*p *q
p = q

valid pointers/operations : DS & Algo :-

(a) assignment of pointers of the same type (for diff types) x = 10 (8080 → 0x16ab … in hexadecimal form)

(b) adding or sub ptr and int → p+1

(c) subtracting or comparing two ptrs to mem of same array → q-p+1 (prints elements p to q)

cout &x ; // print address

(d) assign or compare store address to zero → (p == 0)

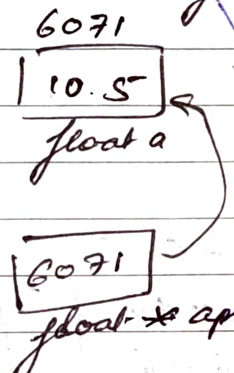
(y only int y written then will become integer but once declared only y written means addr storing made)

declare int *y = &x ; // declare & initialize

int *y // declare

y = &x ; // assignment

(for later use * for addr it will take it as val)



float a = 10.5 ;

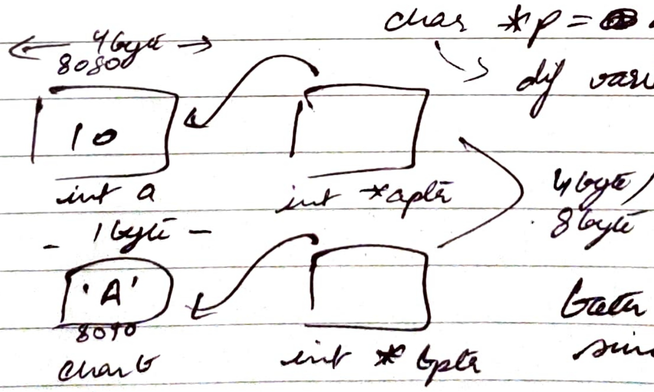
float *aptr = &a ;

double * ptr2 ; → garbage val

double * ptr2 = 0 ; // Null pointer

int a = 0

char *p = &a ; // don't do that → diff variable



both same size since both are addresses

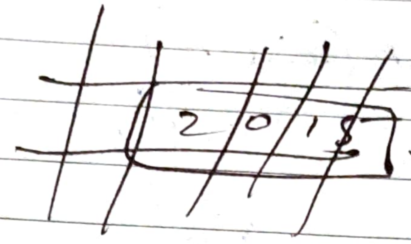
int x = 10;

int &y = x ;

size of int, float etc

dereferencing : when you read val. from ptr

↳ if assign diff data type of ptr → it will read only size of that box



→ int ~~float~~ val (4 byte)

if char ptr it will read only first box (1 byte)

* → multiplication
* → pointer
* → ~~deferen~~ dereference any address (gets val from address)

& ← 5 & 3 (Between And)
& ← & x (Address of)
& ← & (reference var)

2054
| 10 | | 2054 |
x xptr

val ← address

cout << *(&x); = 10
cout << *(xptr); = 10
↓ is address

cout << *(xptr)+1 ; // = 11
cout << *(xptr+1); // garbage
→ but for array same means arr[1]
*(xptr + (1 × size of datatype)) if xptr pts to arr

struct Node (3 usg)
{
int i, j;
}

struct Node a, *p;
p = &a;
*p.i ; → acc to precedence it will read *(p.i) assuming p is instance of node but will give error
↓
(*p).i ;
↓ shortcut
p→i;

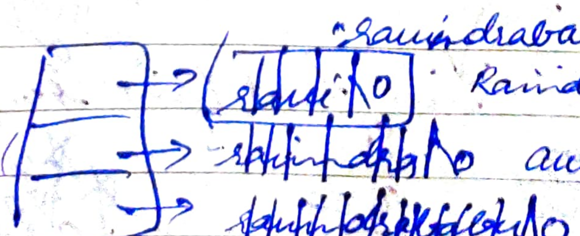
| 10 | | 20 | | 40 |

x
name is array of pointers of char datatype
int * xptr
int * * xxptr → syntax

→ double ptr : ptr to the ptr

array of pointers better than 2D array (base) ↳ less memory space

char *name[] = {"ravi", "ravindra", "ravindrababu"}



Ravindra = printf("%s", *(name+1))
avi = printf("%s", *name+1)

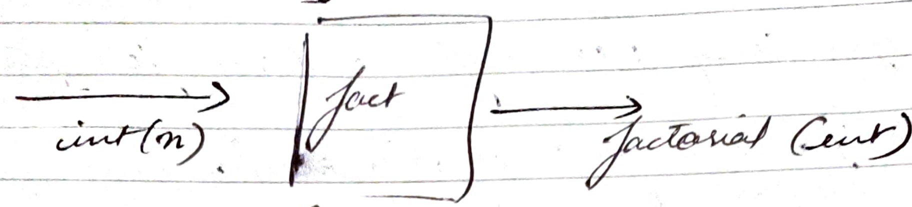
int (*a)[6]; pointer to array 6 ele — pure (int a[][6]
int *a[6]; array of ptr of int datatype

Extramarks

Date ___/___/___

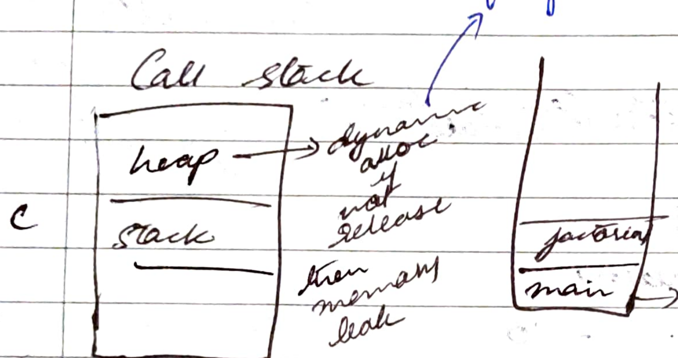
Page No. ______

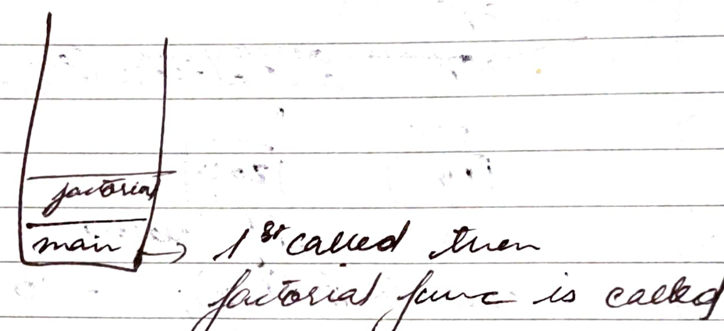
function




```
int fact (int n)
{
    ..
}
```

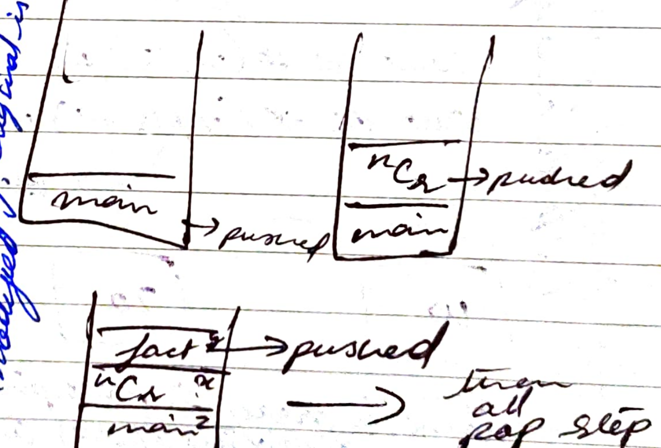
Call stack






```
fact ( ) {
    y = 0
}
nCr ( ) {
    fact()
    x = 0
}
main ( ) {
    nCr( )
    z = 0
}
```

→ here memory loc. of passed variable is same as parameter, so no copy, original is modified.



Call by Value & Call by reference (see prog.)

```
int fact(int n)
{
}
int main () {
    int n = 0;
    fact(n);
```

pass by pointer: mem loc of variable is passed as param

(P.T.O)

```
int fact (int *n)
{
}
int main () {
    int n = 0;
    fact(&n)
}
```

→ this actually pass by pointer (&n) = n ; can use another pointer instead of &

this is pass by reference:

```
int fact (int &n)
{
}
int main () {
    int n = 0
    fact(n);
}
```

fact(head)

Extramarks
Date: class to head public
Page No: fun a (Node *&p)

other dy b/w ref & ptr (* page *p)

When you pass ptr head, p will point at same loc as head but since pass by val any change in p won't affect head but any change in *p will affect val
↳ for ref pass (Node *&p)

?
3
?;
int main()
{ a(head);
?

Q1 ~~stare~~ 0 1 2 3 4 5 6 7 8 9 10 11 12 13
coding junction
(c) junction ✓

Q2 c ✓
Q3 c ✓
Q4 c ✗ → malloc : assign block of uninitialized memory to a pointer
Q5 (d)
Q6 b O24 ✓
Q7 (C) sum = sum + *(*(arr + i) + (i++));
Q8 (b) No ✓ → check why

[Arrays 1.0]

- Problem solving:

C_1 = 1 Rick or cab — 1 time
C_2 = 1 Rick or 1 cab — ∞ time
C_3 = all Rick or all cabs — ∞ time
C_4 = all Rick & all cabs — ∞ times

n = Rick, m = Cab

1	3	7	19
c_1	c_2	c_3	c_4

(2) Rick — 2 times R1, 5 times R2
(3) cabs → 4 cab_1, 4 cab_2, 4 cab_3

cost of cabs = 4C1 + 4C1 + 4C1
cost of cabs = min(4C1, C2) + min(4C1, C2) + min(4C1, C2)

min((1), C3) —— I

Extramarks

Date ___/___/___

Page No. ________

cost of rickshaws = min (2C1, C2) + min (5C1, C2)] — (3)

min ((3), C3) — II

min (C4, (I) + (II)) ⇒ final ans

- C4 ↓ every ride
- (I) ↓ C1, C2, C3 (rickshaws)
- (II) ↓ C1, C2, C3 (cabs)

in vectors (2d) you can push_back 1d vectors with for loop (ref: test.cpp in PT)

Ramdralabu

arrays

$a[i] = (a+i)$ (a = base) — means & $a[i]$ — if starting from 0

$*(a+i) = a[i]$

$a[i] = (a+(i-1))$ if starting from 1

$a[i] = (i-1) * size + base\ address$ (for 0 should be i)

Side notes: external starts with 0: a[0], a[0][0], a[0][0][0]; (−1) is already adjusted; 1 2 3, 1 2 3

2D array: m : n (rows : columns)

$a[i][j] = ((i*n)+j)*size + base$ — row major (starting from 0)

for indices 1: i−1, j−1

$loc\ A(i,j) = L_0 + [(i-L_1)(u_2-L_2+1)+(j-L_2)] * c$

$[L_1 \ldots u_1, L_2 \ldots u_2]$, c = cell size

Side working: ((11×10)+5)×4 = 115×4 = 460, +100 = 560

column major:

$a[i][j] = ((j \times m)+i)*size + base$

$loc\ A(i,j) = L_0 + [(j-L_2)(u_1-L_1+1)+(i-L_1)]*c$

In binary in

$a[i][j] = ((i \times n)+j)*size + base$ — $n = (2^k)$, base (2^x)

no. of rows = m = 2^l = l bits rep (i lies in m)

no. of cols = n = 2^k = k bits rep (j lies in n)

[ ] [0000...] [000...0]

l bits, k bits, x bits

+ [j] k bits

[l | k | 00...0]

l, k, x

for column major

[k | l | x]

k, l, x

Extramarks

Date ___/___/_____
Page No. _____

Asymptotic Notations

$f(n) = O(g(n))$

$0 \leq n^3+1 \leq n^4$

$n_0 = 2$

hence for $n_0 = 2$, it is ~~Big~~ n^3+1 is $O(n^4)$

$0 \leq n^3+1 \leq cn^3$ $\quad c = 2$, $n_0 = 2$

hence for $n_0 = 2$, ~~it is~~ n^3+1 is $O(n^3)$

Always give ans in big theta unless ~~asked~~ asked specifically for those

Q n^2+n+1

$O(n^3)$

$0 \leq n^2+n+1 \leq n^3$

$n_0 = 2$

for $n_0 = 2$, n^2+n+1 is $O(n^3)$

$\Omega(n^2)$

$0 \leq n^2 \leq n^2+n+1$

$n_0 = 0$

for $n_0 = 0$, n^2+n+1 is $\Omega(n^2)$

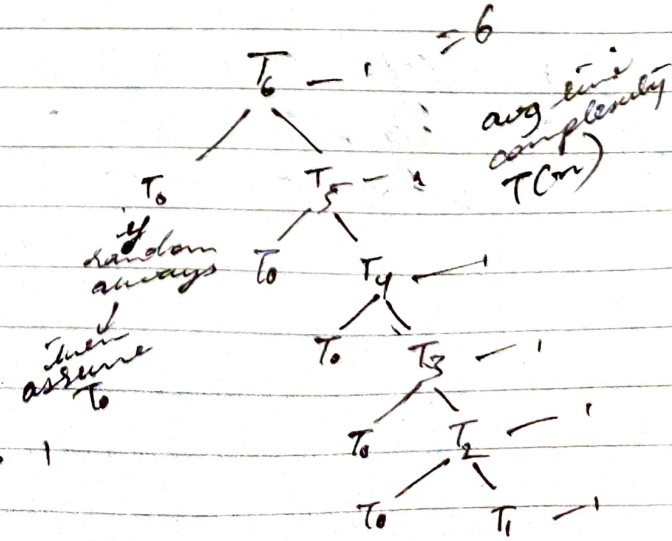
$\Theta(n^2)$

$0 \leq c_2 g(n) \leq f(n) \leq c_1 g(n)$

Q1 O(length) Q2 $O(n^2)$

Q3 (Time complexity)

```
int fun(int n)          for 6
{   int i;          ] k1 = 0
    if (n<=0)
    {   return 0;
    }
    else
    {   i = random(n-1);
        printf(" ");
        return fun(i) + fun(n-1-i);
    }
}
```



Extramarks

Date ___/___/_____

Page No. _____

Q4 (a) ✓ $O(n)$

(b) ✓ $O(n)$

(c) ✓ $O(n)$

(d) $O(N+M^2) \rightarrow O(M^2)$ ✗ hence not n

Q5 : Balanced BST (Binary search tree)

- visits all nodes so $O(N)$

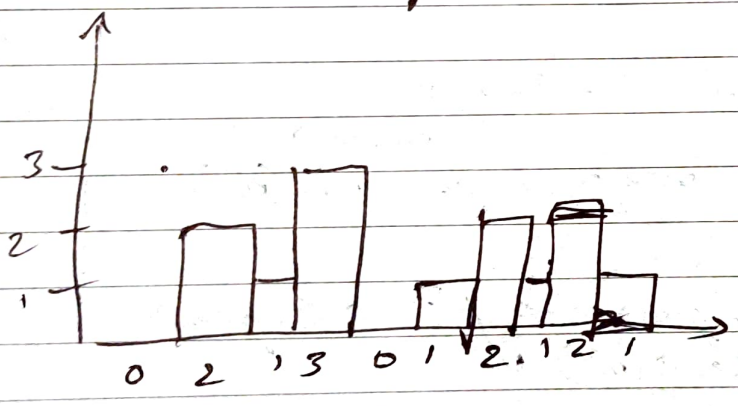
Q6 for $i^2 = (n-1)$

$i = (\sqrt{n})$

$O(\sqrt{n})$

Q7 $T_n = O(1)$

TLE (Prateek Narang) → Time limit exceeded



find the building

left 0 2 2 3 3 3 3 3 3 3

right 3 3 3 3 2 2 2 2 2 1

water 0 0 1 0 2 1 0 1 0 0 = 5 units of water

$\min(l(i), r(i))$ - ht of curr. building

Extramarks

Date ___/___/_____

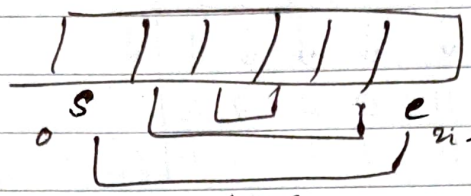
Page No. _____

"My sol"

Reverse array:

intuition

while s < e



time complexity = O(N/2)

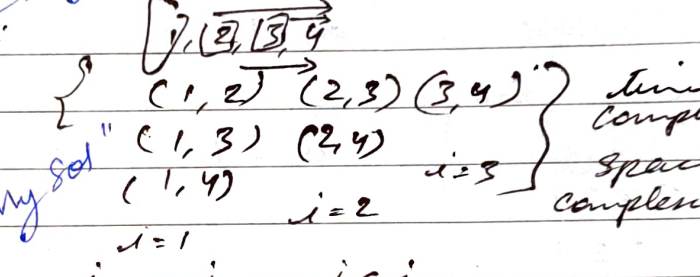
= O(N)

space complexity = O(1)

"My sol"

Print all pairs:

intuition



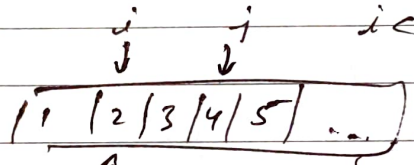
time complexity = O(N²)

space complexity = O(1)

"My sol"

Subarrays:

intuition



time complexity = O(N³)

space comp = O(1)

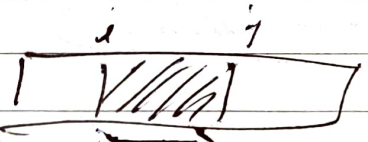
$\propto {}^{N}C_2$

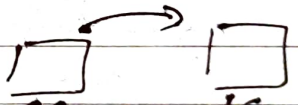
$\propto N^2$ subarrays

"My sol"

Largest sum subarray:

intuition





- Brute force

also calc sum

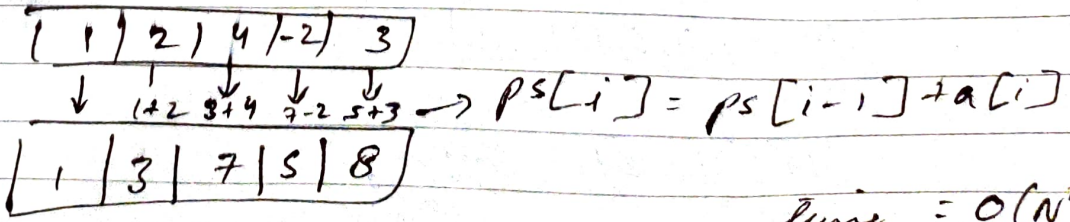
"My sol"

Here cumulative frequency

time complexity = O(N³)

space comp = O(1)

- Prefix sum



$ps[i] = ps[i-1] + a[i]$

time = O(N)

$ps[j] - ps[j-1]$ Complexity

space comp = O(1)

Q lowerbound – find val in arr closest val to assigned val
→ "My sol"

Q K rotate – right circular shift k time

→ "My sol"

Extramarks
Date
Page No.

Kadane's algo: Time complexity: O(N)

Space complexity: O(1)

	-2	3	4	-1	5	-12	6	1	3	2
if -ve take 0	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
CS (curr sum)	0	3	7	6	11	0	6	7	10	12
MS (max sum)	0	3	7	7	11	11	11	11	(11) highest	(12) → then this highest

2 → if add 2 in end

- Bubble sort intuition:

bigger bubble comes up first

Time complexity: O(N²) all cases

j=0 ——→ n-2

5 4 3 (2) 1	4 3 2 1	3 2 1	2 1
4 5 3 2 1	.	.	.
4 3 5 2 1	.	.	.
4 3 2 5 1			
4 3 2 1 5			
1st time			
a[j], a[j+1]			

4th time for 5 elements

n-1 times

```
for (times)
{ for (j=0 - n-1-times.)
    {
    }
}
```

- Insertion sort intuition

[5 | 7 | 8] [6] insert & shift

5, | 4, 1, 3, 2
unsorted

[5] [4, 5] | 1, 3, 2

↑ Take no. in temp, then if sorted ones greater then keep shifting till right place

[1, 4, 5] [3, 2]

[1 3 4 5] [2]

[1 2 3 4 5]

Time complexity: O(N²) avg & worst case
O(N) in best

Space comp.: O(1)

N= O(1) < logN < N < NlogN < N² < N³ < 2ᴺ < Nᴺ
1 |
2 |
3 |

Date / /
Page No.

