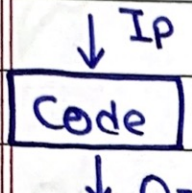
classmate
Date
Page

★ DS Algo



★ Algo
→ Work for any input
→ Produce correct output
→ Consumes bounded resources
time, space
power, data
packet

★ Heuristic :
Case if any algo does not follow one or more of above mentioned points

★ Data Struction
Storing data in structured manner
→ Storing
→ Accessing
→ retrieving
→ rearrange
→ update

★ How to prove the three points for algo
→ Model of computation
Word-RAM → Random access memory ↗ Randomly access any address in const time
↓
number / float

CLASSMATE
Date
Page

★ Operation in constant time

→ All the mathematical operation
+, −, *, /, %, [ ], [ ]

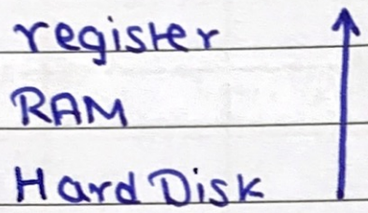
→ Copy any word

→ Accessing an address

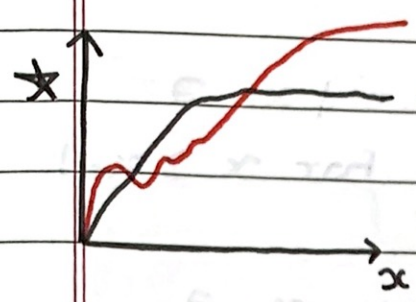
→ Read any address

→ Control Operation (<=, <, >=, >, if, returning)

★ There is no memory hierarchy in Word-RAM



★
```
int ct;
ct = 0;
for (int i = 0; i<10; i++)      → 40 units
        ct++;
Output ct
```



1) $O(g(x))$: { $f(x)$: ∃ a positive constant c & x_0 ∋ $0 \le f(x) \le cg(x)$ for $x \ge x_0$ }

then $f(x) \in O(g(x))$

eg: $x^2 + 2x + 1 \in O(x^2)$

$x^2 + 2x + 1 \le x^2 + 2x^2 + x^2$ $\quad x \ge 1$ (x_0)

$\le 4x^2$ (c)

$n! \in O(n^n)$

$\log(n!) \in O(n \log n)$

$n \in O(2^n)$

$\log n \in O(n)$

$$f(x) = \sum_{i=0}^{n} a_i x^i \in O(x^n)$$

$$f(x) \le \sum_{i=0}^{n} |a_i| x^i \le x^n \left(\sum_{i=0}^{n} |a_i|\right) \quad \text{for } x \ge 1$$

$f_1(x) \in O(g_1(x))$

$f_2(x) \in O(g_2(x))$

then $(f_1 + f_2)(x) \in O(\max(g_1(x), g_2(x))$

$(f_1 f_2)(x) \in O(g_1 g_2(x))$

2) $\Omega(g(x)) = \{f(x) : \exists$ +ve const c & $x_0 \ni$
$0 \le cg(x) \le f(x)$ for $x \ge x_0\}$

3) $\Theta(g(x)) = \{f(x) : \exists$ +ve const $c_1, c_2, x_0 \ni$
$0 \le c_1 g(x) \le f(x) \le c_2 g(x)$ for $x \ge x_0\}$

↓

$f(x)$ is order of $g(x)$

★ $\log(n!) \in \Theta(n \log n)$

$\log(n!) = \log n + \ldots + \log(n/2)$
$+ \log(n/2 - 1) + \ldots + \log(2) + \log 1$

$\geq \left(\frac{n}{2} + 1\right) \log\left(\frac{n}{2}\right) + \left((n-1) - \left(\frac{n}{2} + 1\right)\right) \log 2$

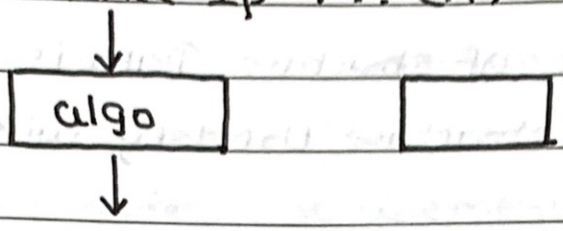
$\geq \frac{n}{2} \log n - 3 \geq \frac{n}{3} \log n$ for $n \ge n_0$

★ $\max(f(x), g(x)) \in \Theta(f(x) + g(x))$

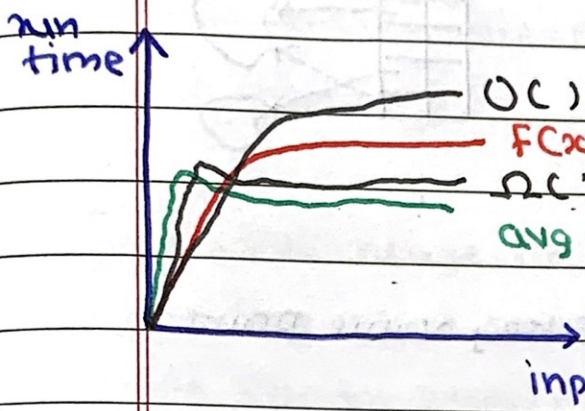
4) $o(g(x)) = \{f(x) : \lim_{x \to \infty} f(x)/g(x) = 0\}$

