Data Structure

Interface / API (Specification)

Data

↳ Which type of data?

↳ - Sequence, Sets

A → $a_1, a_2 \dots, a_n$

(Problem)

#data fixed
↳ Static

Which operations are needed?

- Build(A)
- length(A)
- get_data(A, i)
- set_data(A, i, a')

Data Structures (Representation)

- Define the structure & how to perform all the operations.

(Solution)

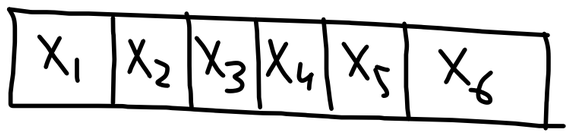
⇓

How to perform the operations efficiently?

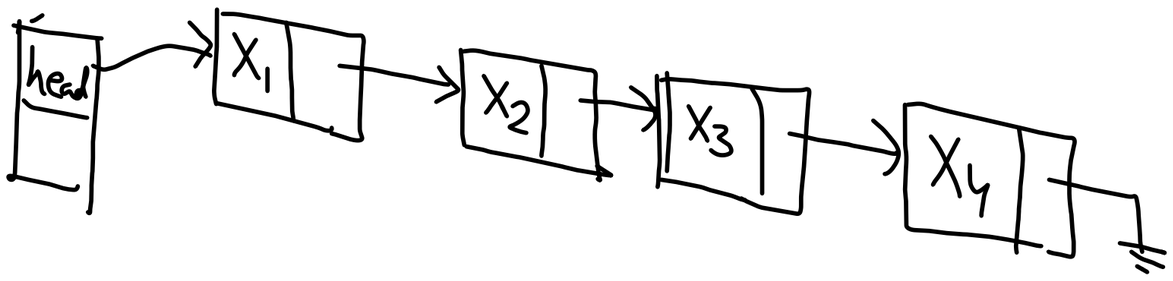
- Array
- Link-list

	Build	length	Get at	Set_at
Array	O(n)	O(1)	O(1)	O(1)
Link-list	O(n)			

$n \rightarrow$ # data



$X \Rightarrow$



$X = \{ X_1, \ldots, X_n \}$

Get at (X, i)

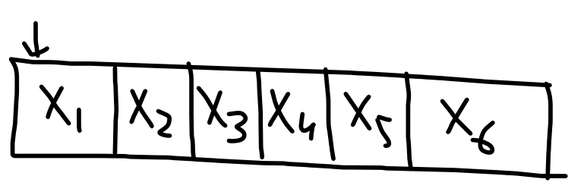
return X[i];

Set at (X, i, x)

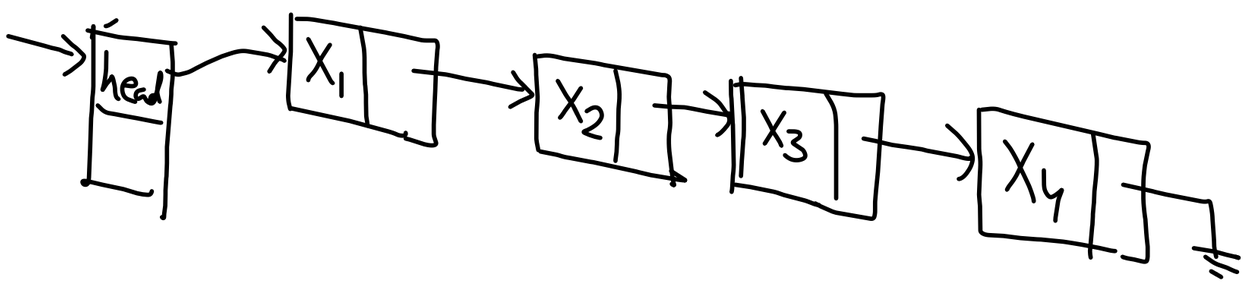
X[i] = x;

	Build	length	Get at	Set_at
Array	O(n)	O(1)	O(1)	O(1)
Link-list	O(n)	O(1)	O(n)	O(n)