Selection sort intuition:

[3, 2, 1, 5, 4]

find min & put it left side

current

[3] [2, 1, 5, 4] — Time complexity: O(N²) all cases

↑ swap with 3

[1] [2, 3, 5, 4]

↑ right place

[1, 2] [3, 5, 4]

↑ right place

[1, 2, 3] [5, 4]

current ↑ swap

[1, 2, 3 4]

n-1 elements sorted

Inbuilt sort intuition: Time complexity: O(nlogn) better than other 3

Counting sort intuition: arr [5 | 8 | 3 | 1 | 1 | 8] (1 1 3 5 6 8)

→ Range

count: | 0 | 2 | 0 | 1 | 0 | 1 | 1 | 0 | 1 |
0 1 2 3 4 5 6 7 8

index

now put val

Time complexity: O(N + Range)

Creating char array:

char a[100];

char a[100] = {'a', 'b', 'c'} //wrong

char a[100] = {'a', 'b', 'c', '\0'} //right

char a[100] = "abc";

a[100];
cin >> a; "Hello world"
Cout >> Hello

[hello world] → [hello | world]

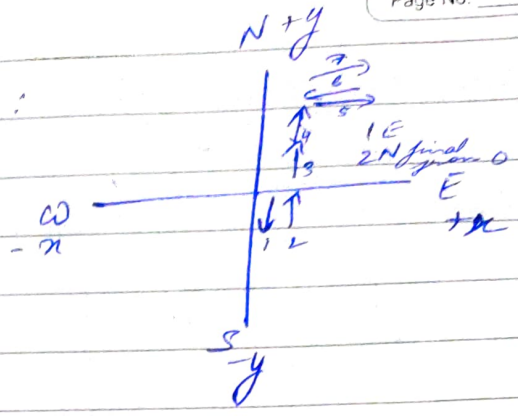
cin doesn't read whitespace

Extramarks

Date ___/___/_____

Page No. _____

Shortest path (String) :
intuition



Sample input :
SNNEWE

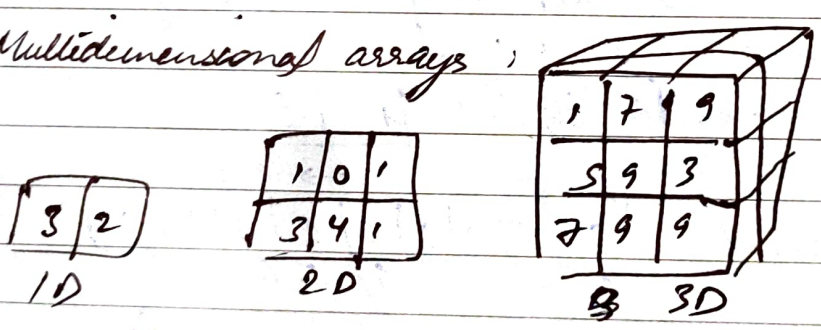
Sample output :
NE

- String (largest) intuition :
5, 6, 10, 8, 2
No B B 10 8 2
L 5 B 10

- String compression intuition : aaaBBccc
a3B2c3
aBc
a1B1c1
↳ in this case return original

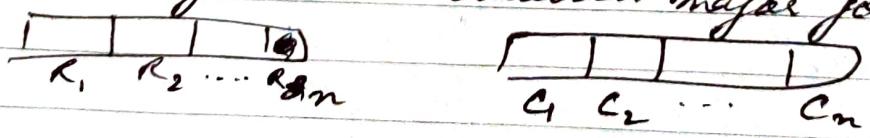
↳ O(N) time : same i being incremented

- Multidimensional arrays :

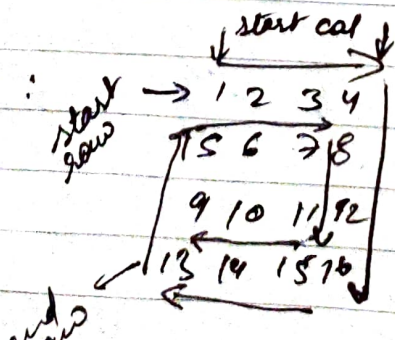


How is it stored ?

Row major & column major form



- Spiral print :



loop:
- startRow (startCol ... endCol)
- endCol (startRow+1 ... endRow)
- endRow (endCol-1 ... startCol)
- startCol (endRow-1 ... startRow+1)

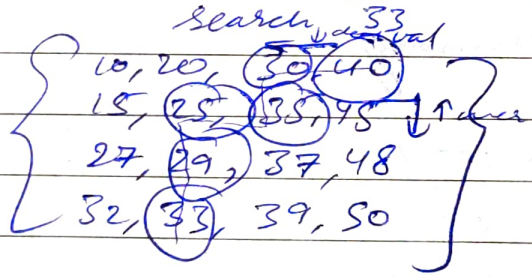
startRow++, endCol--, endRow--, startCol++

Output:
1, 2, 3, 4, 8, 12, 16, 15, 14, 13, 9, 5, 6, 7, 11, 10

Sorted Array search: intuition

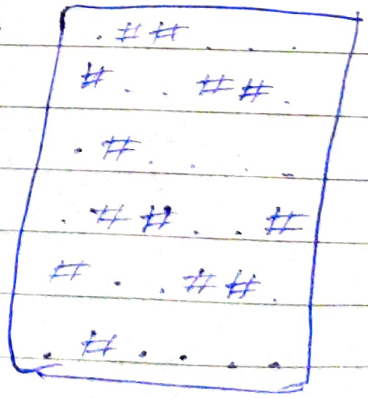
10, 20, 30, 40
15, 25, 35, 45
27, 29, 37, 48
32, 33, 39, 50

- Brute force (iterate all) : $O(N \times M)$
- Binary search : $O(N \log M)$
- Staircase search :



$O(N+M)$

Ramu & Mango trees : intuition — max kis min



- Brute force : cut at every pt & then count

Make cut at every i, j $O(N^2)$

compute
- S1
- S2
- S3
- S4

$\Rightarrow$ (min) $O(N^2)$

Max (x)

$O(N^2 \cdot N^2)$

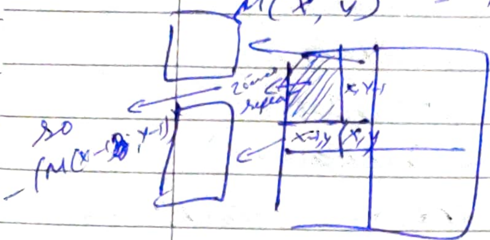
$O(N^4)$

for N = 100
$(10^2)^4 = 10^8$ = 1 sec

for N = 1000
$(10^3)^4 = 10^{12} \Rightarrow$ TLE

M(x, y) = 1 + M(x-1, y) + M(x, y-1) - M(x-1, y-1) (if tree)

= 0 + " (if no tree)



0	1	1	0	0	0
1	0	0	1	1	0
0	1	0	0	0	0
0	1	1	0	0	1
1	0	0	1	1	0
0	1	0	0	0	0

Second part

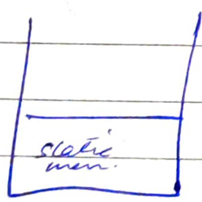
S_1 = M(x, y)
S_2 = M(x, N) - M(x, y) (last col.)

S_3 = M(N, y) - M(x, y)
S_4 = M[N, N] - (S_1 + S_2 + S_3)

do pointers from earlier notes

Static memory

{
int a[100];
int b, c;
}



Compiler creates & destroy

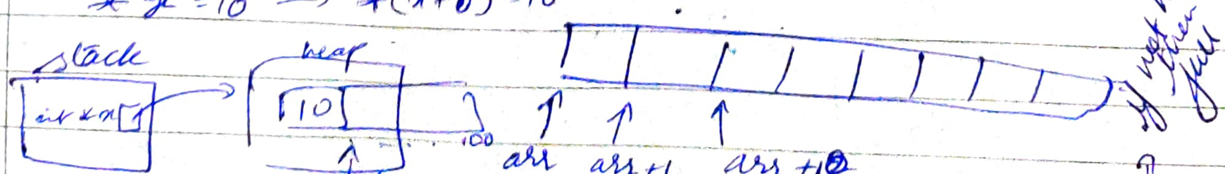
Dynamic memory
↓
created during runtime

int n;
cin >> n;
int * a = new int [n];

↓ static alloc
↓ dynamic alloc goes into heap memory

job of programmer to delete it

int *x = new int [100]
*x = 10 → *(x+0) = 10

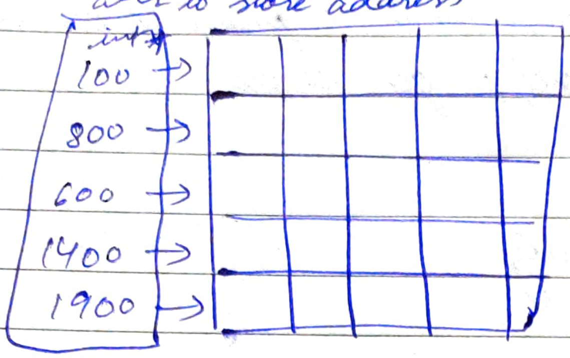


delete [ ] x; deletes this memory → if change address or doesn't delete → even memory leak

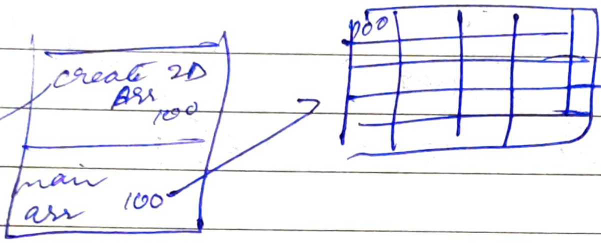
arr[2] = *(arr + 2);

2D Dynamic array intuition:

arr to store address



combine multiple 1D arrays together



created arr & returned address so when even func destroyed, memory can still be accessed

Vectors: ⟶ Part of STL ⟶ DS (data structure)

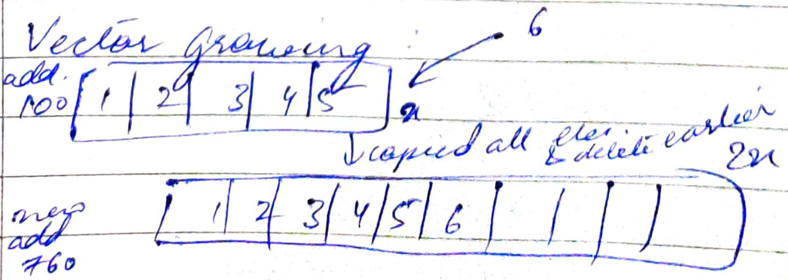
Standard Template libraries

2 ques ⟶ How? fundamentals, how to implement

⟶ Use: when asked directly in problem no need to implement just use from STL

- Vectors: - seq. containers representing arrays that can change size
- continuous storage location so accessed in O(1) time (like array)
- size can change dynamically (unlike arrays)

Vector growing:



~~Fast~~ Pushback (data) O(1) but in case of more than size O(N) since create new & copy paste

Average time = O(1)

Extramarks

Date ___/___/_____

Page No. _____

functions of vector: push_back : insert ele at last place

pop_back : pop out ele from last place

get[i] : get ith ele

→ create → front, back, size, capacity

const is used if val of data mem is not modified

all present in doc. accessed through C++ STL

Inbuilt ~~find~~ : arr.begin() ... arr.end()

vector<int> arr = {11, 12, 13, 2, 4, 6, 8};

Inbuilt ~~search~~ : finds a subsequence

Inbuilt binarysearch

Operators:

1. Binary AND &
2. Binary OR |
3. Binary XOR ^
4. Binary One's complement ~
5. Binary Left shift <<
6. Binary Right shift >>

(1)

0&0 = 0

1&0 = 0

0&1 = 0

1&1 = 1

5 & 7 = 5

0 0 0 0 1 0 1

0 0 0 0 1 1 1

" 1 0 1

(2)

1|0 = 1

0|1 = 1

1|1 = 1

0|0 = 0

5 | 8

0 1 0 1

1 0 0 0

1 1 0 1

$2^3\ 2^2\ 2^1\ 2^0$

8+4+0+1 = 13

Extramarks

Date __/__/____

Page No. _____

(3) Exclusive OR

0 ^ 0 = 0

1 ^ 0 = 1

0 ^ 1 = 1

1 ^ 1 = 0

5 ^ 7

000101

000111

000010 = 2

(4) ~ 0 = 1

~ 5

~ 000101

111010

also flips most signi bit so can be -ve

int a = 0;

cout << (~a); → -1

since integer zero so 32 bit 00000000000

11111111111111

sign bit

2's complement

(5) BLS (multiply)

5 << 2

(left shift by 2)

5 × 2^2 = 20

00000101

last off

00101100

a << b

= a * 2^b

(6) BRS (divide)

10 >> 1

(right shift by 1)

000101 = 5

000010 = 5/2 = 2 (lower bound)

a >> b

= a / 2^b

Bit Manipulation

→ Bitwise add even intuition:

1 000001
2 000010 > all add last bit 1
3 000011
4 000100

so AND with 000000001
if last bit 1 then odd else even

→ Set ith bit:
intuition
take 000001
shift left i times
then AND
→ make it 0 if already 0 then no change

→ Clear ith bit:
i ← ith bit
1 1 0 1
& 1 0 1 1 ← to get here first BLS i
1 0 0 1
this will preserve other bits

1 << i
0 1 0 0
then complement
1 0 1 1
then AND it

→ make it 1 if already 1 then no change

→ Set ith bit:

→ Update ith bit:
↳ call clear
~~set ith bit~~
then OR with
val (pass val as 1 if need 1 or 0 if need 0)

if 1 then set to 0
1 1 1 1 0 1 0 0 1
if 0 then set to 1
1 1 1 1 0 1 0 0
clear bit
OR 0 0 0 0 0 0 0 0
1 1 1 1 0 0 0 0

→ clear last i bits
1 1 1 1 0 1 0 1
4 3 2 1 0
1 1 1 0 0 0 0 0

↳ done by: create mask where upto ith bit 0 then all 1s

mask → 1111 0000
& 1011 1010
1011 0000

do left shift of -1 with i
or
~0

→ clear range of bits

1110 1010 1111 0
(i ... j)
1111 1 000000 1 ← mask needed
(-1<<j)

2 parts : a 1111 1000000000
b 0000000000001
2^i bit

$2^i = 1<<i$

$2^i - 1 = (1<<i) - 1$

m = a|b
mask
n = n & mask

→ replace bits

N = 1000 00000 00
7 6 5 4 3 2 1 0
i ... j
clear range of bits

mask = 1010 100
with
i shift

→ Power of 2 :

N = 16 = 2^4 = 10000
N-1 = 15 = 01111
00000

N = 8 = 2^3 1000
N-1 = 7 & 0111
0000

hence if (N&(N-1)) == 0
N is power of 2

Extramarks

Date ___/___/___

Page No.

→ Count bits: intuition

1001
& 1

Time complexity = logN

16 = 10000 (5 total bits)

7 = 111 (3 bits)

count += (n & 1)

n = n >> 1 (right shift till no bits left)

→ Count bits (hack): intuition

do

n = n & (n-1)

while

n > 0, loops till no. of bits

n = 9 : 1001 & 1000 = 1000

n-1 = 8

n-2 = 7 : 0111 & 0000

2 bits

n = 3 : 011 & 010

n-1 = 2

n-2 = 1 : 001 & 000

2 bits

→ fast exponentiation:

Time : logN+1 (a^4 a^2 a^1 / / a^1)

normal : O(N) time on exponentiation

$a^N = 3^5 = 3^{101}$

ans = 1 × a × a^4 = $a^4 \cdot a^1$

power_a = a, a^2, a^4 = a^5

bit = 1, 0, 1 = 3^5

= 243

→ Convert to Binary

9 = 1001

n = n >> 1 (keep right shifting)

ans = $10^0 \times 1 + 10^1 \times 0 + 10^2 \times 0 + 10^3 \times 1$

= $10^3 + 1$ = 1000 + 1 = 1001 (Binary rep)

p = 1 10 10^2 10^3

★ Static int - executed only once in recursion

Extramarks

Date ___/___/_____

Page No. _____

Recursion

Technique where

Solution to a problem, depends on solution to smaller instances of same problem

→ Factorial:

$5! = 5 \times 4! = 120$

$4 \times 3! = 24$

$3 \times 2! = 6$

$2 \times 1! = 2$

$1 \times 0! = 1$

$0! = 1$

$N = 5$

$f(N) = N!$

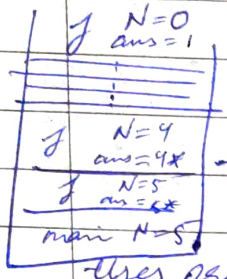
$= N * (N-1)!$

$= N * f(N-1)$

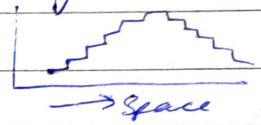
Time Complexity $= O(N)$

Space complexity $= O(N)$

Call stack



→ Recursion takes extra space



Uses principle of Mathematical Induction

1. Figure out smallest case
2. Always assume subproblem can be solved
3. Solve current problem assuming subproblem sol exists

→ Fibonacci:

0, 1, 1, 2, 3, 5, 8, 13 ....

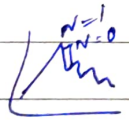
$f(N) = f(N-1) + f(N-2)$

Base cases = 0, & 1

$f(0) = 0, f(1) = 1$

$N = 4$

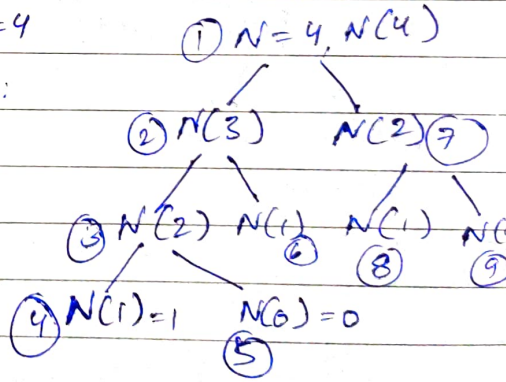
Call stack:



Time $= O(2^N)$

Space $= O(N)$

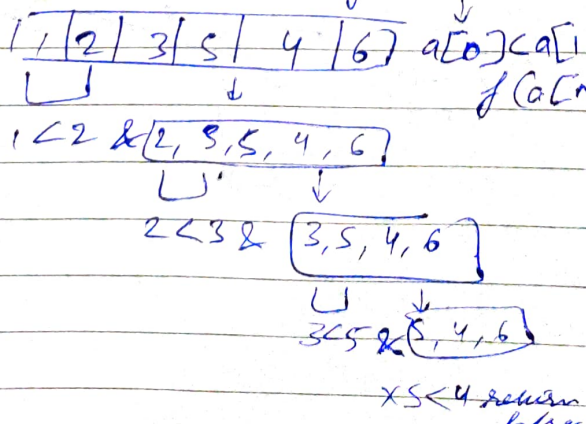
Recursion Tree

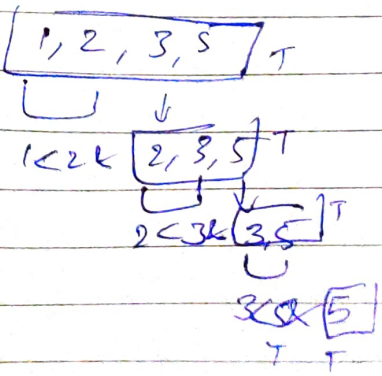


→ Check sorted:

f(a[N])

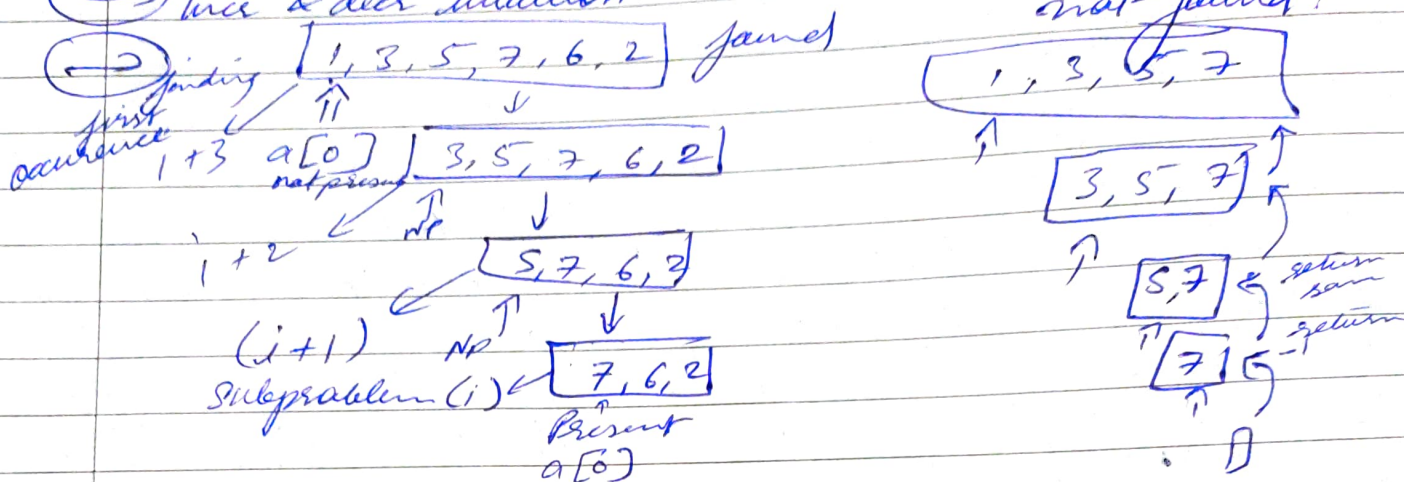
a[0]<a[1] & f(a[N-1])