5) $\omega(g(x)) = \{f(x) : \lim_{x \to \infty} f(x)/g(x) = \infty\}$

* Prb.... Instance Ip ... (n) ← Input Complexity



Output ← Output complexity



O( ) → worst case asymptotic upper bound of the algo

if Θ( ) : tight bound

* Space complexity:

Input complexity + Algo complexity + Output complexity

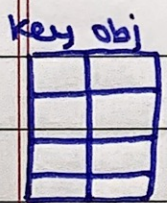
Workspace complexity

If O(1): Inplace algo

* Hard Problem: No efficient algo is known
* ADT (Abstract Data type)

Before coding we define structure of data like class, struct, object

(These org. structures known as ADT)



classmate
Date
Page

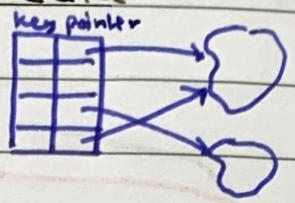
* Satelite Data:

In this type of structure Data is outside & We have to structure the key with pointer to data.

→ Adv:

multiple key can point to same data

no wastage of memory.



* ADT for object set

Object of ADT:

S: An unordered collection of key value pairs.

Functions:

→ Search (S, key)

insert (S, x)

delete (S, key)

min (S)

max (S)

replace (S, key, new)

traverse (S, (func*)(obj))

classmate
Date
Page

* Array vs Linked List

Array Adv: vs Linked List Adv

① Disadvantage of array: (dynamic)

→ contiguous memory requirement

→ resizing involves bulk copy

→ deleting / inserting an element r in the middle is inefficient

② Disadvantage of Singly linked list:

→ Sequential traversal

→ Cannot exploit locality based caching

→ maintaing & storing pointers with each block.

* Adv of doubly linked list

→ reverse traversal

* Disadv of doubly linked list

→ additional pointer to prev node to be maintained & saved with each node.

★ Dummy Blocks in DS (Sentinel)

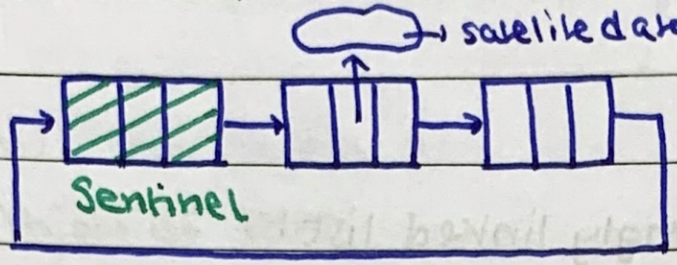
1) Increased speed of operations

2) Reduced algo complexity & code size

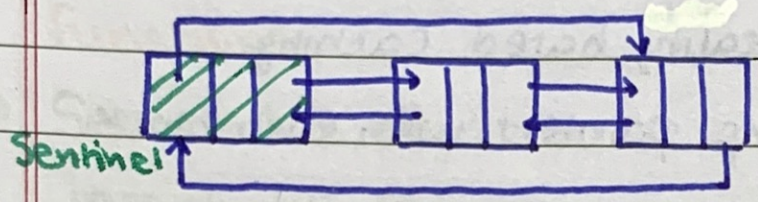
★ To add node to head of linked list ↑start

we have to check if (L.head != NULL)

Circular Singly linked list



Doubly



classmate
Date
Page

★ Stack

1) LIFO
   - → Dec to Binary Conversion ($25_{10} \rightarrow 11001_2$)
   - → Postfix expression evaluation
   - → Infix to postfix conversion

2) Precedence

^	Right to Left
* /	Left to right
+ -	Left to right

★ Infix → Postfix

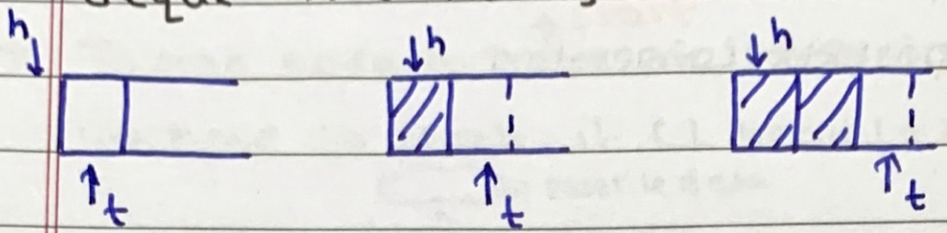
1) Scan from L to R
2) If Scanned chrc is operand output it
3) If stack empty or top has '(' or precedence of scanned op > S.top push the operator to stack.
4) Else pop

(A + (B * C - (D / E ^ F) * G) * H) ....

ABC * DEF ^ / G * - H * +

★ Queue

1) Job Scheduling, Message buffers - FIFO
2) enque t shifts by 1 - O(1)
   deque h shifts by 1 - O(1)



★ Deque (Deck)

ADT:

```
void* dequefront()
void* dequeback()
int enquefront(void*)
int enqueback(back*)
```