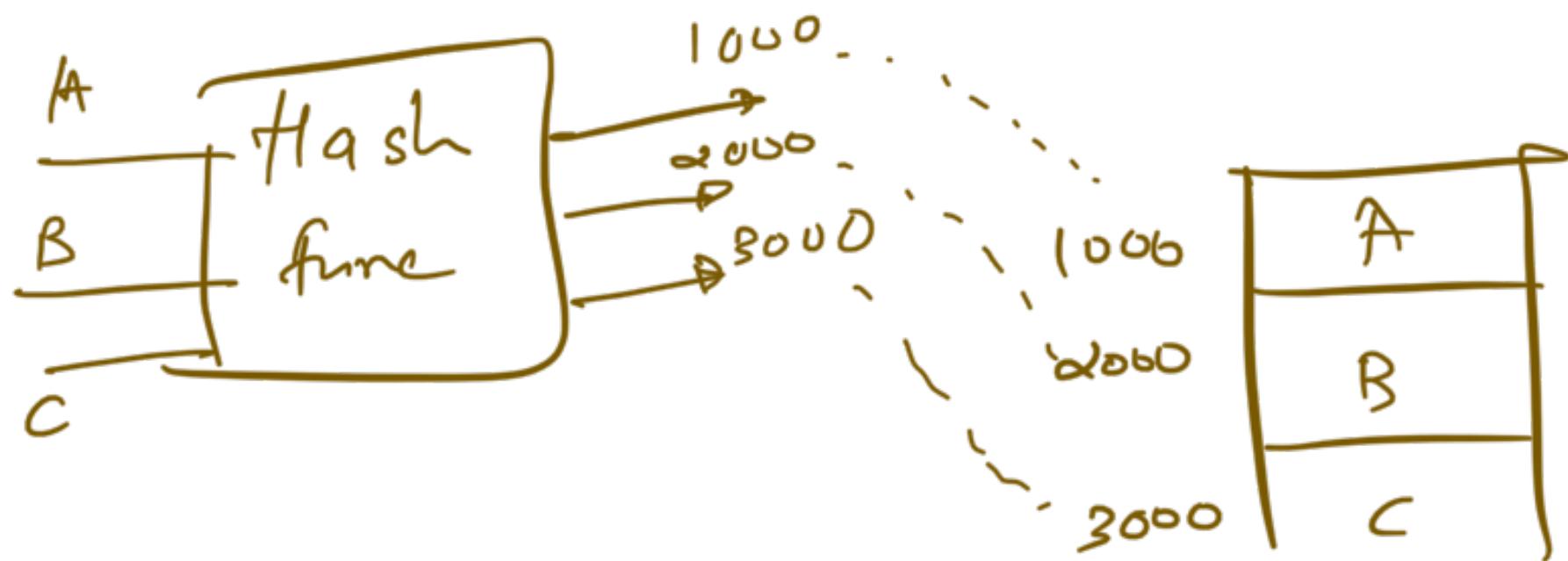
(*) Hashing (Searching)