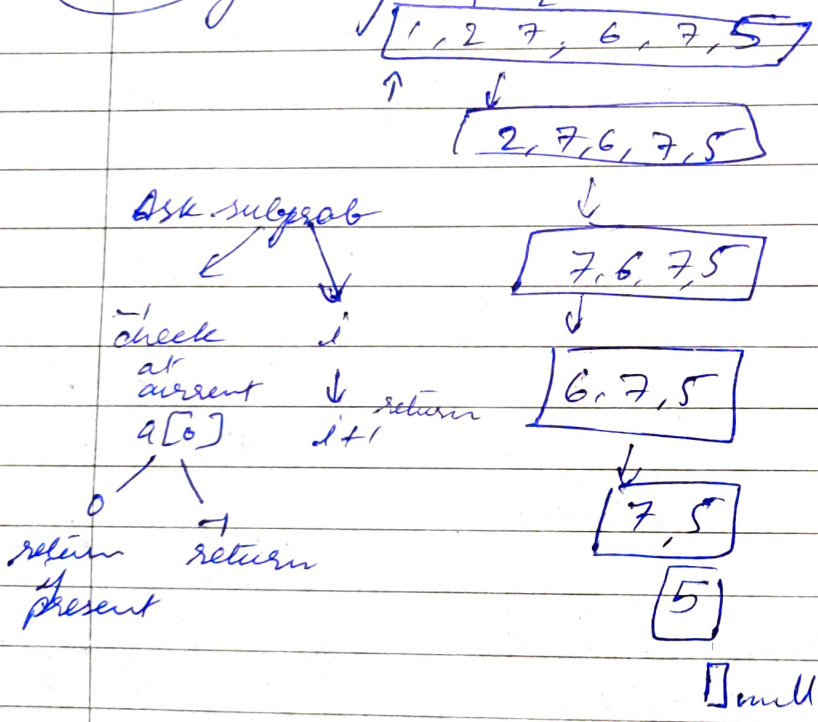


→ Incr & decr intuition:

→ finding first occurrence



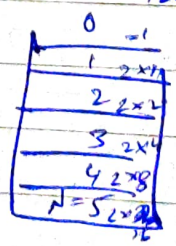
→ finding last occurrence intuition:



→ Power function intuition: Time: O(N)

(I) $a^N = a \cdot a^{N-1}$ Space: O(N)

subprob N=5



→ Optimised power intuition :

$a^N = (a^{N/2})^2$ — N is even

$= a(a^{N/2})^2$ — N is odd

N ↓ N/2 ↓ N/4 ... 1, 0

$2^7 = 2(2^3)^2$

$2^6 = (2^3)^2$

Time : $\log_2 N$

2^{10}

$2(2^5)^2$

$2(2^2)^2$

$2(2^1)^2 =$

$2(2^0)^2 = 2$

Iterative codes faster than recursive

→ Bubble sort recursive

reduce array size recursively : bubble_sort (a, n-1, 0) ← resets

run j from 0 to n-1 : bubble_sort(a, n, j+1);

→ Number spell intuition :

2021

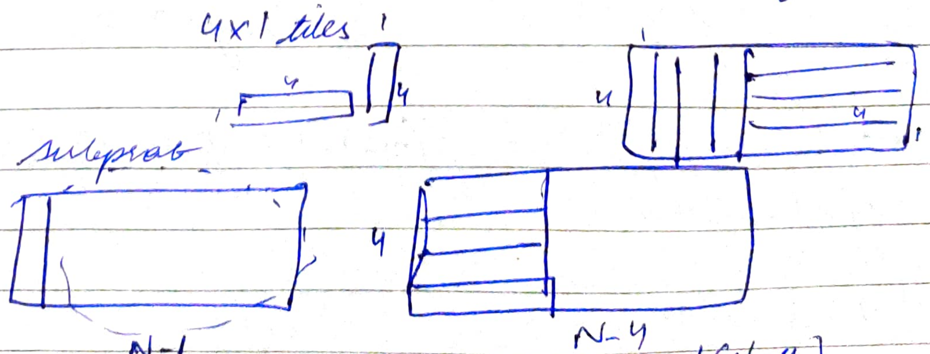
↓ % 10	↓ n/10
1	202
2	20
0	2
2	0

n = n/10

f(n) → f(n/10)

→ Tiling problem : 4 x N board → how to fill (board size will always be given)

4 x 1 tiles



$f(N) = f(N-1) + f(N-4)$

Extramarks

Date ___/___/___

Page No. ___

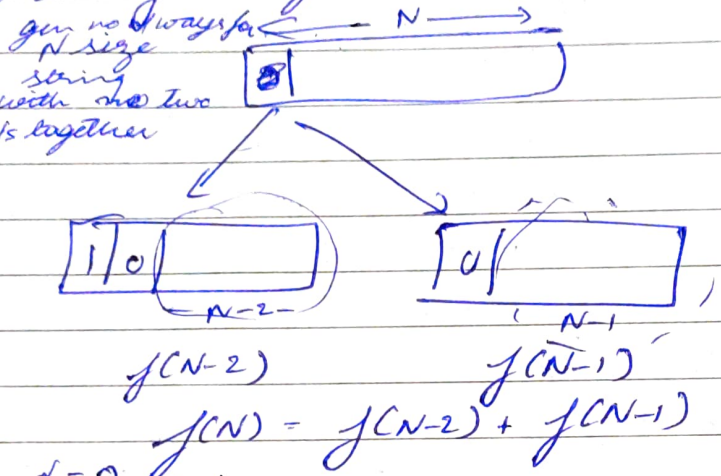
if $N <= 3$:
return 1 (way)

$N=0$ 1 way
$N=1$ 1 way
$N=2$ 1 way
$N=3$ 1 way

eg:- for $N=4$

$$f(N) = f(N-1) + f(N-4)$$
$$= 1 + 1 = 2 \text{ ways}$$

→ Binary strings intuition:



$$f(N) = f(N-2) + f(N-1)$$

$N=0 \rightarrow 1$

$N=1 \rightarrow 2$

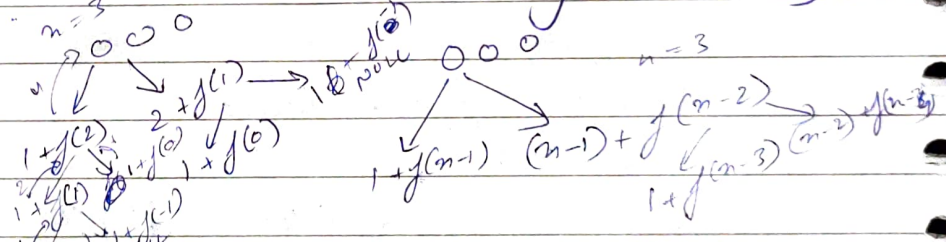
$N=2 \rightarrow 3$

$N=3 \rightarrow 5$

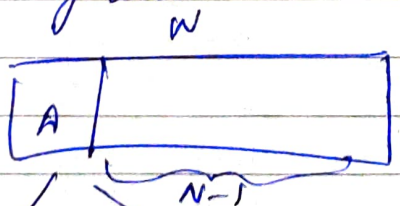
$N=4 \rightarrow 8$

Time = ~~O~~ $O(2^N)$

using DP = $O(N)$

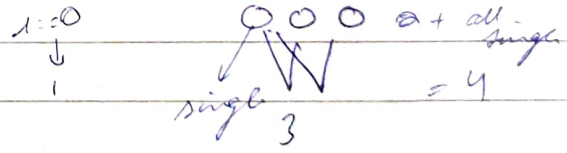


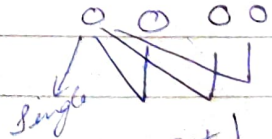
→ Friends party intuition:



How many ways N no. of friends can pair up

goes alone: $1 \cdot f(n-1)$ OR goes in pair: $(n-1) \cdot f(n-2)$ ($^{n-1}C_1$)





→ Approach: Mathematical eq:

$i = 0$ to 2

$dp[i] = dp[i-1] + (i-1) * dp[i-2]$

$i >= 2$: $dp[i] = dp[i-1] + (i-1) * dp[i-2]$

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

$N=0 \rightarrow 1$ way

$N=1 \rightarrow 1$ way

$N=2 \rightarrow 2$

$N=3 \rightarrow 4$

Extramarks
Date __ / __ / ____
Page No. ______

Divide & conquer :

→ Merge sort :
① Divide
② Merge sort (left) and Merge sort (right)
③ Merge (left + Right)

10, 5, 2, 0, 7, 6, 4

10, 5, 2 left → 2, 5, 10

0, 7, 6, 4 right → 0, 4, 6, 7

→ depth first manner

why: because O(n) work done at each level & log n levels → log n times

Time complexity : O(n log n) all cases

Space complexity : O(n)

i ptr, j ptr: compare & put, then copy remaining

0, 2, 4, 5, 6, 7, 10 — k ptr, temp

→ Quick sort : 10, 5, 2, 0, 7, 6, 4
① choose a pivot - last element
② Partition
③ Rec. sort - quicksort

2, 0 (<, left) | 4 (correct pos.) | 5, 10, 7, 6 (>, right)

worst case $O(n^2)$

unstable because swap ele. acc. to pivots pos w/o considering actual pos

base case : 0, 1 then return

10, 5, 2, 0, 7, 6, 4
i, ↑ pivot
j: run from 0 to pivot

[< i | | > j]

10 > 4 only j++
5 > 4 ″
2 < 4 i++, then swap with j then j++

finally i++ swap with pivot ele.
then call QS on both partitions

2 ⓪ 4 5 7 6 ⑩

0 2 5 ⑥ 10

5 6 7

Extramarks

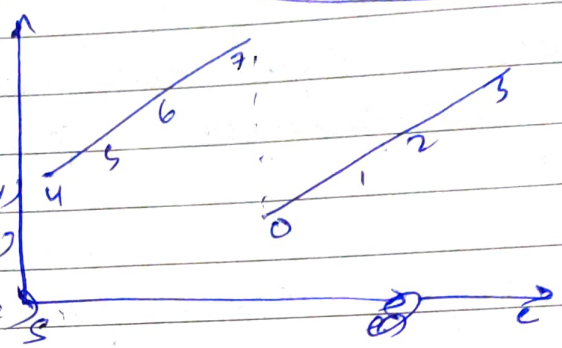
Date ___/___/_____

Page No. _____

Rotated array search intuition :

[4, 5, 6, 7, 0, 1, 2, 3] ⟸ 0, 1, 2, 3, 4, 5, 6, 7

k = 6



(1) a[s] <= a[mid]

↳ a(s) <= key < a(mid)

left (s ... mid)

↳ right (mid+1 ... e)

(1) a[mid] <= a[e] → (2)

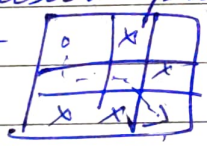
↳ a(s) ≤ key ≤ a(e)

↳ right (mid + 1 ... e)

↳ left (s ... mid - 1)

Backtracking :-

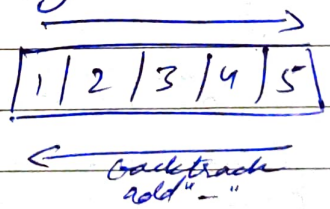
1. Decision prob : feasible sol

eg :-  → not possible w/o diagonal mov

2. Optimisation prob :- find best sol

3. Enumeration prob :- find all solutions

⟶ Array filling



print -1, -2, -3, -4, -5

All STL are passed by values, so ~~use~~ pass by ref explicitly (&args)
(eg vector)

⟶ Finding subsets

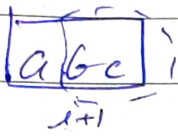
Input abc

path a or ∅ b or ∅ c or ∅

2 × 2 × 2

= 8

"", a, b, c, ab, ac, bc, abc



Take a [bc]

b → ab

c → ac

bc → abc

" " → a

Don't take a [bc]

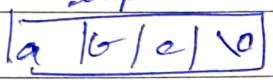
b

c

bc

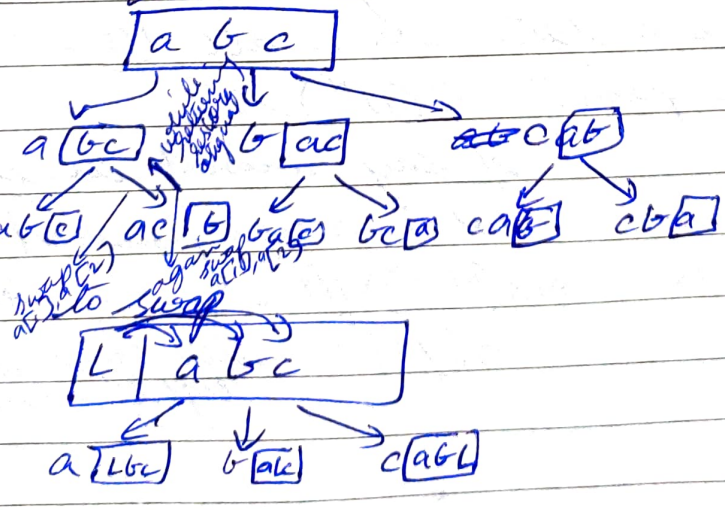
" "

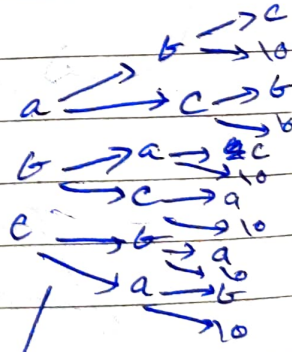
input



Recursion tree output

→ Permutation intuition :





(myself) working

Extramarks

Date ___/___/_____

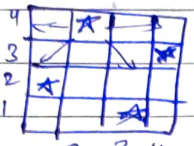
Page No. _____

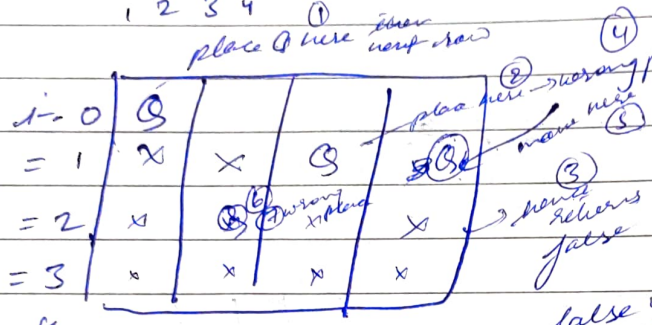
for odd
$i <= \lceil \frac{n}{2} \rceil$ place at odd
$i > \lceil \frac{n}{2} \rceil$ place at even

my own research:
for even $i < N/2$ row
place at even pos
$i > N/2$ row
place at odd pos

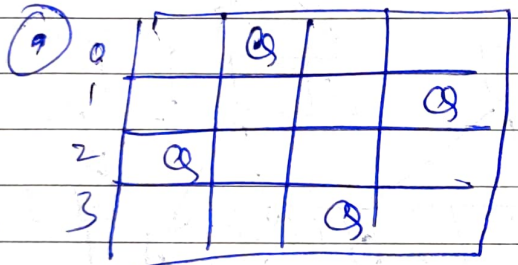
→ N-Queen's prob.

N×N chess board, N queens : no two Q attack each other
move : row, column, diagonal





(8) false warning goes back



4 i == N ⇒ hence complete

→ sol for both "print 1 config"
"print all config"

↳ in count return 1 if this case case & try for all cases, & later sum all 1's

Time complexity $O(N^N)$

let N=4

R_1 pos 1 .... pos 4 then

R_2 pos 1

R_3 pos 1

R_4 pos 1 (max depth)

4 × 4 × 4 × 4

4^4 = degree N^N

if say, doesn't check for col where already placed queen

$N \times (N-1) \times (N-2) \times \ldots \times 1$

$O(N!)$ still exponential but lighter bound

Extramarks

Date ___ / ___ / ______

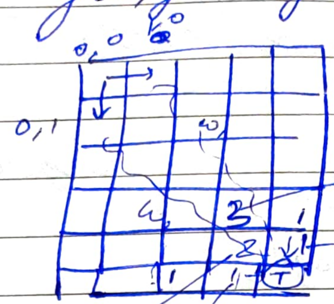
Page No. __________

→ Grid path : $f(x,y) = f(x+1,y) + f(x,y+1)$

(M×N) grid

can only go down & right

- find no. of ways



3×3

6	3	1
3	2	1
1	1	1

r_1 pos 1

↓

r_2 pos 1

↓

r_3 pos

r_3 pos 2

↓

r_4 pos 3

↓

r_4 pos 3

6 calls = m+n recursively

$2^{(m+n)}$

To calc time, just calc max depth

Alt sol: $\frac{(m-1+n-1)!}{(m-1)!\,(n-1)!}$ ← no of ways

O(m+n)

→ Sudoku :

intuition

for (no 1 to 9)

→ check if no can be placed at i, j (row, col., subgrid)

→ place it

a[i][j] = no $\left(3 \times \frac{N_x}{3}, \frac{N_y \times 3}{3}\right)$ → subgrid start pt

→ solve (i, j+1)

→ try next if solve returns false

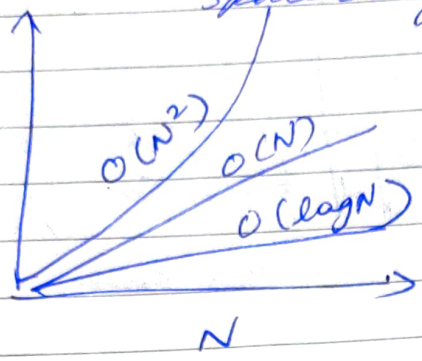
→ return true

if all no. placed still false then a[i][j] = 0 return false

Date ___/___/___
Page No. ___

Space time complexity

Theoretical analysis ← Time $\propto f(N)$ input → quadratic, linear, exponential, const etc
Space $\propto f(N)$ input



ignore constants

Experimental analysis → using clock

also refer code with variety

Big O notation:

Time = $an^2 + bn + c$ → take highest power, ignore constants

$O(N^2)$

worst case (KN^2)

$O(1) < O(\log N) < O(N \log N) < O(N^2) < O(2^N)$

Nested loops:

```
for (i=0; i<=n-1; i++)
{   for (j=i+1; j<=n-1; j++)
    {   const work } K
    }
}
```

i=0 j run N times
i=1 N-1
:
i=N-1 0

$T = K(N + N-1 \dots 1)$
$= K \sum N$
$= K \cdot \frac{N(N+1)}{2}$
$= \frac{KN^2}{2} + \frac{KN}{2}$ highest power, ignore const
$= O(N^2)$

```
for (i=0; i<=n-1; i=i+j) {
    for (j=i+1; j<=k; j++) {const work
    }
}
```

i=0 j=1
=1 2 3
3 4
4 8
15

j runs to k

so $i = i + k$

let N=30
k=3

1 3 6 9 30

i runs $\frac{N}{k}$ times
j runs k times worst case

so $O(\frac{N}{k} \cdot k)$

Stable algo: If 2 values are same they have relative order is maintained
↳ Merge, insertion, bubble, BT sort → stable (4,5) (4,3) → unstable (4,3) (4,5)
unstable: quick, heap, selection
↳ here each element is put in right place one by one and others are swapped out of place

Bubble sort best case:

1, 2, 3, 4, 5 O(N)

i runs 0 times
j runs N times

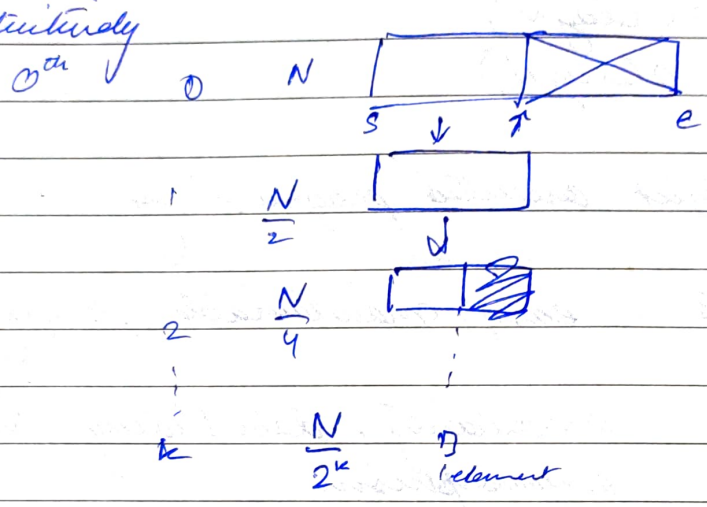
5, 4, 3, 2, 1 O(N²)

i runs n − 2 times
j runs N, N−1, N−2 ... 1 = ΣN

$$\frac{N(N+1)}{2} = N^2$$

Binary search:

Intuitively

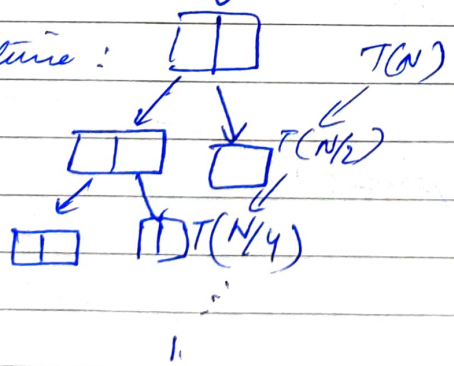


$$\frac{N}{2^k} = 1$$

$$N = 2^k$$

$$k = \log_2 N \qquad O(\log_2 N)$$

Merge sort time:



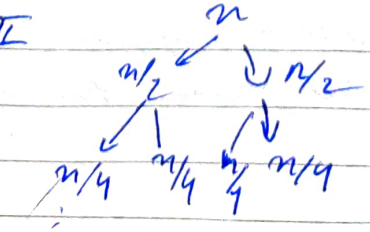
~~$O(\log_2 N)$ time~~

$O(N \log_2 N)$ time

Recursion time call: total no. of calls × work in each call

half power rule — II

Space = log N



$N \times \frac{N}{2} \times \frac{N}{4} \ldots 1$

O(N)

power - III

$T(N) = K + T(N/2)$

$T(N/2) = K + T(N/4)$

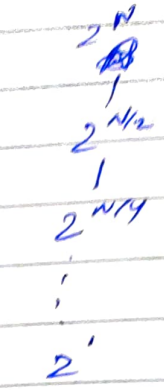
$T(1) = K + \dots$

$T(N) = \sum K$

$= K \log N$

Time $= O(\log N)$

Space $= O(\log N)$



OOPs : Object oriented programming

↳ Objects : easy, maintainable, encapsulation (properties, security, not accessible by other classes), modularity, data hiding, inheritance, polymorphism

```
class car {
    int a, b, c; //private
public :
    void sum() {

    }
    void update_price() {
    }
};
```

access modifiers

constructor :

- constructor Name = class name
- called automatically when an obj is created
- memory allocation happens when constr called
- constr called once for each obj

Extramarks

Date ___/___/_____
Page No. _____

→ pointer → points to property of ~~class~~ obj

(*this).x

this → x

class x {
int x;

eg:- fn(int x) {
this → x = x
(*mean obj's x is eq to x passed as param.
}
};

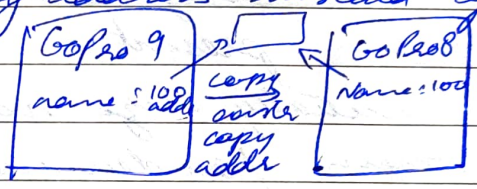
Constructor overloading : Constructors with same name but different parameters

- default cons (1)
- parameterized cons (2)

Copy constructor (3)

- Creates a copy of given object
- initialise an object using another object of same class

Shallow ~~Deep~~ copy : when copy address instead of memory



Deep copy : create new memory instead of copying addr.
: done by alloc new mem in copy ~~&~~ constr using "new"

- When to create user defined copy constr ?
- if an obj ~~has~~ has ptr to dynamically alloc obj & deep copy of obj is needed.

