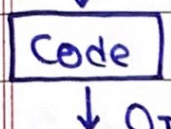
classmate
Date
Page

★ 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

★ Operation in constant time

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

→ Copy any word

→ Accessing an address

→ Read any address

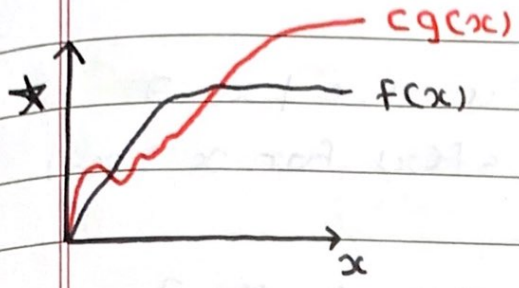
→ Control Operation (<=, <, >=, >, if, returning)

★ There is no memory hierarchy in Word-RAM

register
RAM
Hard Disk

↑ Speed

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



1) $O(g(x))$: { $f(x)$: $\exists$ a positive constant c & x_0 $\ni$ $0 \le f(x) \le c\,g(x)$ 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)

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

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

$n \in O(2^n)$

$\log n \in O(n)$

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

$f(x) \le \sum_{i=0}^{n} |a_i| x^i \le x^n \left(\sum_{i=0}^{n} |a_i|\right)$ 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 $g(x)$

★ $\log(n!) \in \Theta(n \log n)$

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

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

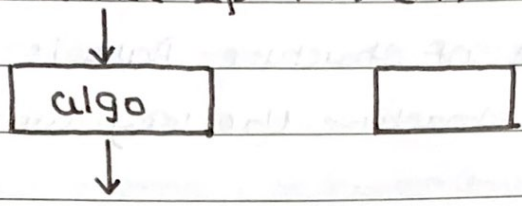
$\geq \frac{n}{2}\log n - 3 \geq \frac{n}{3}\log n$ for $n \ge n_0$

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

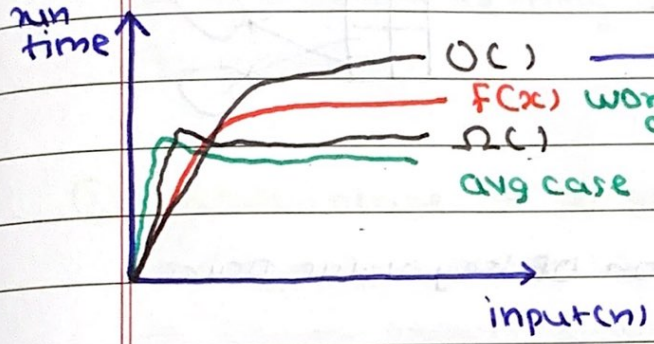
4) $o(g(x)) = \{f(x) : \lim_{x \to \infty} f(x)/g(x) = 0\}$

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

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



Output ← Output complexity



→ worst case asymptotic upper bound of the algo

if Θ( ) : tight bound

★ Space complexity:

Input complexity + Algo complexity + Output complexity

Workspace complexity

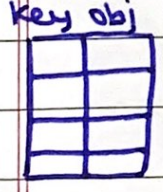
If Θ(1): Inplace algo

★ 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)



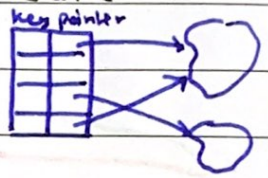
* Satelite Data:

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

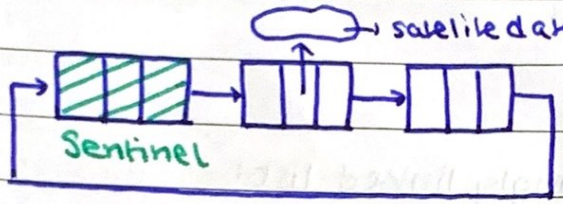
→ additional pointer to prev node to be maintained & saved with each node.

* Dummy Blocks in DS (Sentinel)

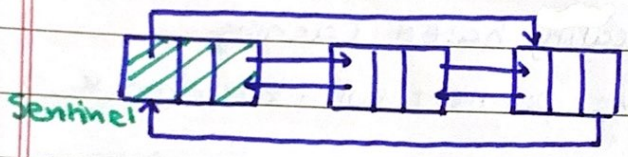
1) Increased speed of operations
2) Reduced algo complexity & code size

* To add node to head ↑start of linked list 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) ....