→ Hashing involves less comparisons

→ Inserting and removing
and searching can be performed in
constant time

→ The goal of hashed function is to
find target data in only one test

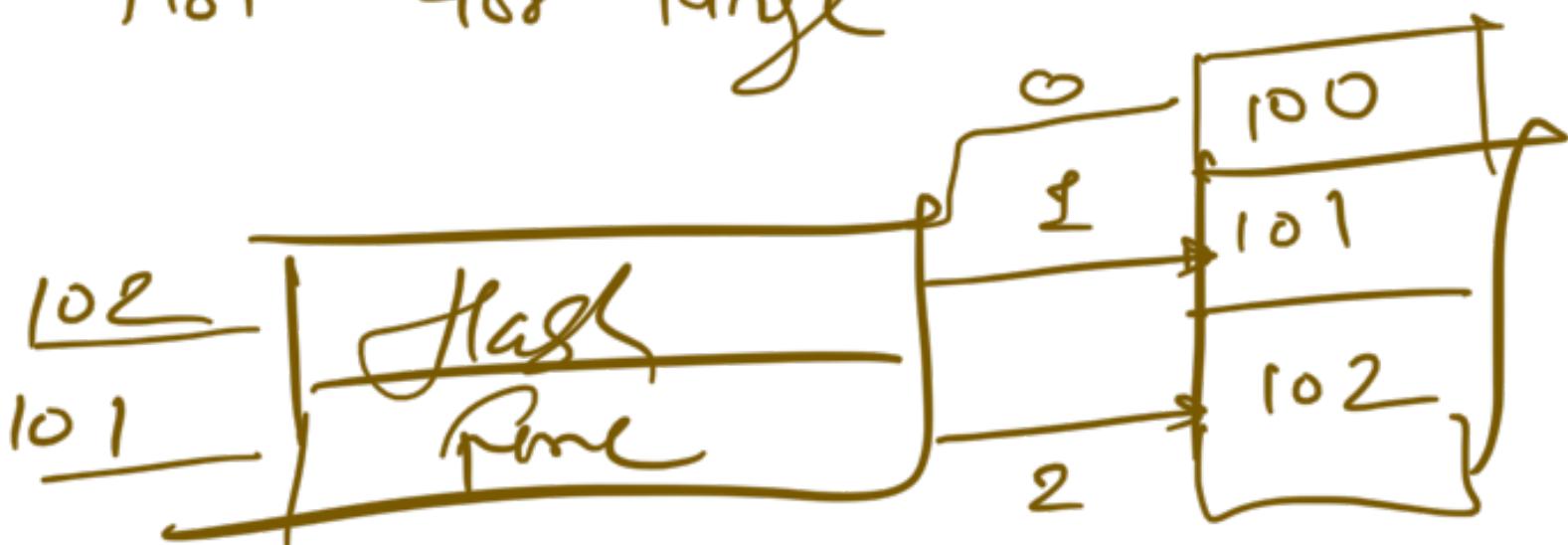
$$\boxed{\mathcal{O}(1)}$$



Types of Hash function

(E) Direct Hashing

- No Algorithm manipulation
- min No. of collision
- limited No. of keys
- Not for large



② Modulo-Division

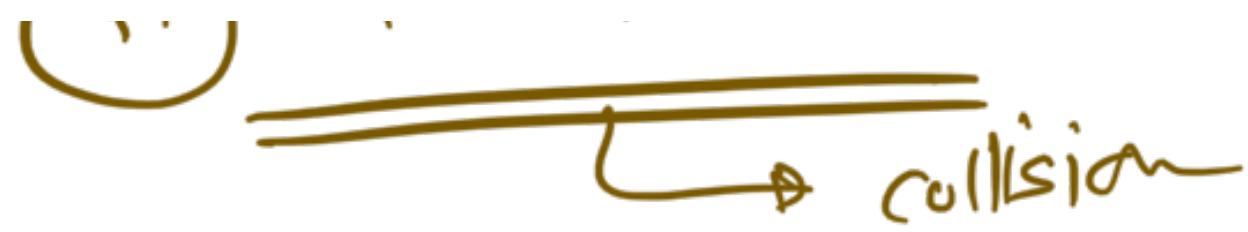
- Division remainder
- counts with any list size
- if (list size) is prime No.
then few collision