2. Copy assignment operator :
called when already ~~assigned~~ initialized obj is assigned a new val from another existing obj
- by default creates shallow copy like copy constr
- custom : void operator = (classname &x) {
}

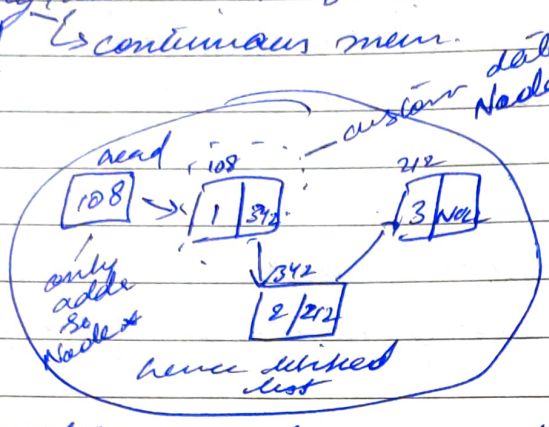
- Destructor :- automatically called when obj is destroyed as goes out of scope, destroyed explicitly by delete
- Compiler provides a default destructor
- User defined destructor needed when class contains ptr to dynamically alloc → otherwise cause memory leak

Linked lists : 1 ⇒ 2 ⇒ 3 ⇒ 4 ⇒ 5

↳ Memory is created on the fly (dynamically allocated mem.)

Array/Vector - dynamic size → shifts data to bigger area.

↳ fixed size ↳ continuous mem.



Best when : - large chunk of continuous mem. not present

- when initially size not known or need dynamic allocate mem. / on demand.

Operations :

- Insertion at head
  - tail
  - middle

at pos means 0, 3rd pos, is 4th element

- Search - linear : iterative, rec.
- Deletion at head
  - tail
  - middle

using this

- Create LL
- Delete LL

using this

- print LL

Code : → function : head present in main

→ OOP(class) : head in class

- STL : list

class node → data

→ address of next node (Node* ptr)

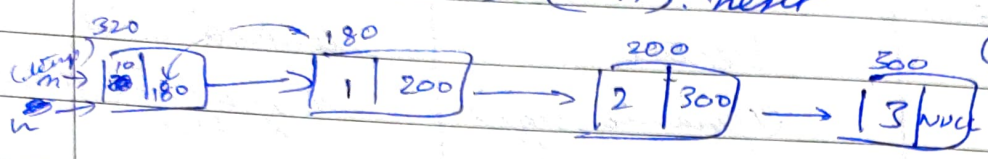
class list → w. head, tail

→ func

in push_front if create static instead of dynamic var (new Node) then it's scope will be limited & would be destroyed after func. end. but in dyn. user has to free mem. / delete var.

- ~~*n is local var but~~

n -> next => (*n). next

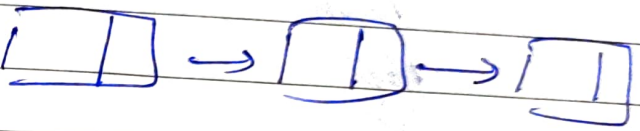


obj. prop
(*ptr). prop

- Recursive destructor call to delete

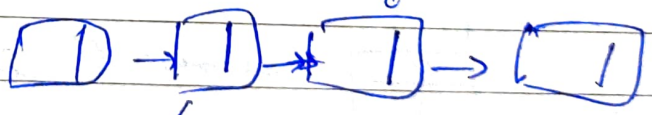
```
~List(){
    if(head != null){
        delete head;
        head = NULL;
    }
```

calls delete node which calls delete on next recursively until next is NULL



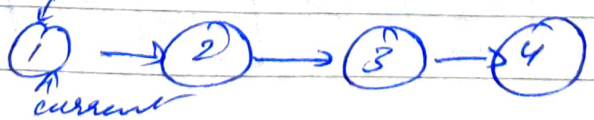
- in pop front: if simply move head to head -> next then it will cause memory leak
- make temp -> next to NULL so it doesn't delete whole LL

- pop(pos) at any pos



n = temp -> next (taking val in n)
temp -> next = n -> next
n -> next = NULL // secure link
delete n;

Reverse a LL: (swapping of not done cause expensive)



prev = NULL

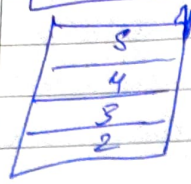
temp = NULL
current = NULL

```
while(current != NULL)
{
    temp = current -> next
    current -> next = prev
    prev = current
    current = temp
}
```

Date ___/___/___
Page No.

Stacks :

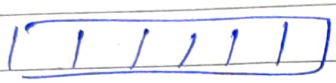
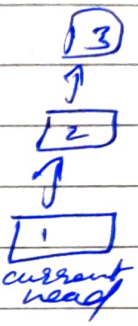
LIFO / FILO



Push() : O(1)
Pop() : O(1)
top() : O(1)

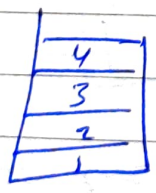
Eg :- Undo (acts like stack)

HELL
↳ HELLY
↳ HELL
↳ HELLO

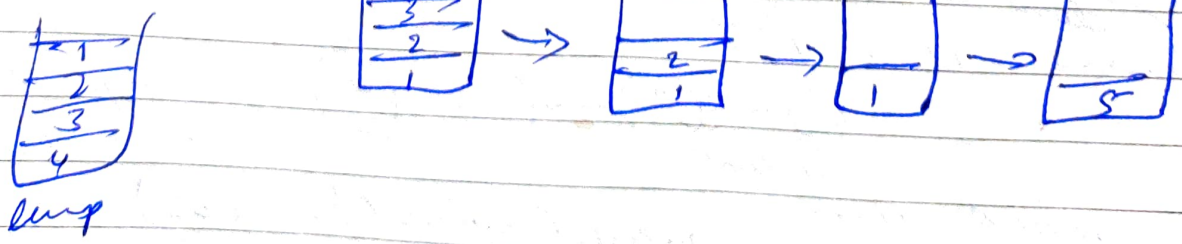
Implement using : Array (Fixed)	Vector (Dynamic size)	linked list (Dynamic data & size)
		
Push	"	"
Pop	"	"
Top	"	"
Stack full		
Stack empty		

Stack : C++ STL : #include <stack>
↳ passed by val like vector

- Stack : insert at bottom intuition :



insert 5 then first move all to temp stack
insert no. then insert back



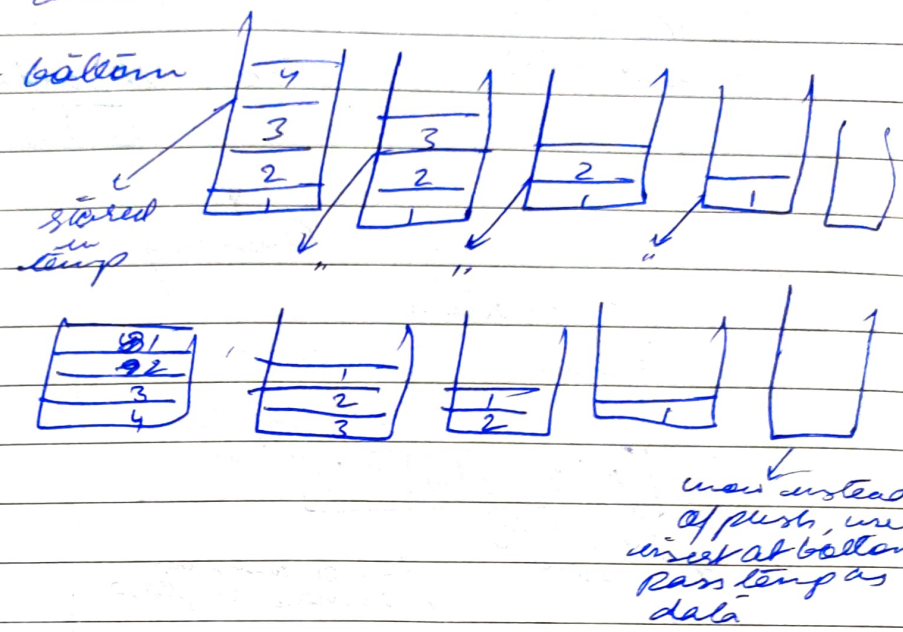
Extramarks

Date ___/___/_____

Page No. _____

- Stack : reverse a stack intuition :

use insert at bottom



main instead of push, we insert at bottom pass temp as data

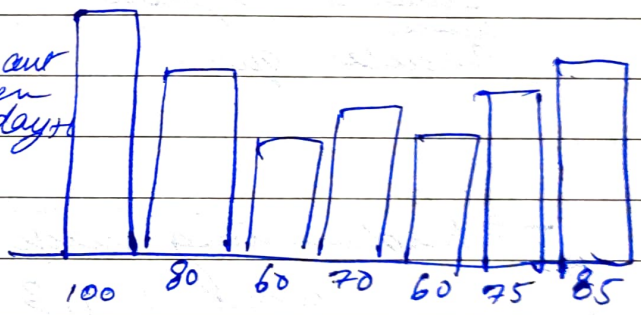
Stock span problem : intuition

→ in case if all pops out then day+1

span	6, 85	Brute way :
6-0=6	~~5, 75~~	using stack span = 5 - 1 = 4
	~~4, 60~~	span = 4 - 3 = 1
	~~3, 70~~	span = 3 - 1 = 2 Price
	~~2, 60~~	span = 2 - 1 = 1 Days
	~~1, 80~~	span = 1 - 0 = 1 Highest span
	0, 100	span = 1

days price

push days, if lower than prev then substract with day of prev if higher than pop it & all lower than



One way :

Brute force : $O(N^2)$

for each element check all prev for highest

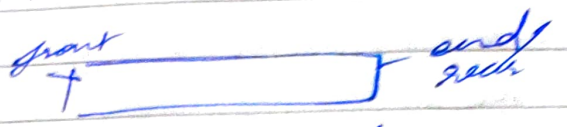
Extramarks

Date ___/___/_____

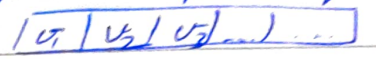
Page No. _____

Queue data structure : FIFO

- Push O(1)
- Pop O(1)
- Front O(1)



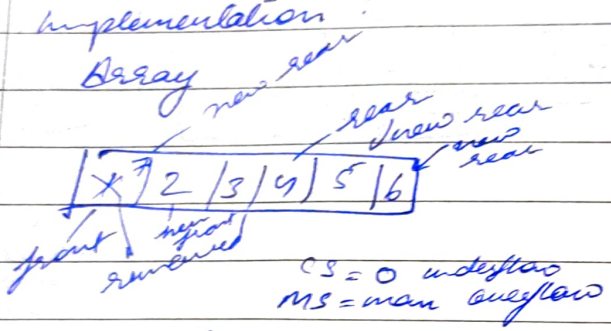
eg:- yt vid upload queue on server



Processing

Implementation :

Array



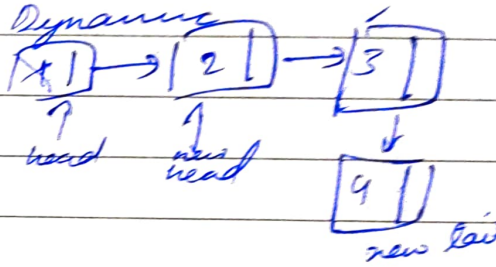
Insertion → Rear +1

Removal → front

hence:
Circular queue

CL

Dynamic



→ Implementing stack using 2 queue (FILO, FIFO)

insert 1, 2, 3

Q_1 - 1, 2, 3

Q_2

pop 3 then move 1, 2 to Q_2 & remove 3

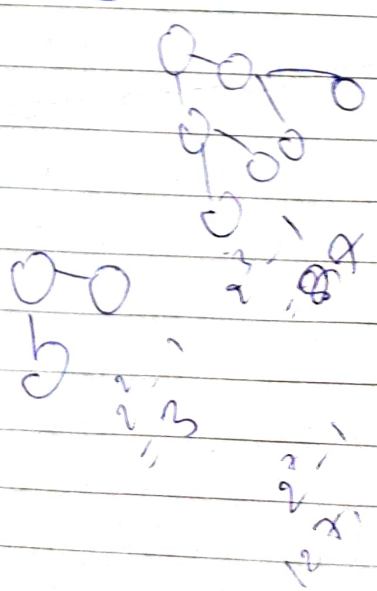
Q_1 - ~~3~~

Q_2 - 1, 2

insert 3

Q_1

Q_2 - 1, 2, 3



Extramarks

Date ___/___/_____

Page No. _____

if root taken as 1, then height of tree will be given as 4

height of tree: max edges from root to leaf so at root: 0

always declare root as h=0 or 2

at max nodes: $2^{h+1}-1$

Queue

CEO — CTO

Teddy lead — marketing lead

hierarchy

$N(0)=1$

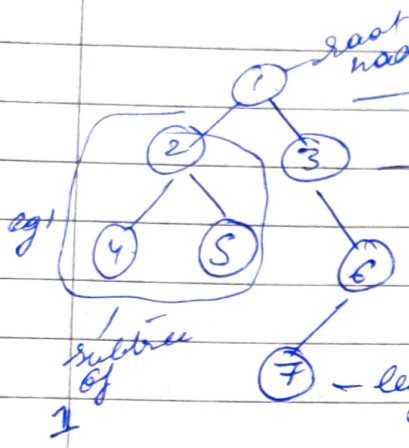
$N(1)=2$

$N(h)=N(h-1)+1+N(h-2)$

Binary tree

each node at max 2 child

- min no. of nodes in BT: $h+1$ (height)
- for N-ary tree: $(n-1)*I+1$ (leaf nodes, internal nodes)



height of max level = 4

depth of Node 4 = 3

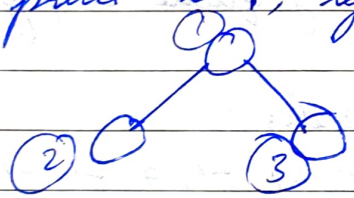
2: Parent — 4 / ancestor: 1, 2

children = ~~3~~ none

4, 5 - siblings

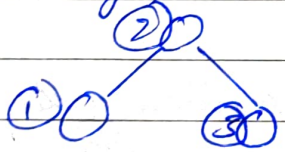
Descendants of node 1: 2, 3, 4, 5, 6, 7

Preorder: print root, left, Right



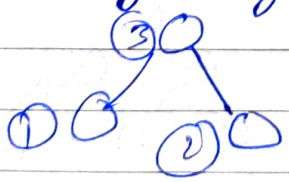
eg!: 1, 2, 4, 5, 3, 6, 7

Inorder: print left, root, right



eg:- 4, 2, 5, 1, 3, 7, 6

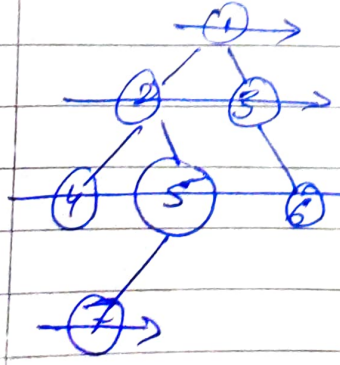
Postorder: print left, right, root



eg:- 4, 5, 2, 7, 6, 3, 1

→ Print level order (BFS) breadth first search

, intuition:



2, 3

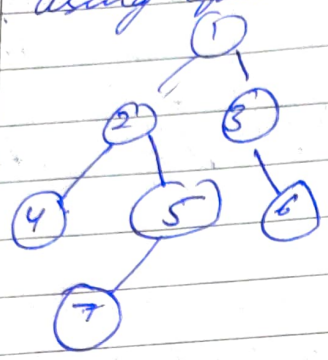
4, 5, 6

7

1, 2, 3, 4, 5, 6, 7

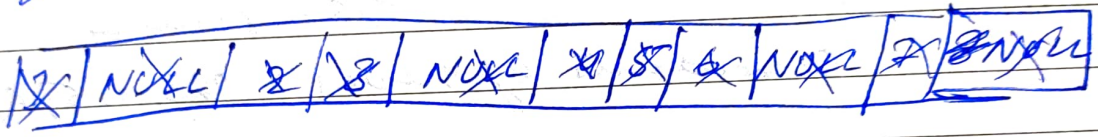
Date ___/___/_____
Page No. _____

using queue:



visit 1,
pop 1,
put its child
visit 2,
pop 2,
put its child
:

to change line after every level
after 1st val push NULL and then for each NULL change line & push another NULL in queue



stop pushing NULL if queue empty

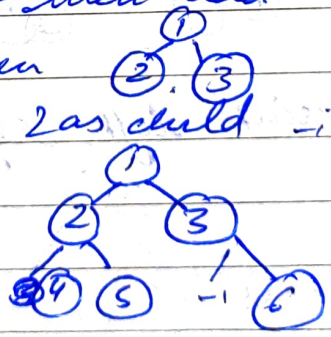
→ Build tree level order:

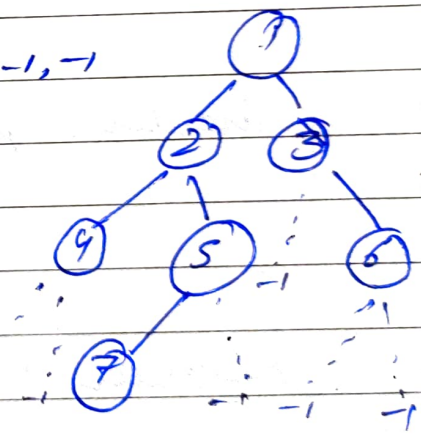
1, 2, 3, 4, 5, -1, 6, -1, -1, 7, -1, -1, -1, -1, -1

using queue

Read 1st val, pop it then add next two as its children

- pop 2, attach next 2 as child





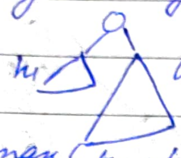
- pop 3, "

-1 means null

:

go on till queue empty

→ height of binary tree

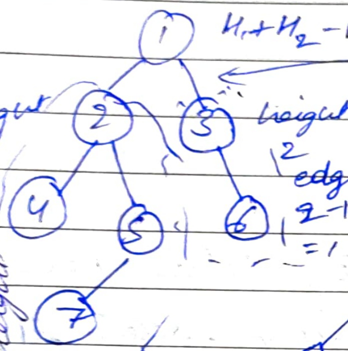


$h = max(h_1, h_2) + 1$

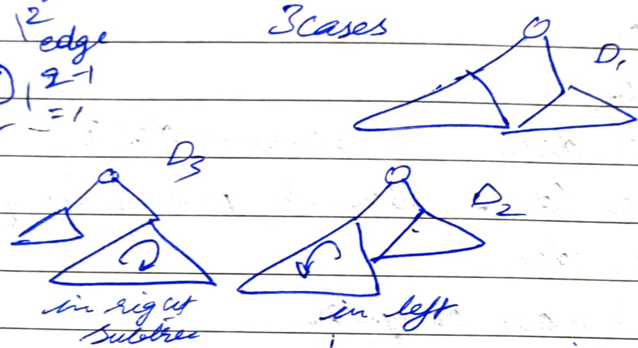
GFG: end start & end
POTD: counting nodes
min: w/o edges & nodes

→ Diameter of tree: farthest distance b/w 2 nodes
intuition max(D_1, D_2, D_3)