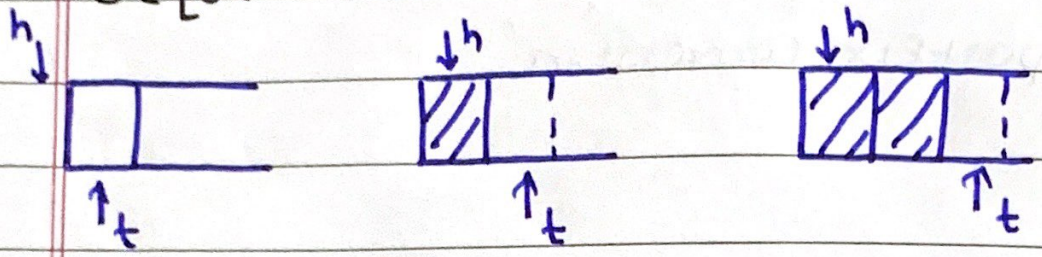
ABC * DEF ^ / G * - H * +

* 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()

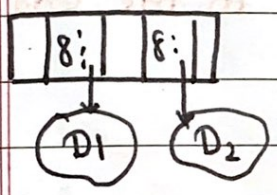
void* dequeback()

int enquefront(void*)

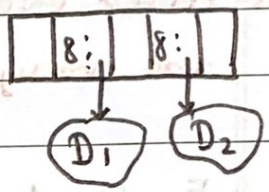
int enqueback(back*)

★ Sorting

1) Worst Case Time
2) Work space
3) Avg case time
4) Avg work space
5) Inplace: O(1) - workspace
6) Stable:



Order before sorting of two keys i same value preserved



7) Adaptive:

Algorithm should work acc. to no. of inversions

2 4 1 3 5 No. of inversions : 3

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

8) Online: (Interactive problem types)

★ Bubble Sort:

5 1 4 2 8 (If inversion two consecutive elements are swapped)

1 5 4 2 8

1 4 5 2 8

least after pass → 1 4 2 5 8 → absolute max element.

Pass 1

★ After every pass max element is to the end of the array, total (n-1) passes

Pass 0	5	3	1	9	8	2	4	7
Pass 1	3	1	5	8	2	4	7	(9)
2	1	3	5	2	4	7	(8)	
3	1	3	2	4	5	(7)		
4	1	2	3	4	(5)			
5	1	2	3	(4)				
6	1	2	(3)					
	1	(2)						

1) $O(n^2)$ # $\sum_{i=1}^{n-1} n-i = n(n-1)/2$

2) Inplace algo

3) Stable (Swap only if strictly less)

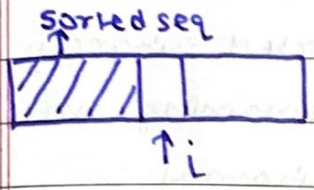
4) Online (Flag)

5) Adaptive (Flag)

6) No. of comp $\Theta(n^2)$

7) No. of swap $\Theta(n^2)$

★ Insertion Sort:



6 5 3 1 8 7 2 4

5 6 3 1 8 7 2 4

3 5 6 1 8 7 2 4

1 3 5 6 8 7 2 4

1 3 5 6 8 7 2 4

1 3 5 6 7 8 2 4

1 2 3 5 6 7 8 4

1 2 3 4 5 6 7 8

★ No of comparisons

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

★ No. of swaps $= n(n-1)/2$

1) $O(n^2)$
2) Inplace algo
3) Stable
4) Online
5) Adaptive (depends)
6) No of comp $\Theta(n^2)$
7) No of swap $\Theta(n^2)$

★ Selection Sort:

64 25 12 22 (11)

11 25 (12) 22 64

11 12 25 (22) 64

11 12 22 (25) 64

11 12 22 25 64

★ Sorted subarray

★ Remaining unsorted subarray

★ min element from unsorted subarray moved to sorted subarray.

1) $O(n^2)$
2) Inplace algo
3) Stable
4) Adaptive
5) No. of Swaps $O(n)$
6) No. of Comp $O(n^2)$

★ To Prove correctness of a Algorithm:

① Initialization:

The invarient is true prior to the first iteration of the loop

② Maintenance:

If invarient is true for iteration n, it is true for iteration n+1.

③ Termination:

When the loop terminates the invarient is true on the entire input.

★ For Bubble Sort

→ Invarient is:

At iteration i, the subarray A[1..i] is sorted and any element in A[i+1...A.size()] is greater or equal to any element in A[1...i]

↑ sorting in descending order

✱ Avg Case Analysis:

1) Hiring Problem

```
best ← 0
for i ← 0 to n
    Interview cost i .. ci
    if i has a better score
    best = i
    hire i .... ch
```