$n \rightarrow$ # data



X ⇒



$X = \{x_1, \ldots, x_n\}$

Get at (X, i)

return X[i];

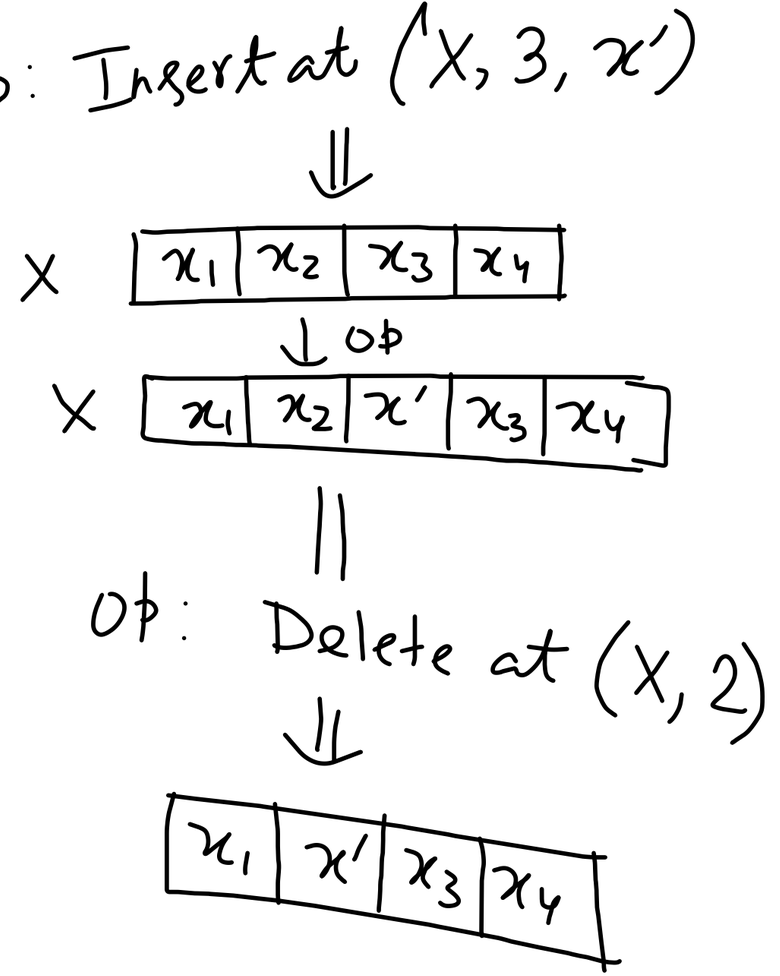
Set at (X, i, x)

X[i] = x;

Dynamic Operations

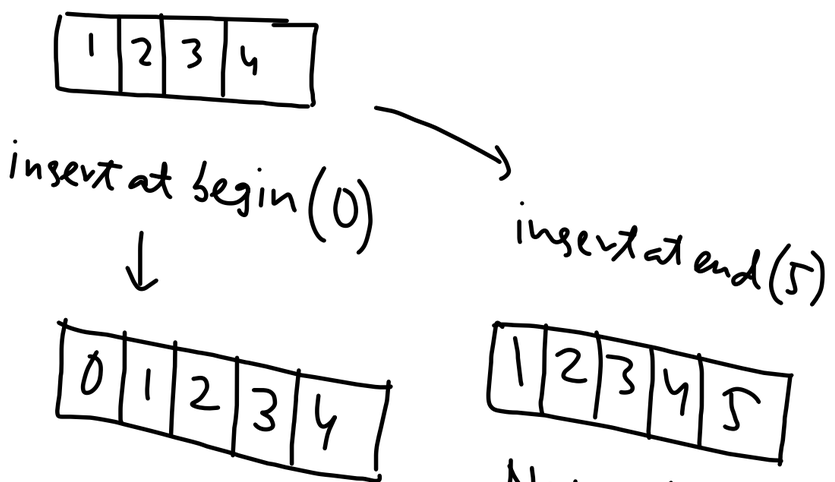
- Insert at (X, i, x)
  - Insert at Begin
  - Insert at End
- Delete at (X, i)
  - Delete at Begin
  - Delete at End