$H_1 + H_2 - 1 - 1 + 2 = H_1 + H_2$
2 edges



3 cases

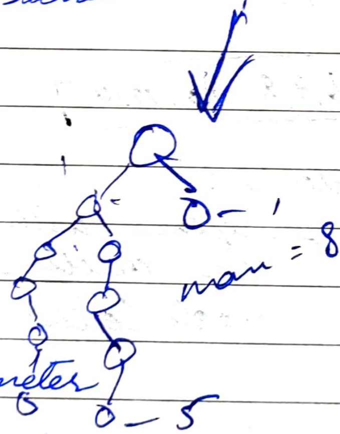


instead of 2 diff calls for diameter & height (recursive call) create class obj having height & diameter

$O(N^2)$

if root == NULL
return 0;

$D_1 = H_1 + H_2$;
D_2 = diam(root->left)
D_3 = diam(root->right)
D = max(D_1, D_2, D_3)
return D;



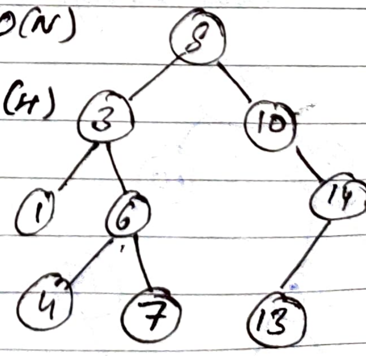
→ Optimised diameter

(Binary search) tree
Binary search tree : intuition

traversal for searching

BT : O(N)
time complexity
BST : O(H)
time complexity

- It is binary tree
- each subtree is BST
- left subtree (nodes) smaller than root
- right subtree (nodes) bigger than ~~root~~ root
- Inorder traversal : left, root, right (incr. order)



Take in key
check if root null then key
if smaller then go left,
if bigger then go right
till root = null

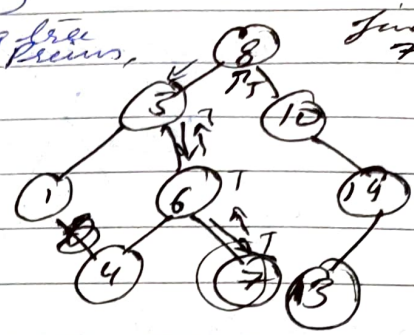
Extramarks

Date __/__/____
Page No. _____

Balanced Binary Search Tree is also called AVL tree (rotation is done)
(different thing)
& MST: min spanning tree is treated by Prims, Kruskal

Search: initialization

bool func, traverse from root to leaf. if NULL then return false, if found then T



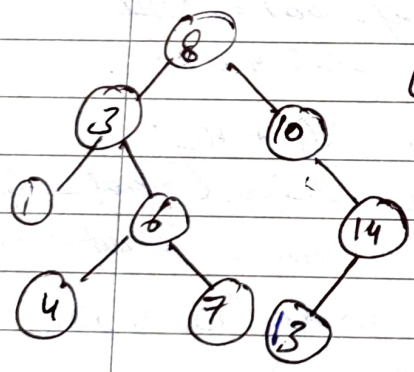
find 7

Search time complexity : $\log N \leq O(H) \leq N$

→ Deletion

BST Nodes

- Non children - search, delete, return NULL to parent
- One child - search, put in temp, delete temp, return root (subtree to parent)
- Two children
  - ↳ most complex



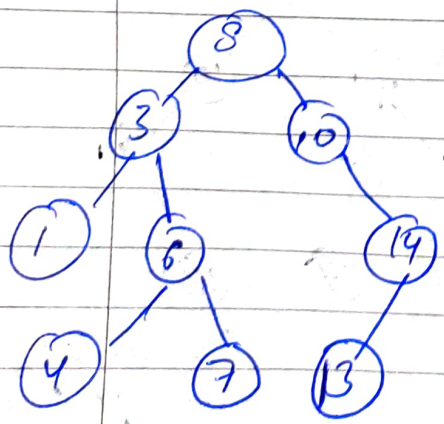
[1, 3, 4, 6, 7, 8, 10, 13, 14]

delete 3:

- replace with inorder successor
- then delete inorder successor node (either 1 or zero child)

→ Print range of numbers (k_1, k_2):

5, 12.



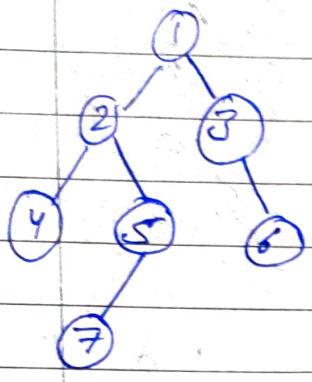
- start with root
  - if in range then call for (inorder) on left, print, call on right
  - if null then return
  - if bigger than range then go left
  - if smaller then go right

Extramarks

Date ___/___/_____

Page No. _____

→ Print all paths : root to leaf



→ 3 paths:

1, 2, 4

1, 2, 5, ~~8~~ 7

1, 3, 6

using vector:

| 1 | 2 | 4 |

| 1 | 2 | ~~4~~ 5 | 7 |

| ~~8~~ 1 | ~~8~~ 3 | ~~5~~ 6 | ~~7~~ |

2 Base cases → leaf node : print path return

→ NULL node : return

Priority queue (Heaps) → eg :- vaccination drive

(1) Sorting acc to priority $O(N \log N)$

(2) Priority queue $O(N + K \log N)$ — build priority, find top, $K < N$ — all no. sorted to get ans

A B C D E F G

10 30 35 22 40 56 18

$K = 3$ { F 56, E 40, C 35 }

(i) Insert : $O(\log N)$

(ii) Pop max / Pop min : $O(\log N)$

(iii) Get max / get min : $O(1)$

Other options :

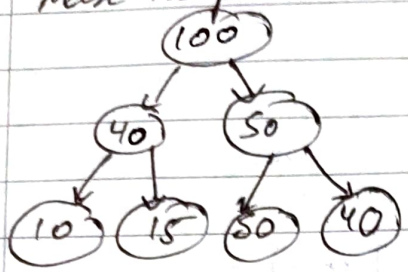
Op.	Array	Sorted Arr	List	BST	Heap
push	$O(1)$	$O(N)$	→ ✓	$O(N)$	$O(\log N)$
pop	$O(N)$	$O(1)$	→ ✓	$O(N)$	$O(\log N)$
Get	$O(N)$	$O(1)$	→ ✓	$O(N)$	$O(1)$

Extramarks

Date ___/___/_____

Page No. _____

Max heap:



rule

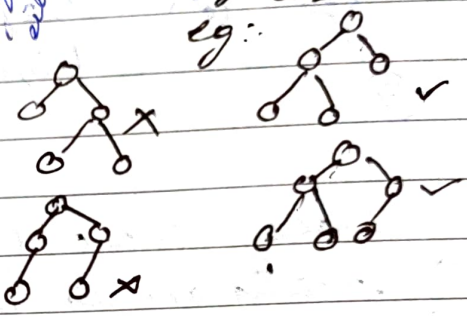
3. heap order property
parent ≥ children

1 → Binary tree

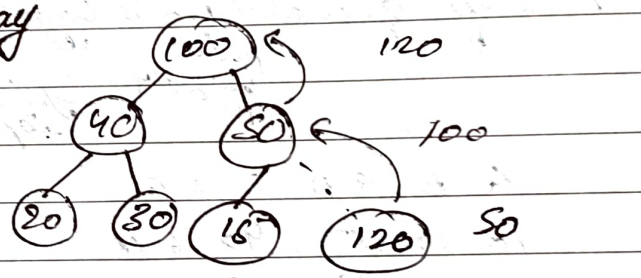
2 → CBT: complete BT all part. of tree filled except last level but may not to be filled left to right order

full BT: either have 2 or zero child

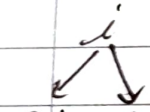
eg:



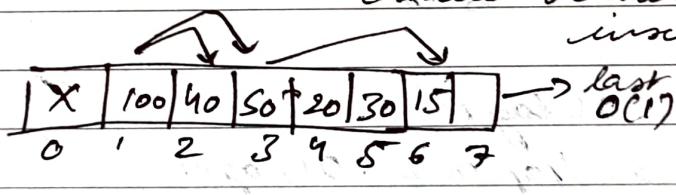
- heap is stored as array
  - why not as BT

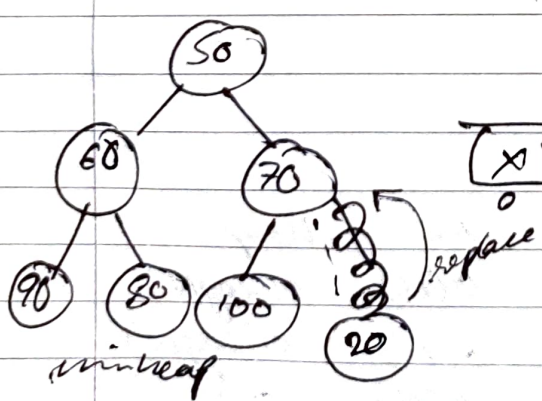


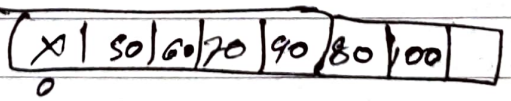
O(N) times ~~search~~



creates bottleneck for insertion







insert 20

worst case: height tree
O(log N)

rule: parent ≤ children

min heap

| x | 50 | 60 | 70 | 90 | 80 | 100 |

Extramarks

Date __/__/____

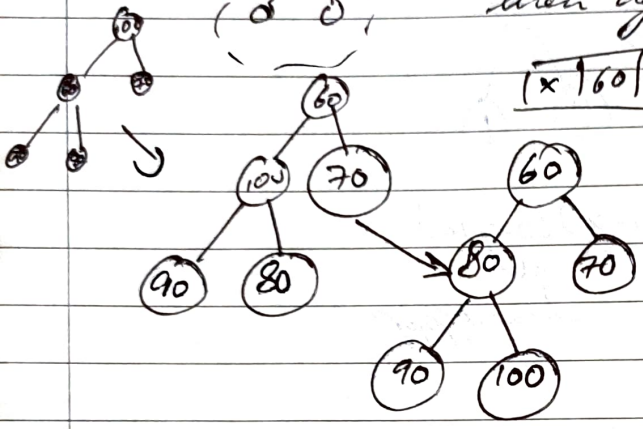
Page No. _____

get min() - O(1) : return arr[1]

Remove min()/Pop()

- swap v[1] and v[size-1] O(1) → | x | 100 ... | 50 |
- remove last element (pop_back) O(1) | x | 100 ... | 80 |
- heapify : logN → so for n ele : NlogN, (but if create heap O(N))
  - compare all 3 swap with smallest
  - then again till becomes min heap

| x | 60 | 80 | 70 | 90 | 100 |



also can use custom comparator

Heap STL:

- Min heap : priority_queue<int, vector<int>, greater<int>>
- Max : priority_queue<int> name;

→ Finding nearest cab : intuition

- N cartesian pts in a 2d plane : each rep a cab
- standing at origin : nearest k cars?

N = 5, k = 3

C1	1, 1	√2	(euclidean dist : $\sqrt{x^2+y^2}$)
C2	2, 1	√5	ans C4, C1, C2
C3	3, 2	√13	
C4	0, 1	√1	
C5	2, 3	√13	

ways :-

1. Sort (NlogN + k)
2. Heap (N + klogN) (minheap)
3. Using max heap :

PTO

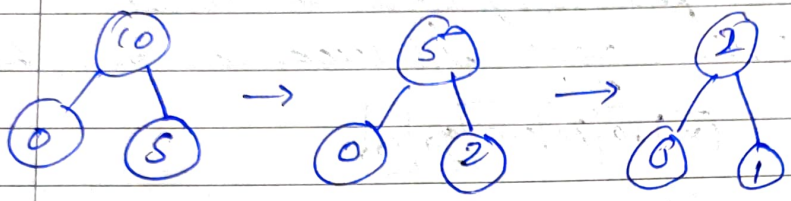
Extramarks

Date ___/___/_____

Page No. _____

Dist : 5, 0, 10, 2, 6, 8, 1

k = 3 create max heap using 1st 3 elem - then insert no. if smaller than greatest
replace w. greatest



complexity : $O(k + (n-k)\log k)$

k → create priority queue of size k

$\log k$ → insert/pop

→ Hash table → unordered DS, not we used to maintain order

Eg:- Menu (key) (value)

Maggi - 40

Dosa - 80

Soup - 35

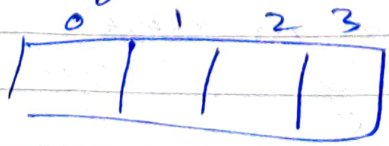
later O(N) time to search soup

but through hashing O(1) time

Operations : Insertion, Deletion, Search } O(1) avg time

Hash func[obj] ==> integer value

→ fn converts keys to integer val



Burger" → fn → address

key

83

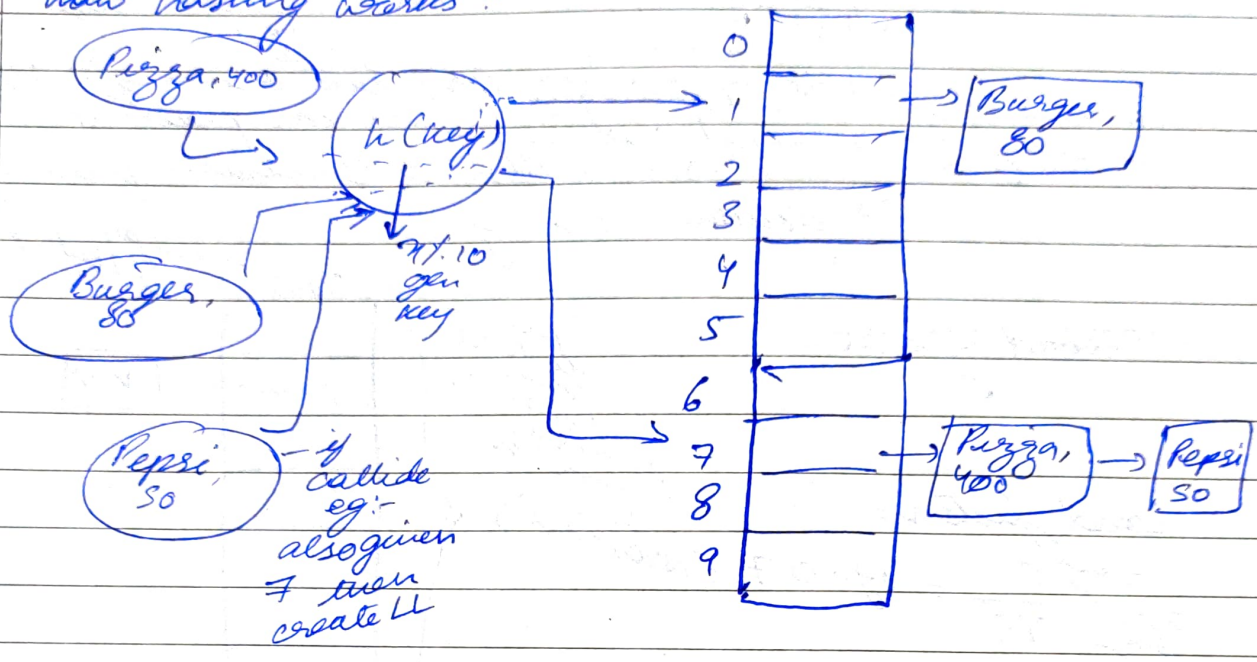
Extramarks

Date ___/___/_____

Page No. _____

Key components: — hash function (design)
— hash table (Array)
— collision handling scheme

How hashing works:



Hash table: - array of fixed size
- indexed by key mapped to an array index (0 to table size - 1)
- mapping (hash func)
↳ h from key to index
↳ h("john") = 12 for john
op → insert: T[h(key)] = value
↳ template
↳ search: return T[h(key)]
↳ delete: T[h(key)] = NULL

Good hash func: → Reduce chance of collision - distribute keys uniformly over the table
→ Should be fast to compute

simple hash func: → key % table size
→ Table size = prime

hash func for strings → add ASCII val of letters → % by table size eg: abcd 394 % 10

another approach (PTO)

Prob → anagrams map to same index
Drawback: strings may not use entire table

Date ___/___/___
Page No. ________

→ data weighted sum

k = prime no.

k = 29, 31, 37..

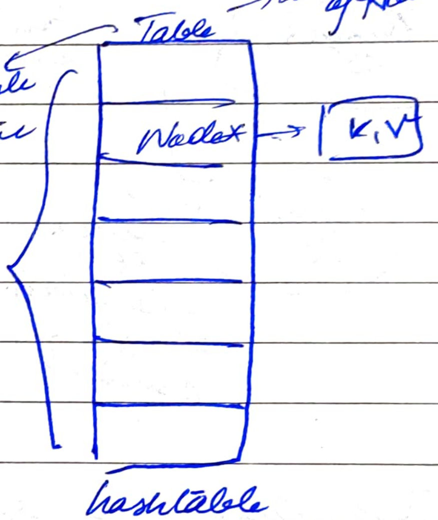
(idx) a b c d

Sum = $a\,29^0 + b\,29^1 + c\,29^2 + d\,29^3$

sum % table size

Collision handling
- → better hash func.
- → ~~Open hashing~~ Open hashing (was covered) (seperate chaining) : using LL — deletion easy
- → closed hashing → open addressing
  - → linear probing
  - → Quadratic probing
- → double hashing

→ ~~Load factor~~



ptr to table since dynamic arr to track in size (Node**)

Load factor = $\frac{CS}{TS}$ = eg:- 0.7

current size: CS
Total size: TS

crosses certain ~~the~~ threshold

do rehashing

↳ during this insertion: O(N)

↳ incr size : 2n

Node
- String key
- T val
- Node* next

→ operator overloading : convert insert ("Mango", 100)
to h["Mango"] = 100

↳ done through pass by reference

h → obj of hashtable, use like array

Trie structure :-

words[] = {"apple", "ape", "mango", "~~news~~" "news", "no"}

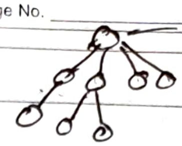
Queries[] = {"apple", "banana", "new", "mango"} search

Sample o/p:

Yes, No, No, Yes

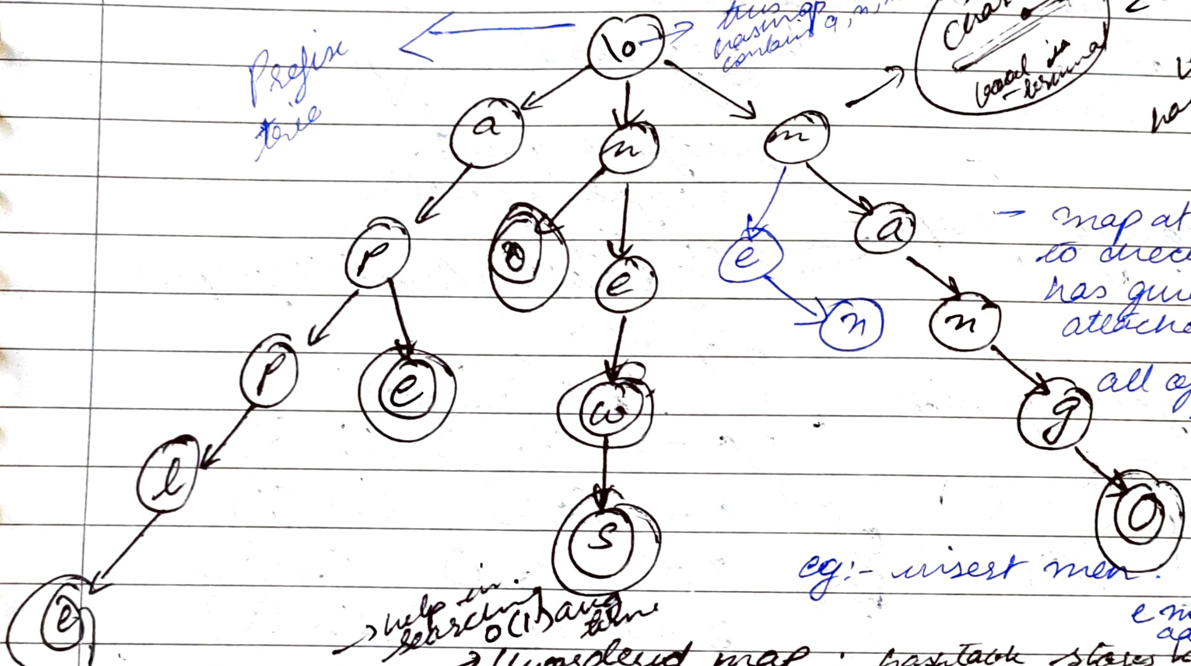
ways:
1. Brute force
2. BST — O(logN)
3. Unordered set
4. Trie — O(W) → word length

Extramarks
Date ___ / ___ / ______
Page No. ______



- generic tree like data structure
- efficient info retrieval data structure
- searches in optimal time O(key length)
but uses extra storage

important op :-
- Insert new word → O(NL) → building trie
  → letters or word length
- Search for a given word → O(w) / O(L)