Prob (ith candidate best among 1 ... i) = $1/i$

X : hiring cost for 1 ... n

X_i : cost in hiring ith candidate

$$X = \sum_{i=1}^{n} X_i$$

$$\Rightarrow E[X] = E\left[\sum_{i=1}^{n} X_i\right] = \sum_{i=1}^{n} E[X_i]$$

$$E[X_i] = Pr(\text{hire cand } i) * c_h + Pr(\text{not hire}) * 0$$

$$= \frac{c_h}{i}$$

$$\therefore E[X] = c_h \sum_{i=1}^{n} 1/i$$

$$1 + \underbrace{\frac{1}{2} + \frac{1}{3}} + \underbrace{\frac{1}{4} + \frac{1}{5} + \cdots + \frac{1}{7}} \leq \sum_{i=0}^{\lfloor \log n \rfloor} \sum_{j=0}^{2^i - 1} \frac{1}{2^i + j}$$

$$\leq \sum_{i=0}^{\lfloor \log n \rfloor} \sum_{j=0}^{2^i - 1} 1/2^i$$

$$\leq \lg n + 1$$

∴ Avg case cost $O(c_i n + c_h \log n)$

2) Hiring Problem 2

best ← 0

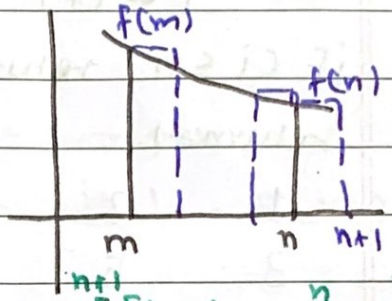
for i ← 0 to n

Interview ... c_i

if i has a better score

best = i

hire i ... c_h



$\int_m^{n+1} f(x)\,dx = \sum_{i=m}^{n} f(i)$

Stop if best $[1,k] < a[j]$ $j \in [k+1, n]$

Find apt k ∋ best guy is chosen

S: event in which we hire the best

S_i: i^{th} is the best in $[1,n]$ & algo picks i

$Pr(S_i)$ = Prob(Best in $[1,n]$) * Algo picks i

$= \frac{1}{n} \times \frac{k}{i-1}$

1 .... k | k+1 ... i-1 | i =

best was here

$\therefore P(S) = \sum_{i=k+1}^{n} P(S_i)$

$= \sum_{i=k+1}^{n} 1/n\,(k/i-1) = k/n \sum_{i=k}^{n-1} 1/i$

$\geq \frac{k}{n} \int_k^n \frac{1}{i}\,dx = \frac{k}{n}(\lg n - \lg k)$

$\frac{d}{dk}\left(\frac{k}{n}\lg n - \lg k\right) = 0 \Rightarrow k = \frac{n}{e}$

3) Linear Search $(x, a_1, \ldots, a_n)$

$i \leftarrow 1$

while $(i \leq n$ and $x \neq a_i)$ → 2i comp (# : $i \leq n$)

$i \leftarrow i+1$

if $(i \leq n)$ return i → 1 comp

return -1.

$(x=a_i)$ i	1	2	....	i	.....	n
Comp.	3	5		2i+1		2n+1

★ If x not in list (2n+2) comp # checked

$\xrightarrow{p}$ $a_1\ a_2 \ldots a_n$ ⟶ Not in list (1-p)

$$E[X_i] = \frac{p(2i+1)}{n}$$

$$\therefore E[X] = \sum_{i=1}^{n} E[X_i] + (2n+2)(1-p)$$

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

4) Avg no. of comparisons in insertion sort

$a_1 \ldots a_{i-1}$ | a_i

i slots

X: no. of comparisons

X_i: no. of comparisons to put at curr pos

$E[X] = \sum_{i=2}^{n} E[X_i]$ & $\sum_{k=1}^{i} \left(\frac{1}{i}\right) k = \frac{(i+1)}{2} = E[X_i]$

$\therefore E[X] = \sum_{i=2}^{n} \frac{(i+1)}{2} = O(n^2)$

5) No. of inversions in a permutation

$a[i] > a[j]$ inversion

$X_{ij} = \begin{cases} 1 & \text{(Inversion)} \\ 0 & \end{cases}$

$E[X_{ij}] = Pr(\langle i,j \rangle \text{ is an inversion in given permutation})$

$= \frac{1}{2}$

$\therefore X = \sum_{1 \le i < j \le n} X_{ij} \Rightarrow E[X] = \sum E[X_{ij}] = \frac{n(n-1)}{4}$

★ Randomised Algorithm:

→ Whatever we did previously calculated the avg case running time (Prob. Deterministic Algo) where prob. distribution was over the inputs to the algorithm

→ In randomised algo we disscuss the expected running time when algo itself makes random choices.
A random permutation of input is fed to the algo

→ The only change we make in hiring problems is we permute the inputs randomly

Las Vegas Algo — In HP1 : Always produces the best output

Monte Carlo Algo — In HP2: May not produce correct output

★ Random Permutation of input

1 n = A.length

2 for i = 1 to n

swap A[i] with A[rand(i,n)]

★ rand(i,j) Prob. of generating a particular no. between i & j is $\frac{1}{j-i+1}$

★ Loop invariant:

Just prior to ith iteration of for loop for each possible $(i-1)^{th}$ permutation of the n elements the subarray A[1...i-1] contains this (i-1) permutation with prob. $(n-i+1)!/n!$

★ Use Initialization i=1

Maintainence i=i (Induction)

Termination i=n+1

Pr(A) = 1/n! each permutation equally likely