op: Insert at $(X, 3, x')$

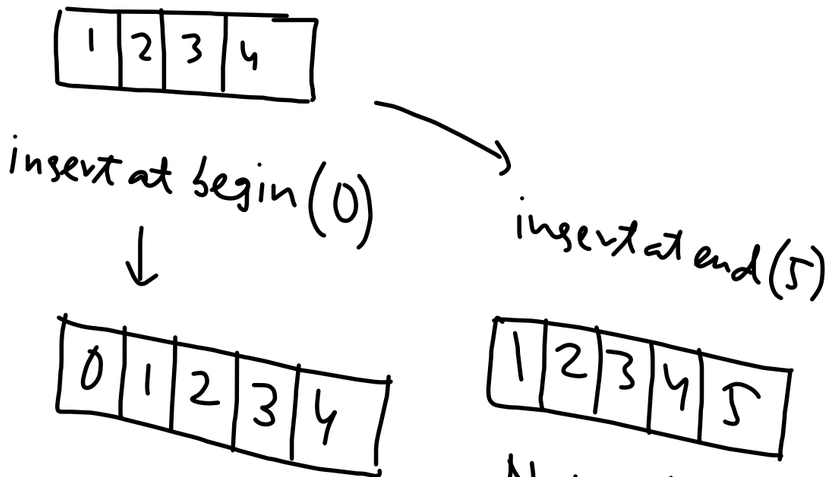


op: Delete at $(X, 2)$

	Insert at begin	Insert at end	Insert at i	Delete at begin	Delete at end	Delete at i
Array	O(n)	O(n)	O(n)	O(n)	?	O(n)
Link-list	O(1)	O(n) O(1) [with modified Linklist]	O(n)	O(1)	O(n)	

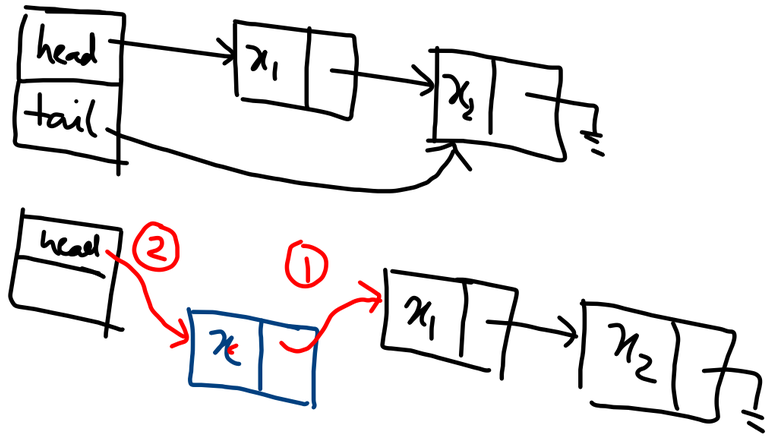


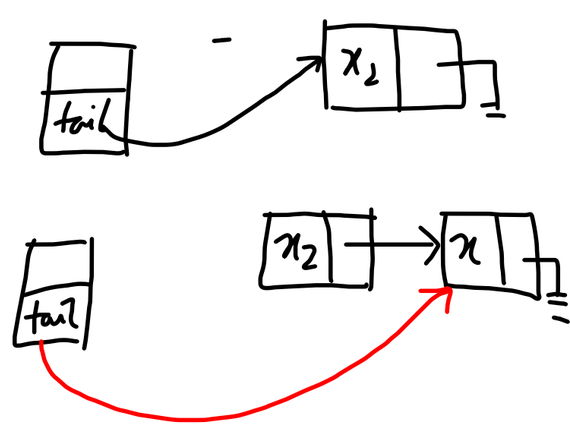
	Insert at begin	Insert at end	Insert at i	Delete at begin	Delete at end	Delete at i
Array	O(n)	O(n)	O(n)	O(n)	?	O(n)
Link-list	O(1)	O(n) O(1) [With modified Linklist]	O(n)	O(1)	O(n) O(1) [Doubly link list]	O(n)



Not O(1)

- Allocate space for the new array.