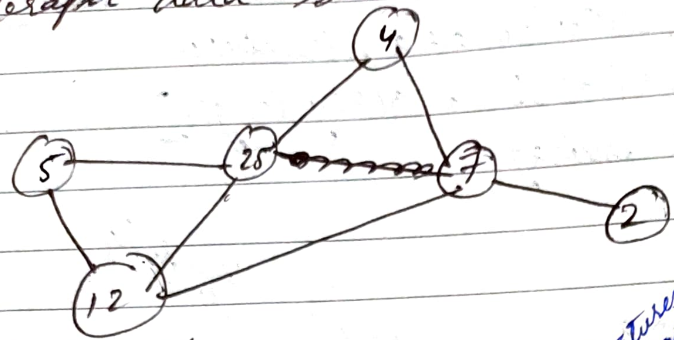
- map at each node to check if it has query letters attached to it;
  all of them store ptrs to attached next node

eg:- insert men. check if map of "m", e not present add node, add its address to map of m, now do same for node "e", check its map "n" not present so add & set terminal

hashtable STL → unordered map : hashtable stores key, val → helps in searching O(1) avg time
→ unordered set : stores key only
↳ used to solve → array problems
→ LL problems
→ sliding window
→ graphs
↓
helps in solving brute force
- map : ordered lexicographically : time : O(logN)
eg prob : Imple. phonebook
↳ use map with 2 datatype <string, vector <string>>
lexicographically arranged
dynamic, any amt of no a person has

Graph data structure

Network →
vertices
edges

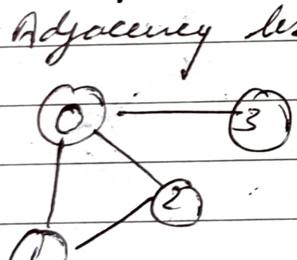


edge has val
↑
weighted vs unweighted
Directed vs undirected → features edges: $\frac{n(n-1)}{2}$ — n vertices
— no. of degree vertices is even
— sum of degrees of all vertices is even

↓ only one direction
↓ bidirectional edge

Storing a graph
→ adjacency list → O(V)
→ adjacency matrix → $O(V^2)$ space
→ Edge list → $O(V+E)$ space
→ 2d Matrix (implicit graph)

- Adjacency list



O(x) to visit neighbour

←x→ neighbours

O(1) for gets

0 ⇒ (1,2,3) list[0]
1 ⇒ (0,2) list[1]
2 ⇒ (0,1) list[2]
3 ⇒ [0] list[3]

can be implemented using:-
list<list>
Array<list> → linked list
vector<list>
vector<vector>
hashmap<int, list>

- Adjacency matrix
V nodes
$O(V^2)$

edge b/w 1 & 2 (1,2)

	0	1	2	3
0	0	1	1	1
1	1	0	1	0
2	1	1	0	0
3	1	0	0	0

disadv: too much memory
O(V) iterating
eg for 3 iterate over whole 3 row
but ad list direct

Extramarks

Date ___/___/___

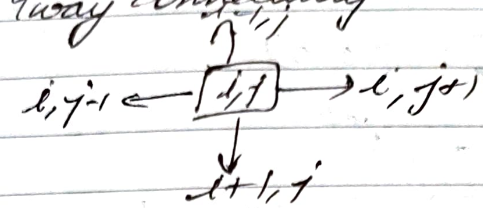
Page No. ______

→ Edge list = {(0,1), (0,3), (0,2) (2,1)}
w1 w2 w3 w4

→ store all edges with weight

list < triplet (wt, source, dest >>
(6, A, B)

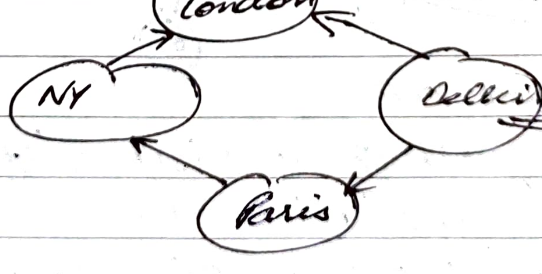
→ Implicit graph : 4 way connectivity



Graph applications :

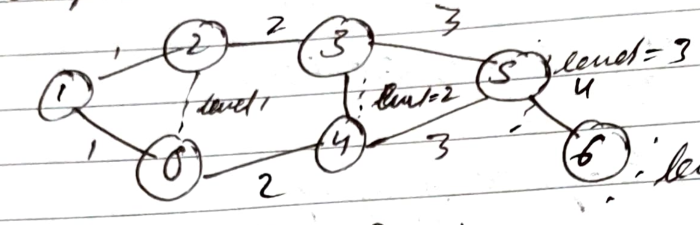
- shortest path (ola, uber)
- social network (mutual conn.)
- shortest cyclic route (amazon delivery vans)
- Dependency graphs (installing softwares & their dependences)
  ↳ DAG directed acyclic graph
- Routing algo. (internet routing)
  ↳ shortest path for datapackets to flow
- Computation graph (Deep learning)
- Computer vision (flood fill & scale find boundary of obj)
- web crawlers (to visit links gen. from a link)
- Physics, chem (modelling atoms molecules)
- Graph database eg:- Neo4j (recommendation engine, fraud detection, AI based app)

Graph (cities)
↳ use when node is complex



→ each city Node contains
string name
list Neighbours (LL)
& to access addr of city create hashtable <string, Node*> then iterate to see nbrs

Graph traversal :



→ BFS : Fifo (queue)

push 1,

[1]

pop 1, add it's neighbour
maintain "visited" ~~list~~ array [1 | | |]

[1 | 2 | 3]

[1 | 2 | 0 | 3 | 4 | 5 | 6]

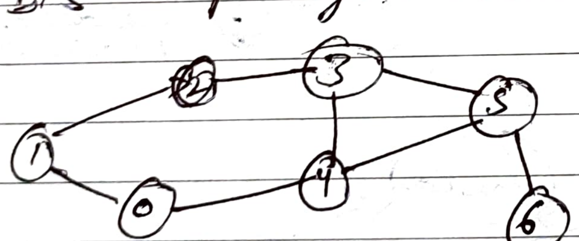
BFS because node at same level printed together
level - no. of edges from source time : O(V+E)
↳ in adj matrix O(V²)

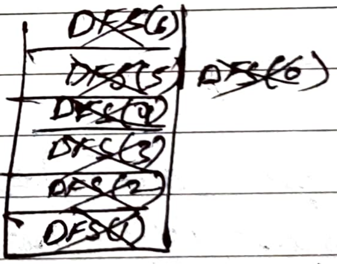
space complexity O(V) worst case all present on stack at same time

space complexity O(V)

technically O(V+2E) since each edge visited twice through each vertex but const. not considered

→ DFS : depth first search

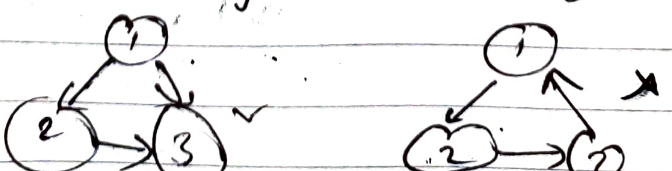




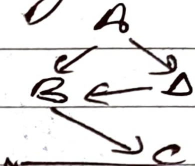
- recursive call
- depth first cause visit child node of 1st child
time : O(V+E) → in adj matrix (O(V²))

→ DAG (directed acyclic graph) : linear ordering of vertices, every directed edge u,v , u comes before v in ordering (hence no cycle)

Topological sorting for DAG

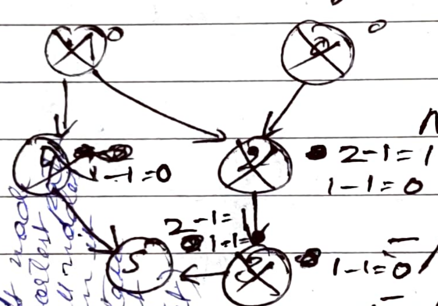


DAG app. :
- Scheduling jobs from given dependencies
- Instruction scheduling
- Ordering of formula cell eval.
- logical synthesis
- Determining order of compilation tasks to perform in make files, data serialization etc.
- Resolving symbol dependencies in linker



A,D,B,C

Topological Sorting :



No. of dependencies = indegree for every node (edge coming towards it)
- push all 0 dep. in queue
- pop & reduce dep. on nodes having outgoing edges from this & print

| X | X | X | X | X | X |

1, 0, 4, 2, 3, 5

Time complexity: O(E log V)

Bellman ford : O(VE)

shortest path: take first node & calc shortest from all nodes. again shortest path, repeat

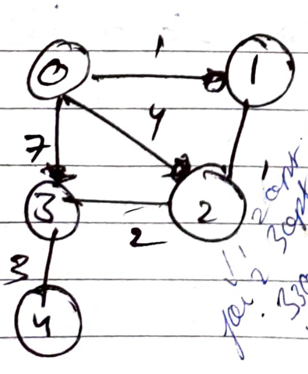
⇒ Dijkshtra's algo : shortest path
- single source shortest path in weighted graph
  ↳ calc shortest dist to all nodes incl dest

on node to another while MST shortest path to cover all nodes

& AVL tree is balanced binary search tree (log n time)

single source to all nodes no cycle

MST ways: Prims — select min cut edge then min cut edge from discovered nodes; all vertices must be connected
Kruskal — select min cut edges, no cycle



1: 2opt: min(1,5)
pair 2: 3opt: min(4,12,9)
3: 3opt: min(7,6,4)
4: 2opt: min(10,9)

Dist

0	1	2	3	4
0	~~∞~~ 1	~~∞~~ ~~4~~ 2	~~∞~~ ~~7~~ 4	~~∞~~ 7

Set. pair (int, int)

~~0 0~~	~~1 1~~	4 2	7 3

- all nodes from 0, greedy algo select lowest

update: delete insert

~~2~~	~~4 2~~	7 4

* in exams, same dp ques are asked so do thoroughly
* for interview most of the time repeated
* for amazon, list on LC
* turns 2^n prob into $M \times N$ but if MN greater than 2^n then bad → 2^n because 2 options & 2^n ways to do it

Date ___ Page No. ___

* → [99% of time questions repeated]

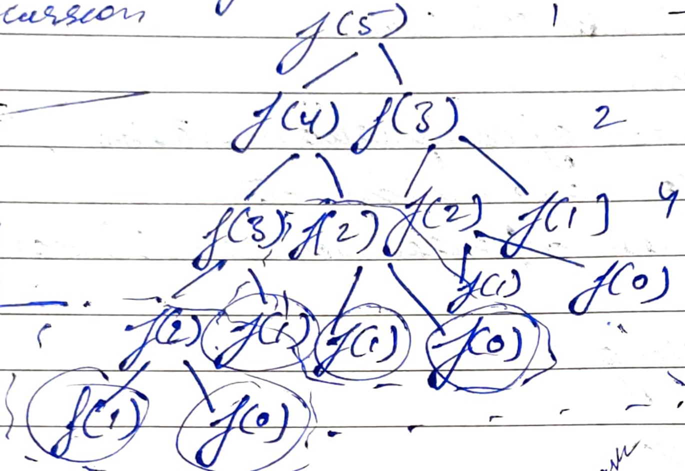
Dynamic programming : learning from past → optimised brute force

- Top down : recursion, memoization (starts from big value & goes to smaller)
- Bottom up : Tabulation (starts calc from start)

0, 1, 1, 2, 3, 5

fibonacci : f(5) = 3

recursion



1, 2, 4, 6 worst case 8, 16, 2^{n-1}

overlapping subproblems → computing same thing every time

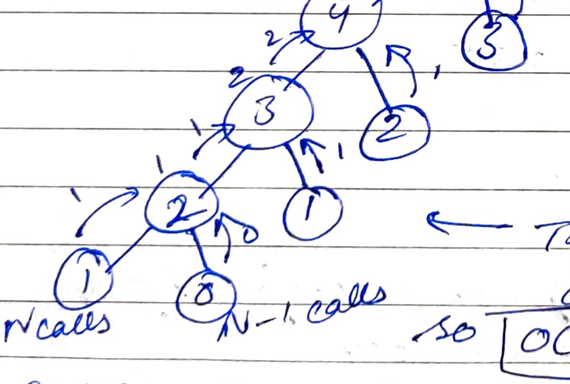
with $O(2^{n-1})$, $O(n)$

w/o

$1 + 2 + 4 \ldots \ldots 2^{n-1}$
$= O(2^n)$

fibonacci : → check codeknown for code → check gfg for difference between greedy & dp

recursion + memoisation



0	1	1	2	3	5
0	1	2	3	4	5

array to store f(n) values

← Top down DP (recursive)

so [O(N)]

Bottom up DP : use small solutions to build big sol

iterative approach

| 0 | 1 | . . . . . |

O(N)

```
for (i=2, i<=N, i++)
{ dp[i] = dp[i-2] + dp[i-1];
}
```

All Greedy algo —
- selection sort → takes min swap
- knapsack prob
- MST
- Single Source shortest path
- job scheduling
- Prims
- Krushkal
- Dijkstra
- Huffman, Ford Fulkerson

concept: select min or max each time

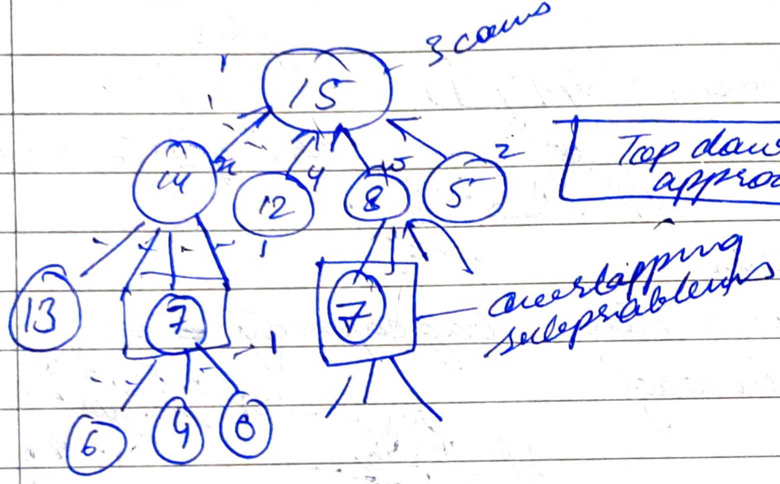
Coin change: → time $O(M * \text{total coins})$

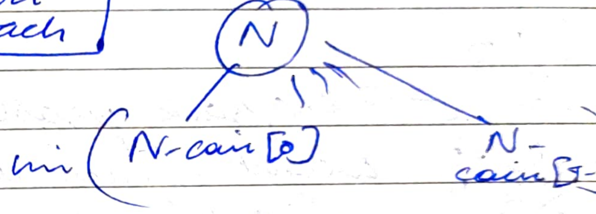
an array of coin change denomination, integer M: large money
min coins seq for change?

Input
coins = [1, 3, 7, 10]
M = 15

Output
3
(7+7+1)

$f(n) = \min(x, y, w, z) + 1$

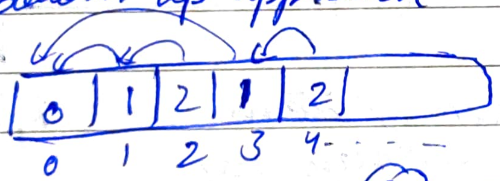




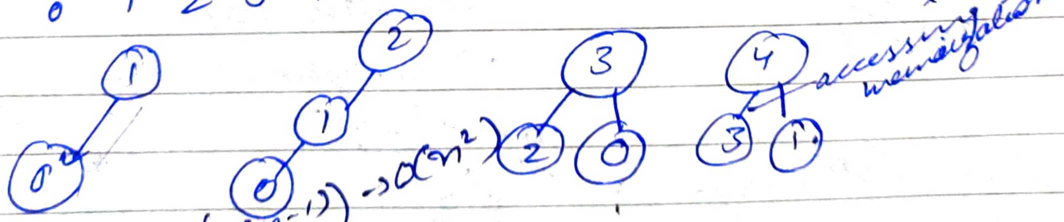
+1

if N==0
return 0
if (N < coin)
return

Bottom up approach:



$dp(n) = \min(dp(n - coin[i]) + 1)$



$\rightarrow O\left(\frac{n(n-1)}{2}\right) \rightarrow O(n^2)$

→ LIS: longest increasing subsequence
array seq., find subseq. that is arranged in incr order

Input:
50, 4, 10, 8, 30, 100

Output:
4
{4, 8, 30, 100}

Extramarks

Date ___/___/_____

Page No. _____

arr = [50, 4, 10, 8, 30, 100, 2] (2)

dp = [1, 1, 2, 2, 3, 4, 1] → iterate & select max (dp)

(50) (4) (4,10,30,100)

subseq of len=1 (4,10) (4,8) (4,10,30) (4,8,30,100)

dp[i] = 1 + max(dp[j])

= 1

for $0 \le j < i$

arr[j] < arr[i]

→ Knapsack problem:

weight & price of n items, bag capacity = W

max total value in bag?

input

N = 4, W = 11

wt = [2, 7, 3, 4]

prices = {5, 20, 20, 10}

Output:

40 (7+3)

[20] [20]

subset of items:

→ max val

→ weight ≤ W

N items, 2^N ways → maximal time $O(2^N)$

f(n, W)

include = price[n-1] + f(n-1, W - wts[n-1])

excl = 0 + f(n-1, W)

Base case:

if n == 0 or W == 0

return 0

Bottom up DP (Knapsack)

	0	-	-	-	W
0	0	0	0	0	0
.	0				
.	0				
.	0				
N	0				N/W → ans

— 2d vector matrix → solve like subset sum

$dp(0)(i) = 0$
$dp(i)0 = 0$

val	wt	i	0	1	2	3	4	5	6	7	8	9	10	11
		0	0	0	0	0	0	0	0	0	0	0	0	0
5	2	1	/	0	5	5	5	5	5	5	5	5	5	5
20	7	2	/	0	5	5	5	5	5	20	20	25	25	25
20	3	3	/	0	5	20	20	25	25	25	25	25	40	40
10	4	4	/	0	5	20	20	20	20	30	30	35	40	40

Time & Space = $O(NW)$ → use RVB notes for ref

dp: $O(n^2)$ → can reduce it to current, prev

normal (2^n ways sell either this or that) → 2^{20} $O(2^n)$

Wines problem:

→ ans will always be at pos from where recursion started ($L = 0$, $R = n-1$)

	2	3	5	1	4
2	40				ans
3	0	45			
5	0	0	25		
1	0	0	0	5	2nd last
4	0	0	0	0	20 → last bottom

yr=1 [2, 3, 5, 1, 4]
every yr → yr2 = 4, 6, 10, 2, 8
price × yr.no → yr3 = 6, 9, 15, 3, 12

condition (bottom up) solution:
if ($i == j$) → $N \times price[i]$
if ($i < j$) right then approach
$y = N - (j - i)$
$PL = N \times price[i] + dp[i+1][j]$
$PR = \times price[i] + dp[i][j-1]$
$max(PL, PR)$
$dp[i][j]$

can sell only 1 wine per year & only leftmost or rightmost

if try greedy algo: sell small ones earlier

[2, 3, 5, 1, 4]

2<4 3<4 4<5 1<5 5
2×1 + 3×2 4×3 1×4 5×5
2 + 6 + 12 + 4 + 25
= 49

In dp (Brute force optimised) ⇒ try all combo, sell left then right

2×1 + 4×2 + 1×3 + 3×4 + 5×5
= 2 + 8 + 3 + 12 + 25 = 50

Do:
→ Matrix chain multip. $O(n^3)$
→ Subset sum
→ Travelling salesman ($n^2 2^n$)
→ Floyd Warshall $O(n^3)$

→ ☆ see cond. from RVB practice table: code: first convert conditions in if else

RVB notes for all

Extramarks

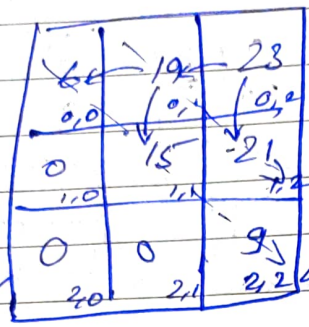
Date ___/___/_____

Page No. _____

$$f(L,R,y) = \begin{cases} p[L]\times y + f(L+1,R,y+1) \\ p[R]\times y + f(L,R-1,y+1) \end{cases}$$

lines: {2, 5, 3}

Bottom up



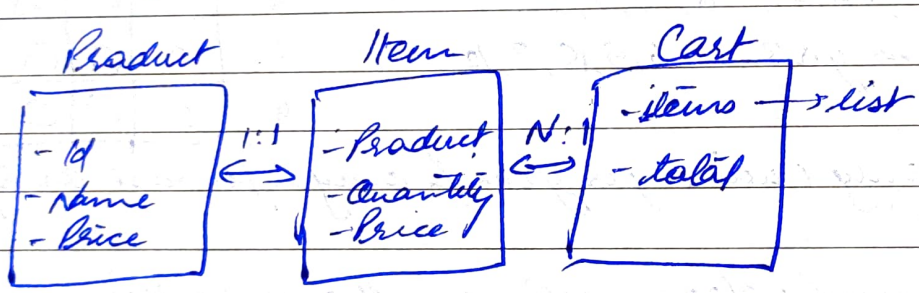
→ when 3 battles left

when 2 battles left

when only 1 battle left

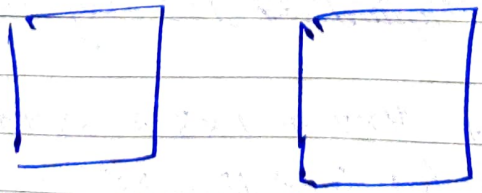
where L > R

Shopping cart:



features: (1) Views products
(2) Add prod, qty
(3) Checkout / Total

- Code judge:



Views code

Server

keep template same (java, cpp)

Run 3 test cases & match

(1) — 01 ✓

(II) — 02

(III) — 03 —

Extramarks

Date ___ / ___ / ______

Page No. ____________

Pairs : if some any pair of no. is K

1. N^2 → brute force
2. NlogN → sort & search
3. O(N) → unordered_set< > "my sol"

"my sol" ↳ add in ~~st~~ set as goes on so doesn't repeat a value

Triplets

- → $O(N^3)$ = Brute force
- → $O(N^2)$ = (N . N) — iterate — pair sum using unordered set
- → O(NlogN + N^2) = N^2 — sort; 2 pointers

1, 2, 3, 4, 5, 6
↑ s ↑ e

→ ① iterate a[i], (a[j], a[k])

same TC ("my sol"): sort then put in map then 2 for loops to find 3 ele in triplet

— Mountain : (my sol)

min 3 — inc, flat, decr

<0 1 2 3 4 5>

i ~~<~~ < i+1 > i+2

count

count = 0

corner case i−1 = −1, i+1 = n

i−1 < i < i+1 — count ++

i−1 > i < i+1 — count = 1, count set count

i−1 > i > i+1 — count ++

i−1 < i > i+1 — count ++

5 6 1 2 3 4 5 4 3 2 0 1 2 3 ~~8~~ −2 4

ct1 ct2 ct1 ct2 ct3 ct4 ct5 ct6 ct7 ct8 ct9 print ct1 ct1 ct2 ct3 ct4 ct5 print ct1

c++ c++ c++ c++ c++ c++ c++ c++ ct3 print or ct1

count = 7

count = 0

all cases of count ++ becomes "else" & print case becomes "if"

Extramarks

Date __/__/____

Page No. _____

→ not optimal

→ longest band (my sol) : band: subseq which can be arranged in consec order

Sometimes it fails

Input : 1, 9, 3, 0, 18, 5, 2, 4, 10, 7, 12, 6

O/P : 8

{0, 1, 2, 3, 4, 5, 6, 7}

- sort

↳ ct++ (if a[i+1] - a[i] == 1) mc = max(mc, ct)

else (not eq 1) → cout ct, ct = 0

O(NlogN) + N = O(NlogN)

→ find start of bands in arr & count for successor

PNG approach: using unordered_set (hashtable)

O(N) for inserting

O(1) look up

at most O(N) time

1, 9, 3, 0, 18, 5, 2, 4, 10, 7, 12, 6

× 2 × 8 1 × × × × × 1 ×

1 ⇒ check if (i-1) 0 then ×

if no then start of a band, check for i+1 & if there then c++

Rains → my sol

[0 1 0 2 1 0 1 3 2 1 2 1]

1st - no < L,R ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑

2nd - min max 0 0 0 0 0 1 0 0 0 2 0 0

O/LR - max

3rd min - no O(N²)

PNG: i = 0; n-1 find max (act as left(max))

i = n-1; 0 find max (act as right(max))

min(L[i], R[i]) - h[i] water = max(curr, max till i-1)

0 1 0 2 1 0 1 3 2 1 2 1 — for each building

Left L 0 1 1 2 2 2 2 3 3 3 3 3 — 0 to n-1 " (curr h, max h till i+1 from n)

Right R 3 3 3 3 3 3 3 3 2 2 2 1

Extramarks

Date ___/___/_____

Page No. _____

"my sol"

Subarray sort:

(my sol) al = [1, 2, 3, 4, 5, 8, 6, 7, 9, 10, 11]

0 1 2 3 4 5 6 7 8 9 10

o/p = [5, 7]

index

problem: Can't solve if arranged in desc

for 0 to < n-1

i < i+1 ✓

i > i+1 e.g. c++

{ if start == -1

start = i

end = i+1

else

end = i+1

}

9 6 7 8 10

9 8 6 7 10

8 9 6 7 10

1 2 3 5 4 9 8 7 10 11

i c new e

0 1 2 3 4 5 6 7 8 9

1 2 3 5 4 9 8 7 11 10

i e e new e c new e

i → -1 → 3

e → -1 → 5 → 7 → 9

PNG sol: → Sort & compare & mark the change (NlogN)

1, 2, 3, 4, 5, 8, 6, 7, 9, 10, 11

1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11

inversion inversion

we find index & min val out of order then in 2nd iteration find both of their pos & that is length of subarr.

O(N) if a[i] > a[i+1] or a[i] < a[i-1]

mark smallest = min(smallest, x)

largest = max(largest, x)

then again loop to insert in right pos

6 right pos 8 right pos

1, 2, 3, 4, 5, 8, 6, 7, 9, 10, 11 hence return index

Extramarks

Date ___/___/_____

Page No. _____

"my sol"

- Minimum swaps : [10, 11, 5, 4, 3, 2, 1]

N^2 time approach : sort NlogN

(My ~~spso~~ approach) compare ele if not eq then find & swap

PNG : NlogN : swaps = sum of no. of elements-1 in each non intersecting cycles

store original pos

sort all elements

if not visited or same pos {

cycle += 1, jump to old pos and mark visited

so we do cycle - 1

0 1 2 3 4 5 6

[10, 11, 5, 4, 3, 2, 1] → 1st cycle 4 ele, 3 swaps (4 cycles)

0 1 2 3 4 5 6

1 2 3 4 5 10 11 2nd cycle 2 ele, 1 swap

6 5 4 3 2 0 1

1+1+1

"my sol"

- Mindiff - Brute $O(N^2)$ if same size else $O(MN)$

my sol: sort arr a, binary search each ele of b in a for closest val $O(MlogM + NlogM)$

arrays product $O(N)$

[1 | 2 | 3 | 4 | 5]

PNG

take two arrays

left = prefix calc for each ele

right = suffix " "

left[i](prefix) = arr[i-1] * left[i-1]

right[i](suffix) = arr[i+1] * right[i+1]

prod = left * right

It is natural but including prods from DAA

Busy life ;

→ max activities

(7,9), (0,10), (4,5), (8,9), (4,10), (5,7)

2 10 1 1 6 2

0, 4, 4, 5, 7, 8

(4,5) (5,7) (7,9) (8,9) (0,10) (9,10)

(0,10) (4,5) (4,10) (5,7) (7,9) (8,9)

N^2 i

j →

i.second <= j.first

Extramarks

Date ___/___/_____

Page No. _____

→ optimised bubble sort :

for best case : O(N),

my sol: for loop at start a[i] < a[i+1] (c++)

if c == n-1 : sorted

size

→ chopsticks

1, 3, 3, 9, 4 n = 2

~~1 3 3 4 9~~

~~1 2 3 4 5 6~~ n = 2

~~1 3 5 9~~

(1, 3) = 1

→ Defense kingdom :

15, 8

(3, 8), (11, 2), (8, 6)

(3, 2) (8, 6) (11, 8)

2 x 1 ~~4 x 3~~ 4 x 1

- run for each row & each col of tower
- calc sq

→ abcd, dabc

for each letter check in string

if not true then return false

string subsequence sort

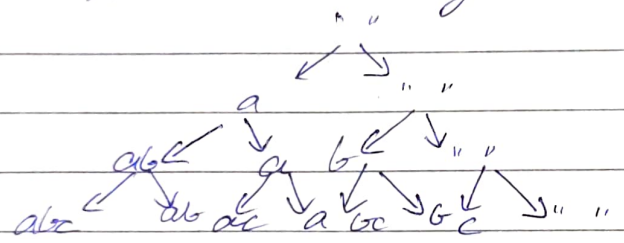
abcd

a	b	~~dc~~ c	d
ab	bc	cd	
ac	bd		
ad	bcd		
abc			
acd			
abd			
abcd			

$2^n - 1$

ab	abc
a	a
b	b
ab	c
	ab
	~~bc~~
	ac
	abc

PNG sol: recursion: gen all subsets → put in vector → sort it



10, 11, 20, 30, 3 (Biggest no. string) – My sol.

3, 10, 11, 20, 30

10, 11, 20, 3, 30, 31, 32, 33, 34

34, 33, 3, 32, 31, ~~21~~ 20, 11, 10

3 54, 53, 52

10, 11, 20, 30

3, 5, 8 30, 20, 11, 10

8, 5, 3 ↳ divide (log10) if ans greater then put it first

(85533020 1110)

66

86260S

31 38 27

39

853

Extramarks

Date ___/___/_____

Page No. _____

PNG sol: for each x and y, compare xy and yx whichever greatest, arrange in that order.

Palindrome break - my sol ✓

reduce letter from leftmost → convert to a then check if palindrome

increase letter from rightmost → convert to b then check if palindrome

abaaba

aaaaba

Sliding window:

solve problems in array & string: reduce complexity from $O(N^2)$ to $O(N)$

→ Hausing: — my sol → $O(N)$ → learned myself as same PNG

A[1], A[2] ... A[n]

subarray whose sum is k

| 3 | 1 | 2 | 4 | 5 | 8 | 7 |

i

j

if greater than k: i++, sub a[i] from cs

if less than k: j++ add a[j++] to cs since j++ will cause it to update after sum

if cs == k, return i, j, i++

1	3	2	1	4	1	3	2	1	1	2
0	1	2	3	4	5	6	7	8	9	10

i

j

cs=1

k=8

8	1	3

i

j

4

Q unique substring :-

prateekbhatiya
↑↑
i,j

hashmap:

PNG sol:
abcaeba   window = ~~3~~ 4

a : ~~0~~ 3
b : ~~1~~ 5
c : 2
e : 4

i
j

j keeps moving
- add if not in unordered map with loc
- if same occurs then shift i to loc+1 of that letter
- update new loc. in map, also maintain size of window

map:

0 1 2 3 4 5 6
abcaeba   j jump = ws+1

a : ~~0~~ ~~3~~ 6
b : ~~1~~ 5
c : 2

i,j

window = ~~1~~ ~~2~~ ~~3~~ ~~4~~ ~~4~~ 3

max ~~1~~ ~~2~~ ~~3~~ 4

every i jump
1-(new i - old i)
= new window size

Q String window : mysol :

s1 = hello; world
s2 = lol

~~h~~

h : 0
e : 1
l : 2

if match then if less than i updt i
else push on map

pt next on s2

if i
find in map. if found then delete
and updt : < i updt i, ~~j~~ ~~updj~~

Extramarks

Date ___/___/_____

Page No. _____

PNG sol:

string s1 = "hello";

string s2 = "oel"

w (hashmap) = [h : 1, e : 1, l : 2, o : 1] compare with FP = [o : 1, e : 1, l : 1]

stop once all matched then reduce size, remove useless [h : 0, e : 1, l : 1, o : 1]

Q Smallest distinct window:

"aabcbcdbcaaaad"

co

min : INT_MAX → less than min update

max : INT_MIN → i greater then set max

iterate i=0 to 'o' set at

set m[ch] = i;

if m[ch] != -1

{

m[ch] = -1

min = min(min, i+1);

}

m[ch] = i;

iterate : count all char

a : 5 → 4 → 3

b : 3

c : 3

d : 2

Q → Pascal's Triangle, array size : n

1

1 1

1 2 1

border cond: -1, curr : n-1 an earlier arr

Extramarks

Date ___/___/_____

Page No. _____

→ Submatrix sum : my sol : prefix sum

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

$M[i][j]$ — $M[$ ~~i~~ $i-1][j]$ − $M[i][$ ~~j~~ $j-1]$ + $M[i-1][j-1]$

(above) (left) (left-above)

Q 1, 2, 3 Subset sum

5, 3, 8

5 − 1 = 4 > 0

~~5~~ 4 − 2 = 2 > 0

2 − 3 = −1 < 0

{ 1
1 2
1 3
1 2 3
2 3
2
3

5 − 1 > 0

1, 2, 3

5 − 1 = 4

4 − 2 = 2

5 − 1 = 4

4 − 3 = 1

5 − 2

3 − 3 = 0

1
1 2 , 1 3
1 2 3
2 3
3

2^{n-k} (k pos)

1 2 3 4

1
1 2
1 3
1 4
1 2 3
1 3 4
1 2 4
1 2 3 4
2 3
2 4
2 3 4
3 4
3
4

Date ___/___/_____
Page No. _____

- Subset sum

{1, 2, 3}

f(5-1, [1] 2)

f([2] 2, 3) < 0 stop return Next

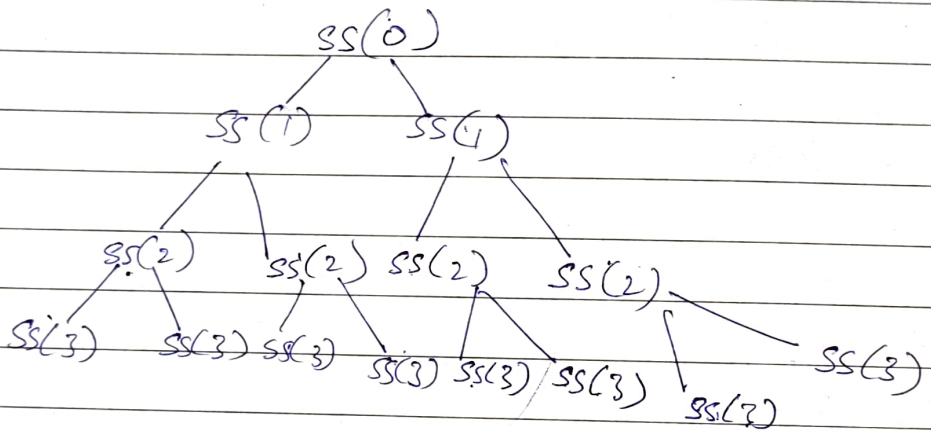
f([2] 4, 3) > 0 i == size stop return null

f(5, [1] 2) → return i → f([2] 3, 3) = 0 return i

→ f(5, 3) >0 i = size stop return null

n = 3 (query), K = 3 (sum size)

i => n times



K = 3, => $1 + (2^4 - 1)$

K = 2 => $7 (2^3 - 1)$

K ... $2^{K+1} - 1$

$O(n(2^K))$

→ ~~Binary strings~~

→ Sliding window max

while(j < size) {1, 2, 3, 1, 4, 5, 2, 3, 6}

i j

if(j - i + 1 != K)

j++

else { maxv = INT_MIN

for i to j }

maxv = max(a[i], maxv)

pushback maxv.

i++

j++

j => 1 to n => n times

i => K times

O(nK)

Date ___/___/_____
Page No. _____

→ cnt subarray with target sum

arr = {10, 2, -2, -20, 10}

k = -10

→ Binary search using recursion

(arr, s, e, k)

mid = (s + e) / 2

if (k < mid)

(s, mid-1, k)

else

(mid+1, e, k)

0 1 (2) (3) 4

1, 3, 5, 7, 9

11

-1

0 1 2 3

1, 3, 5, 7

11

→ 2D array merge :

	0	1	2	3
0	18	4	16	8
1	23	13	20	11
2	28	24	26	25
3	1	30	15	19

→ Inversion count :

count total no. of inversions where $i < j$ & $a[i] > a[j]$

merge func : if $[j] < [i]$, $i < j$, count : $(m - i + 1)$

→ Quickselect :

$[10, 5, 2, 0, 7, 6, 4] \Rightarrow [0, 2, 4, 5, 6, 7, 10]$

k = 4

My sol : used priority queue (heaps) : $(N + k \log N)$

PNG sol : used divide & conquer : $N \log N$

Extramarks

Date ___/___/_____

Page No. _____

→ Smallest string :

a + ab + aba

1 1+2 1+2+1

3 4

use comparator $X+Y < Y+X$

→ ICPC standings (Not done)

1	2	2	1	5	7	7
A	B	C	D	E	F	G
0	1	2	3	4	5	6

rank = i + 1

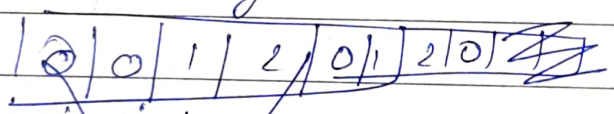
address = S

brute force : for each rank, check address with all

$O(n^2)$

→ Jeggling balls :

my sol:



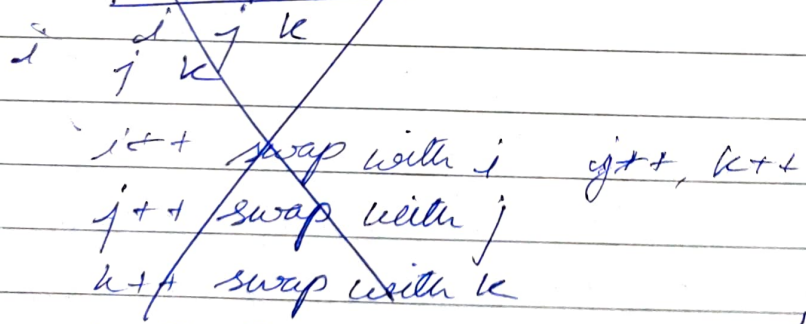
i j k

i++ swap with i

j++ swap with j

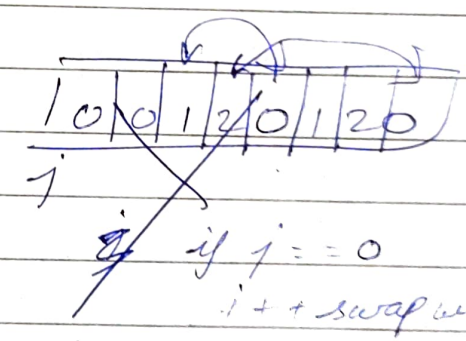
k++ swap with k

i iterate 0 to n-1



i j k

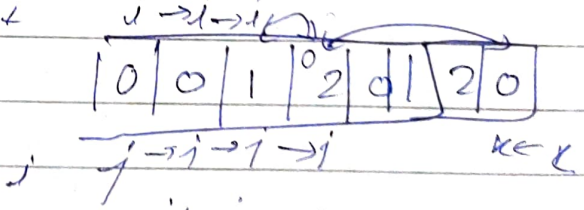
j++, k++



i j

if j == 0

i++ swap with i



i → i → i

j → j → j → j

k ← k

if j == 0

i++ swap with i

if j == 2

k-- swap with k

stop at j == k

since we swapped k we interchanged so check again for i

→ frequency count:
[0, 1, 1, 1, 1, 2, 2, 2, 3, 4, 4, 5, 10]

mysol: key = 2 — k = total appearance
if = 3 — O(logN + K-1)

logN → find ele. : count++
mid-1 = midl → midl--, count++ while a[midl] == key
mid+1 = midr → midr++ " " [midr]

PNG sol: find lower & upper bound by continuously binary searching

→ squareroot using binarysearch:
find int using BS: 0 to n (log N time) N = 10, int = 3
then linear search loop upto 3 times (P) O(qP) (q = range)

→ Angry birds:

→ pair pair:
mysol: sort both O(NlogN)
run on first O(N) (for each ele)
perform BS for each ele (if eq then ans; else put closest)

PNG sol: same

→ Game of greed: concept (cuemath guess on udemy)
search space: monotonic [min: 1, max: 10]
eg: 1, 2, 3, 4
k = 3
BS: search for val where last ele greatest

Extramarks

Date ___/___/_____

Page No. _____

→ Reading books :

[10, 20, 30, 15]

→ Rat & mice

(1,1) 5,4

0 x 0 0

0 0 0 x

x 0 x 0

x 0 0 x

x x 0 0

if i,j == 5,4 return true

else if (i <= 5, j <= 4)

{ if val == x or val == 1

return false

else

{ set i,j = 1,i

if (one(i-1, j) (i, j-1) (i+1, j) (i, j+4)

return true

~~set i,j = 0~~

else { i,j = false

return false

} }

}

// else if (i == 5, j < 4)

// else if (i < 5, j == 4)

else

return false

→ wordbreak :

dict = { i, like, sam, sung, samsung, mobile }

str = "ilikesamsungmobile"

↓

i (func(i, j, str, dict)

str ptr, dict ptr

ilikesamsungmobile

i, func(i, i, str, dict) func(0, i, str, dict)

if str[i] == '\0'

return 1

PNG: (13. "Coin subsets with sum X") → do recursive and put dp → basically subset sum problem

Recursion

1. Subset sum to X → in an array, sub-set of array summing to X & how many are there

↳ eg: X = 6 → a = f(arr, i+1, x − arr[i])
b = f(arr, i+1, x)
return a + b

(my code doesn't decrease)

↳ base case: empty array
if (x == 0)
return 1;
else
return 0;

(right & left always) (my approach recursion call with count of left & right brackets "PNG: 24")

2. Generate brackets → input: 2, two pair of brackets ()(), (()) →))((unbalanced
→ generate proper brackets
→ can use dp in this, (2^n size) recursive call, just check dp table how many outputs generated at open: x close: y if no ans then call else take that value

(same given by PNG)

↳ rules
① close ≤ open
② open < N

↳ base case: (i == 2n): return

(but here we increase L & R, O & C PNG: "60")

↳ recursion: if (open < n)
gen br (output + '(', n, open+1, close, i+1);
if (close < open)
gen br (output + ')', n, open, close+1, i+1);

(my sol: to calculate gen all combinations recursion using ip1, ip2 — ip1, ip2+ip2)

2 ABC, 3 DEF, 4 GHI, 5 JKL, 6 MNO, 7 PQRS, 8 TUV, 9 WXYZ

3. Phone keypad → given number N, gen all strings possible
→ 23: AD, AE, AF, BD, BE, BF, CD, CE, CF

(same given by PNG)

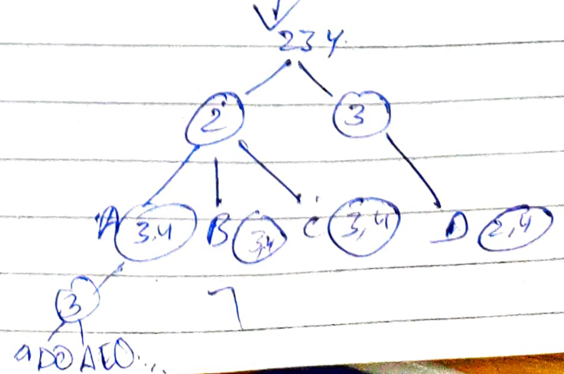
↳ base case:
if (input[i] == '\0') {
~~cout output & endl~~
return

↳ rec case: int curr_digit = input[i] − '0'; → conv to no.
if (0 or 1)
pko (ip, op, i+1);
for (k = 0, < key[curr_digit].size() k++)
{ pko (ip, op + key[curr_digit][k], i+1)

(better than recursion my approach only when no. of inputs are known)

(basically calling function for each letter on that key)

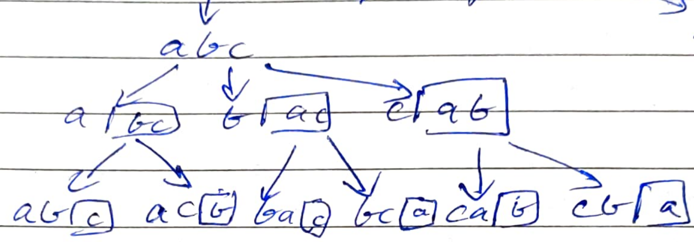
tree



→ PNG: "12."