$$h(k) = k \% n$$

(101) if $n = 10$

$$\left\{ \begin{array}{l} 101 \% 10 = 1 \\ 100 \% 10 = 0 \end{array} \right.$$

(111) $111 \% 10 = 1$



③ Mid-Square hashing

→ Middle of Square

→ Square the key and take middle
of the ~~as~~ no. of address

will be the Address

if $k = 35$

$$k^2 = \boxed{1225}$$

$$n=2$$

$$h(k) = 22$$

(4) Folding Hashing

Fold Shift Hashing

→ key value is divided into parts and then added to seq. address

e.g. 123 456 789

Fold Boundary Hashing

left & Right nos. are folded on a fixed boundary 1/100

e.g.
123 456 789
Fixed

1	2	3
4	5	6
7	8	9
1	3	6
8	5	8

if $n=3$

$$h(c_k) = 368$$

3	2	1
9	8	7
4	5	6

$\overline{1764}$

$h(c_k) = 764$

if $n=3$

③ Subtraction Hashing

Subtraction a fix no. from key

$$h(c_k) = k - c$$

- No collision
- for small size

$$h(R) = R - C$$

$$G = 1000 - 1000$$

$$Z = 1002 - 1000$$

$$P_0 = 1040 - 1000$$

(#) Collision Resolution

↳ A collision occurs when a hashing algorithm produces the same address for a key and for a key that has been inserted.

addreses is already 2^{n-1}

~~or~~ $h(k) = k \% n$

$$n = 10$$

$$h(k) = k \% 10$$

$$\begin{aligned} h(101) &= 101 \% 10 \\ &= 1 \end{aligned}$$

$$\begin{aligned} h(111) &= 111 \% 10 \\ &= 1 \end{aligned}$$



collision Reshetion

Open Addressing

linked list
or chaining

+ linear Probe

+ Quadratic Probe

+ Pseudo Random

$$Y = \alpha \downarrow \text{key} + C \rightarrow \text{const}$$

$$h(k) = Y \times n$$

④ Open Addressing

→ there are address are
selected for an open \cong
unoccupied element where
new data can be placed

~~→~~ linear probe

$$h(k) = k \% n$$

$$h(101) = 101 \% 10 = 1$$

$$h(111) = 111 \% 10 = 1$$

$$2 = 1$$

$$\left. \begin{array}{l} \dot{z} = (i+k) \times n \\ k = 1, 2, 3, 4 \end{array} \right\}$$

④ Quadratic Probe

$$\left. \begin{array}{l} h(k) = k \times n \\ h(101) = 101 \times 10 = 1 \\ h(111) = 111 \times 10 = 1 \end{array} \right\}$$

$$z = (i+k) \times n$$

$$k = 1, 2, 3$$

$$0 \leq i \leq 2$$

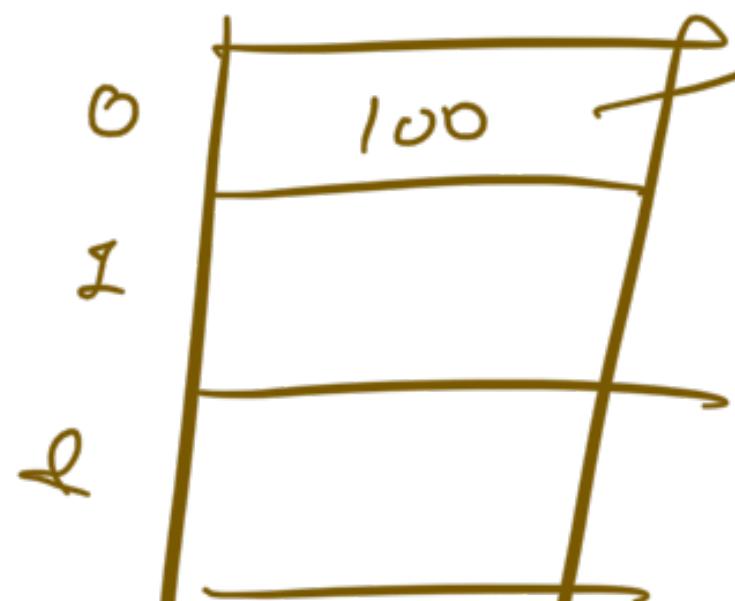
$$\leftarrow \leftarrow \leftarrow \leftarrow$$

$$5 \times 10 = 2$$

II

linker list

linker list



$$h(k) = k \times 10$$

$$h(100) = 0$$

$$h(200) = 0$$