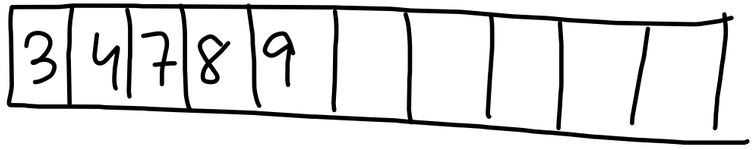




Dynamic Array

- Size
- length

A



{ length = 5
Size = 10

If (length == size)

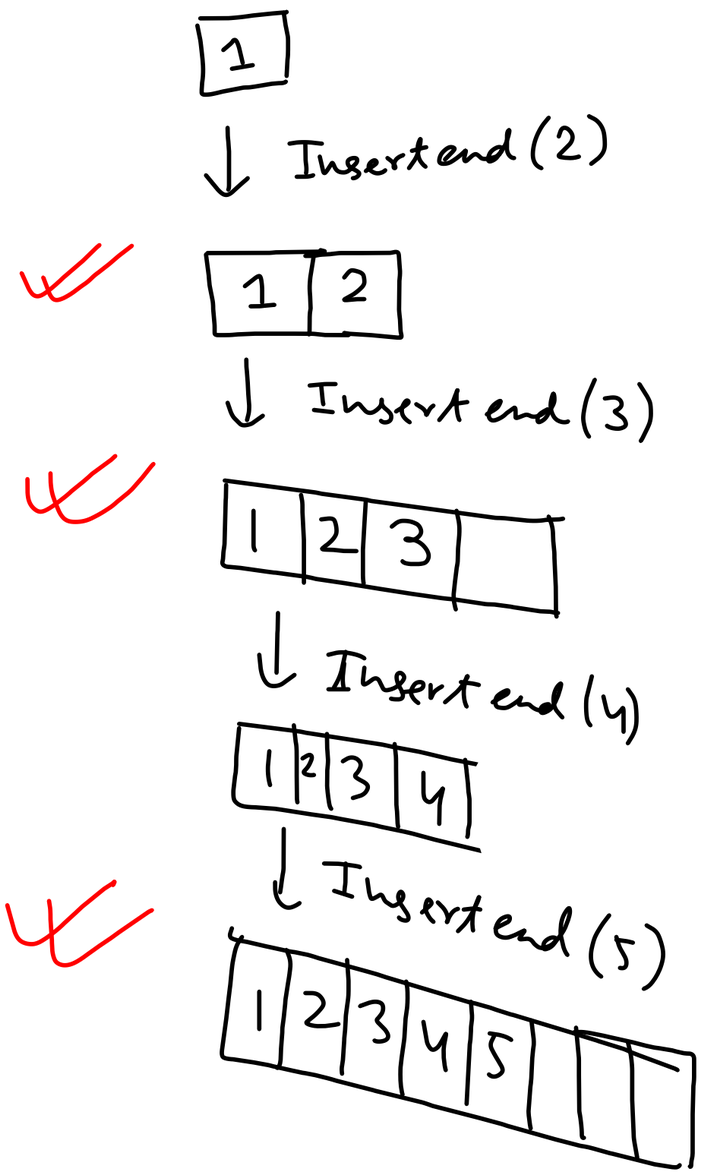
↓ insert at end (A, 10)

{ A[length + 1] = 10
length += 1

Allocate memory of a new array of size = 2 · size

$n = 2^k$

Insert at end for n times



Extra work

$$2^0 + 2^1 + 2^2 + 2^3 + \cdots + 2^{k-1}$$

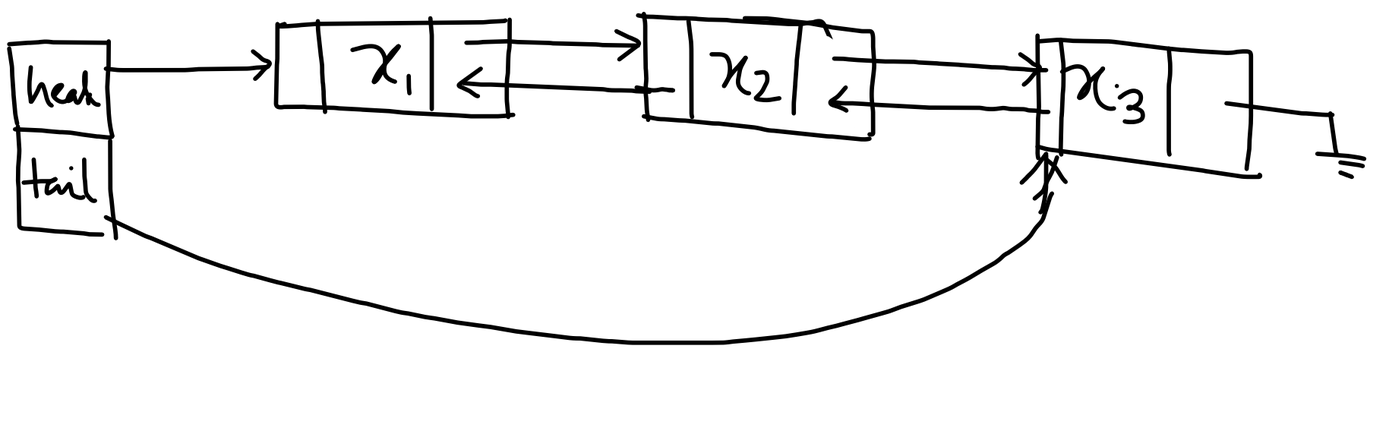
$$= 2^k - 1$$

$$= O(n)$$

n Insert at end $= O(n)$

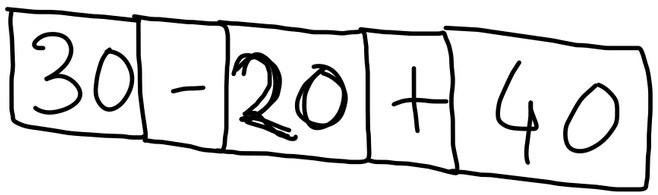
Amortized cost $\rightarrow O(1)$

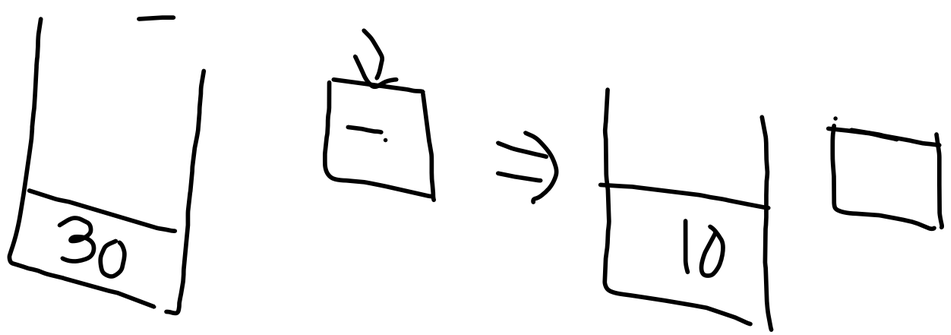
Doubly Linked-list



- Stack (LIFO)
- Queue (FIFO)

} Deletion operation is pre-defined





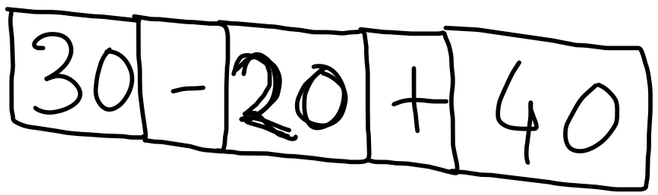
Stack

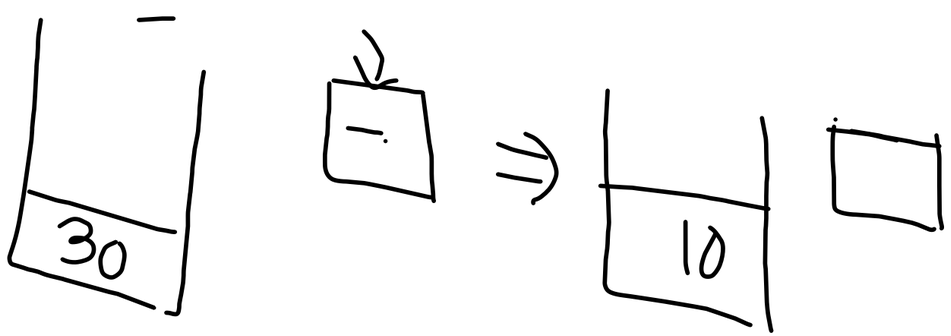
Insertion → push
deletion → pop

a op b — infix
a b op — postfix

- Stack (LIFO)
- Queue (FIFO)

} Deletion operation is pre-defined





Stack

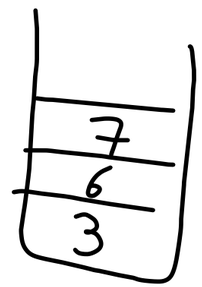
Insertion → push
deletion → pop

a op b — infix
a b op — postfix

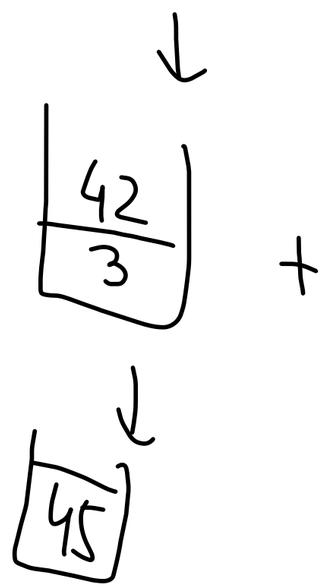
$3 + 6 * 7 - 2$

$= 3\ 6\ 7\ *\ +\ 2\ -$

postfix



*



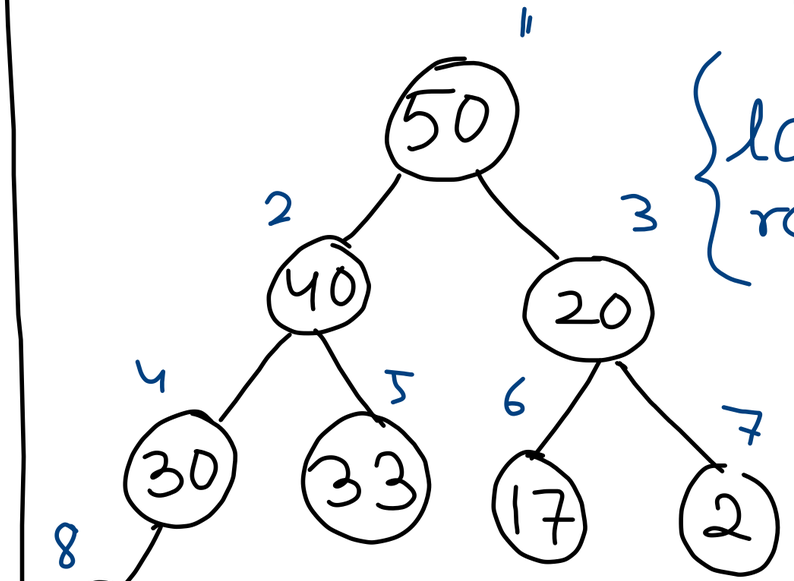
+

Heap

- max-heap

  (parent value ≥ both its Child value)

- almost full- binary tree



$parent(i) = \lfloor \frac{i}{2} \rfloor$

$lchild(i) = 2i$

$rchild(i) = 2i+1$

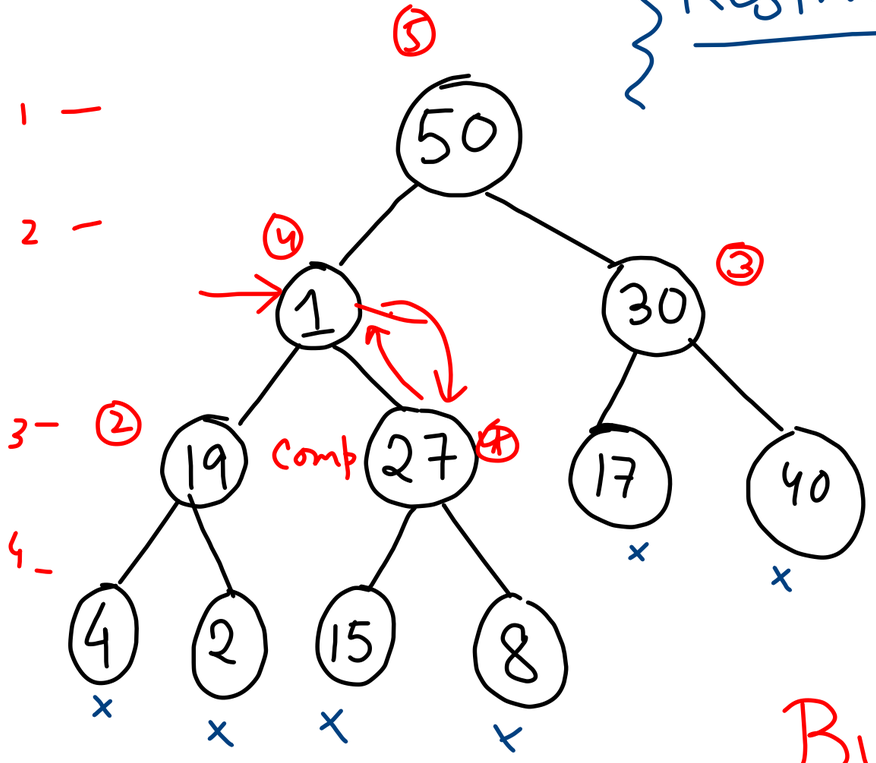
$n = 2^k + i$

$\forall i,\ A[i] \geq A[2i] \geq A[2i+1]$

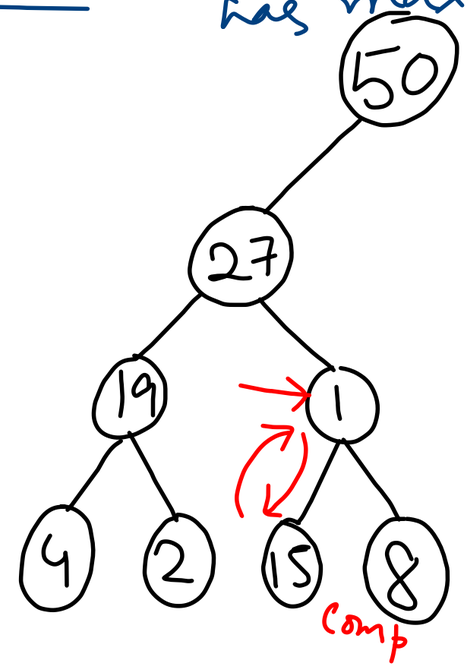
A →

50	40	20	30	33	17	2	23

Restrictions → Sub-tree under node i has max-heap property



#leaf = $\lceil n/2 \rceil$



Build-Heap (A, n)

For i = $\lceil n/2 \rceil$ to 1

heapify (A, i)

Heapify (A, i)

if (A[i] < A[largest]

Swap (A[i], A[largest])

Heapify (A, largest)

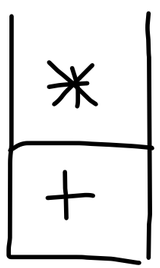
Complexity = O(log n)

⇒ O(n log n)

⇓

O(n)

3 + 6 * 7 ↓ − 2



Infix to Prefix

pref (−) ≤ pref (*)