4. Permutations → given string find all permutations
abc → abc, acb, bac, bca, cab, cba
→ my sol: call func from begin, call func for all letters in string except current and earlier in rec stack
↳ instead of comparing, easier to swap

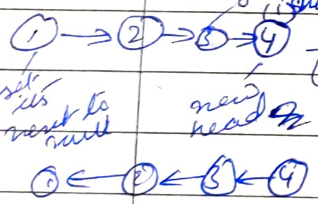
PNG sol → call for letter then swap with all letters on that pos after curr letter
↳ string will be modified so no repeat ↔ pass copies of array else repeat string
or restore original array during backtrack



~~5. Linked list → insert in middle~~ → PNG: "4." in LL folder

35. Recursive reverse a linked list → done iteratively (4 cond., in while loop t, c, p, n)
↳ call on subproblem
Base: if (head → next == NULL or head == NULL) return head

go till here
(i) shead = f(head → next); // call till reach end (second last element)



(ii) head → next → next = head // basically put next ptr of next ele of head on head itself, easiest trick in the book
(need to return new head & set original head's next to NULL (do this for all calls))
(iii) head → next = NULL
(iv) return shead // shead will give new head which is last element

"8. in LL"

6. K reverse a linked list → my approach: do all things same as reverse LL just add a count K & restart reversing from there
↳ PNG sol also same → Reverse LL 2 subprob → join 2 small LL
↳ reverse upto K, call func on subprob. (using reverse LL) → PNG used iteration method
↳ while loop with K count

Extramarks
Date / /
Page No.

"7" Node * c
if (a smaller)
c = a;
7. Merge two sorted LL → c→n = merge (a, b→next)
else
c = b, c→n = merge (a→next, b)
Base case: if a == NULL or b == NULL
put other LL as same

8. Runner Technique → find mid element
(fast) 1 takes jump 2 → fast = head→next→next
while fast != NULL & fast→next != NULL
(slow) 2 takes jump 1 → slow = head → fast = fast→next→next
slow = slow→next

9. Merge sort of on LL → O(n) time to find mid but doesn't matter as overall complexity still (nlogn)
→ all same just runner technique

```
mergesort
{ h == N or h->next == N)
  return head
  node * m = midpt (h);      → same
  * a = h;
  * b = mid->next;
  a = mergesort(a);
  b = mergesort (b);
  return merge (a, b);       → same as 7.
}
```

"10" stacks
10. Balanced parenthesis: push in stack "open" pop upto "open" if closed comes, ignore all other operands

"10" stacks
11. Redundant paranthesis: in between one pair of open and close there should be one operator
same as 10 pop & push just check op so push them and pop as well

12. First non repeating letter: use hashmap + queue
→ add to queue, incr count
pop from queue when count more than 1.

It always seems impossible until it's done.

→ "13" stacks

13. Simplify path:
- → starts with / → absolute else relative
- → /x/y → y which is in x
- → .. → parent directory (eg /x/y.. is x)
- → . → current directory (eg:- /x/y/. is y)
- → symbols together: no consequence /x/y/z = /x//./y/

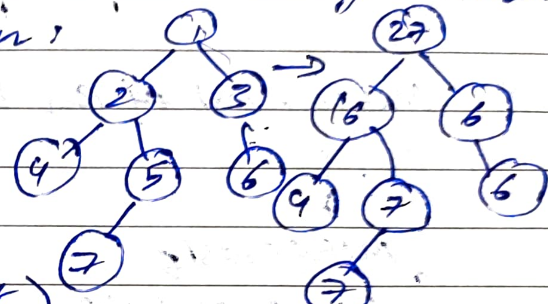
↳ use stack

↳ my sol
- ↳ if .. → pop stk
- ↳ if . → nothing
- ↳ end of path, reverse display a string
  - recur call store in temp & pop, cast from base case & during backtrack

↳ PNG: first clean i/p → remove "." & "" b/w two ~~slashes~~ fwd slash
- ↳ then same

→ "30" descendant sum (trees)

14. Replace with descendant sum:



↳ ~~Root~~ leaf remains same

↳
- → 27 = 2+3+4+5+6+7
- → 16 = 4+5+7
- → 6 = 6
- → 4 = ~~5~~ 4
- → 5 = 7 (only descendant)

PNG Sol: at all stages 3 things needed →
1. temp = node → data (temp to store curr. node val)
2. node data = SR + SL (↳ this is new val)
3. return temp + root data (↳ this gives total sum of child needed)

15. Print at k level: normal level order print strategy use null to mark end of a level

→ done earlier → "28" trees

16. Height balanced BT → like AVL
- ↳ only check: ↳ return 2 things → height of subtree; → root of balanced or not
- return true or false
- ⇓ left bal && right bal and |hL − hR| ≤ 1

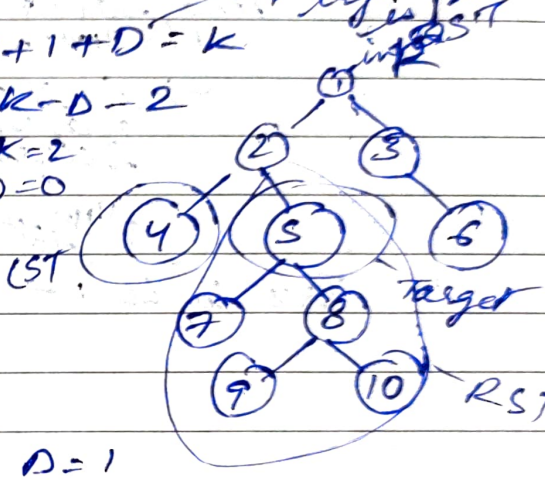
→ "8" trees

17. Max subset sum → find max sum from BT
- ↳ subset: includes a node then can't include parent or children
- ↳ my sol: levels relation → if select node call on grandchild. else call on child → take max
- ↓ PNG also same
  - ↳ inc = root→data + opt(root→left→left, inc...) ...
  - exc = max(opt(child, inc), opt(child, exc))
  - base: root = NULL return 0

Date ___/___/_____
Page No. _____

18. Nodes at distance K from target node
find all nodes at distance K
↳ descendants can be found using traversal use k, reduce at each recursive call, return elements at 0
↳ for ancestors: store all ancestors while searching for node

Case I → if node found in RST then find all nodes at dist "x" in LST



from root of LST $x + 1 + 1 + D = K$

$x = K - D - 2$

$K = 2$

$= D = 0$

$x = 0$

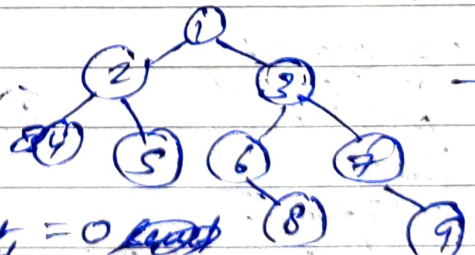
Case IIa → if in LST

Case IIb → from (1) root, D = 1

if (D+1 == K)
return node
else if Target in left
call right (K − D − 2)
else Target in right
call left (K − D − 2)

"Vertical print"

19. Vertical order print:



→ 4, 2, 1, 5, 6, 3, 8, 7, 9

↳ my LPNR
same recursive call, root (level) = 0
L = l − 1
R = l + 1
same in map → print map in ascending order
↳ map<dist, vector<int>>
↓
at each dist store val of all nodes

Extramarks

Date ___/___/_____

Page No. _____

"8 Trees"

20. Minimum height BST : AVL tree → take mid & put element in L & R

↓

S to m-1 in LST

m+1 to e in RST

"32 Trees"

21 Closest in BST : → my sol : maintain prev, choose node b/w prev & current which is closest

↳ PNG sol : maintain diff : if ≥ 0, go left

if < 0, go right

↳ var < closest diff

↳ in the end return closest so whenever hit NULL, always return closest in whole path traversed

"16 Trees"

22. Tree to LL : → recursive call left & right ST return H, T from both & create new if - only LST : L.H & R.T

only RST : R.head & R.T

both : L.H & R.T

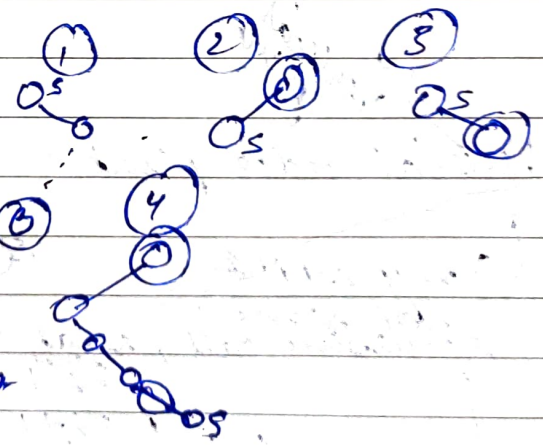
base case = root = NULL

if no ST then root itself

LT & RH are used to connect to root

"25 Trees"

23. Inorder successor →



↳ not use closest since actual val present

↳ PNG sol :

↳ RST exists

↳ go extreme left in it

↳ if not exists :

store all nodes > larger < curr node in path

Extramarks

Date ___/___/_____

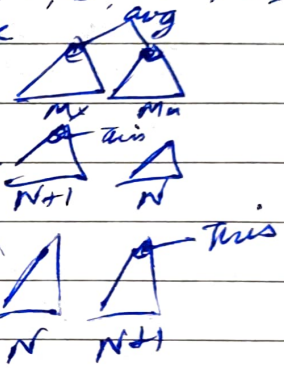
Page No. _____

24. Merging ropes → {4, 3, 2, 6} → minimize cost
"1" heaps — ropes size
sol
create min heap
pop top 2, calc & push back
repeat

4+3 = 7
7+2 = 9
9+6 = 15
= 31 (cost)

2+3 = 5
5+4 = 9
9+6 = 15
= 29 min cost

25. Running median → write median after insertion of each no
"2" heaps
BNB sol
10, 5, 2, 3, 0, 12, 18, 20, 22
10, 7.5, 5, 4, 3, 4, 5, 4.5, 10
→ 2 heaps → min & max
→ 2 cases (no. of elements) → even → add
"heaps"



26. K sorted array merge: 0, 2, 5
3, 6, 9
8 11 15
K = 3
- create K size min heap: push, ~~min~~ ele. → pop min
TC: total elements = NK
push + pop = 2 × log K
→ NK log K

27. Triplets in GP: count no of triplets such that in GP
→ i < j < k : for a given common ratio r
→ 2 representation / a, ar, ar² / a/r, a, ar → we use this
→ 2 maps per each / freq left (a/r) / freq right (ar)
→ sliding hash algo
multiply both val to get GP

| 4 | 2 | 4 | 4 | 16 | 0 |

Right = {2:1, 4:2, 16:1}
left = empty
as you move reduce right ↑ left

28. Count rectangles: → axis parallel rectangles
→ N cartesian pts given $P_1(x_1,y_1)$, $P_2(x_2,y_2)$
↳ take diagonal, see if other two present
↳ use hash
"hash" $P_2(x_1,y_2)$ $P_4(x_2,y_1)$
look for P_2 & P_4 → two counts so divide by 2

29. Count triangles: → N pts, find no. of triangles ∥ or ⊥ to x or y axes
↳ just like rectangles ↳ right angle
↳ for each x, y find all (_, y) & (x, _) ↳ multiply both form that many Δs
↳ 2 unordered maps $(\text{cnt}_x - 1) \times (\text{cnt}_y - 1)$
$\langle x, \text{cnt}_x \rangle$
$\langle y, \text{cnt}_y \rangle$
→ part of str. but together

30. Anagrams & Substring:- subsequence: can be seperate abcde: bcd: substr ade: subseq.
↳ anagrams: when letters of word rearranged to form another word
↳ given str: find no. of substr anagram of each other
↳ if use hash each substr in form 26 binary bits $B[1100\ldots0_2]$ for abc & compare the map val - const time to check if anagram
↳ now club all same hash values together as a bucket for easy calc.:
eg bab, bba, abb
↳ hash final (key, int)
↳ not need substr.
↳ finally for each int val do: $^{int}C_2$

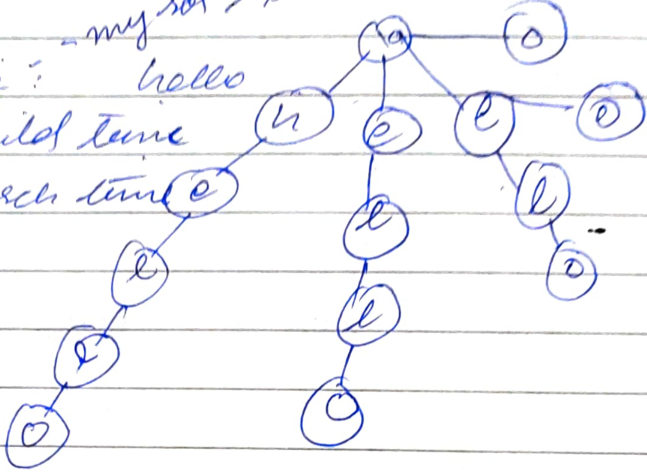
31. Quick brown fox → str. w/o spaces
↳ sa: quickbrownfox → place min. no. partitions, all words found in dictionary
words = ["the", "quickbrown", "fox", "quick", "brown"]
min partition = 2

→ sol: recursion + hash + DP

↳ DP: you put partition and go on & you don't put partition & go on.
↳ use hash after each letter if present or not
↳ unordered set: since only need T/F not count
↳ basic concept as done in DAA with helper functions

my sol → trie → suffix trie

32.→ Suffix trie: hello
O(N^2) build time
O(L) search time



33.→ Little cute cat: creative thinking & use of DS
↳ cat given list of words, check if present in readme doc
↳ word can be substr of a word, would be counted as present
↳ so "only hashmap" won't work
→ brute force: check each word in doc.
↳ suffix trie: → D^2 to build, WN to search → slightly better than brute
↳ sol: O(NW + DW)
↳ make trie of given word
↳ search ~~word~~ of doc. in it.
→ from each letter little
(l then i then t ...)
↳ finally: iterate over hashmap to see which words marked true
→ for each w: words, check in map

34. Biggest XOR: max. XOR you can form by picking two nos. ⇒ I/P {3, 10, 5, 8, 25}
O/P: 28 : 25 & 5
↳ brute force: O(n)
↳ O(n): using trie → we need 1s w. most bits on left

00101
11001
11100 ⇒ 28

↳ make trie of all no. then take binary of a no. & look for exact opp. in trie. Eg:- 25: 11001, opp 00110, take ~~whatever~~ closest

1000 > 0111

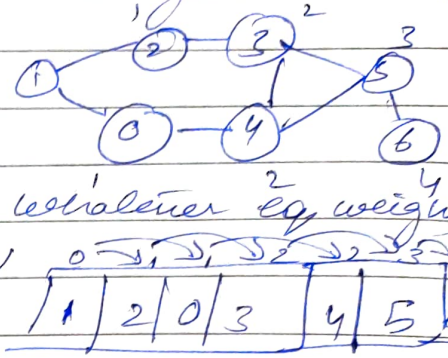
since length of all bits 32/64
looking for same is O(1)
since l = const

Extramarks
Date ___/___/_____
Page No. _____

→ approach almost same in all 2d matrix

* usually grid questions can be solved using graph & DP approach

35. Shortest path using BFS: when equal weight (1)
↳ just like level: whenever a node first visited, it would be its distance
↳ bfs: done using queue, everytime children inserted, since their dist by 1 (or whatever eq weight)
↳ dist node = dist(parent)+1
↳ update dist of child when you push it in queue

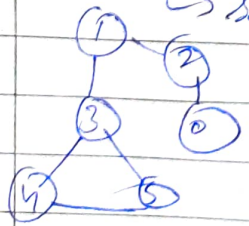


→ "graphs" → snake & ladder ⇒ PNG codes

36. Snakes & ladders: vector containing snakes & ladders
vector<pair<int,int>>
↳ board size N, give min. no of dice throws to reach end
↳ can easily do with DP
↳ but here directed graph 6 edges from each based on no. of dice val
↳ unweighted (since all move to cost 1 dice throw)
1 → 2 is 15 (ladder at 2)
1 → 3
1 → 4
1 → 7
1 → 6
↳ first: for all nodes insert all edges, & check if ladder then add that value as well
↳ run BFS to find shortest path to destination
O/p cost

→ code using DFS

37. Cycle detection in undirected graph: if no cycle then tree
↳ my sol: just do ~~BFS or~~ DFS, mark node as visited
↳ if any visited node again then cycle
↳ same sol by PNG: just also check nbr != parent (node)



if !visited set parent
if visited not just return, first check parent of current if nbr who is visited

in this case no cycle yet 0 will be shown visited when nbr of 4 is pushed, but 0 parent of 4
↳ both case BFS & DFS need to check this

Extramarks

Date ___/___/_____

Page No. _____

"detect cycle in graph" → PNG

38. Backedge detection: directed graph → DFS

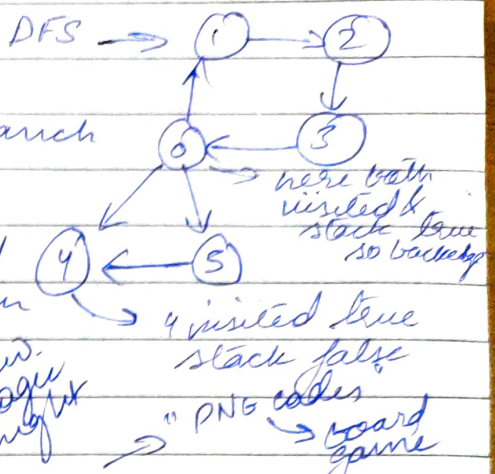
↳ we maintain a stack

↳ whenever DFS, all of same branch pushed in stack & set T, when come out of branch, set all false of branch.

↳ this way: if both visited & stack true then backedge.

↳ TC: O(V+E)

★ would it work in interview asked in interview coz logic might



→ " PNG codes → board game

39. Board game: M×N board → each cell one char

↳ find wether words in list present or not

↳ take adjacent cells to find (all directions) can't use a cell more than once.

↳ Eg:-

S	E	R	T
U	N	K	S
T	C	A	T

words = { "SNAKE" ✓, "FOR" ✓, "QUEZ" ✗, "SNACK" ✓, "SNACKS" ✓, "SENS" ✗, "TUNES" ✗, "CAT" ✓ }

↳ Brute force: for each char, make all words from it

↳ my sol: make a trie of all words, iterate over it & DFS in board to see if can be formed

↳ PNG: almost similar. make trie: pick a char in board, check for each & nbr if can go there acc. to trie for that char., if not then word not present if trie at terminal then word present

↳ this won't need hashmap

↳ 8 way DFS, trie guides search

may be will need hashmap to look for children of certain char.

40 Shortest grid path: → 2d grid, each cell contain cost to travel thru it. find min cost to reach bottom right from top left

↳ my sol: use DP

21	100	65	12	18
10	13	17	157	6
100	113	174	11	33
88	124	41	20	140
99	32	111	41	20

sol: 327

↳ we not exist (★ dead ~~generally~~ for dykstra)

↳ same in 4 ~~dykstra~~ direction

↳ PNG sol: straight dykstra ques

Extramarks

Date ___/___/_____

Page No. _____

TC & SC: M×N → since at most visit all cells of table; once visited never repeat

41. largest island : grid 0 & 1, 0 water, 1 piece of land

→ find largest island (most 1 are together)

↳ my approach : DP

↳ PNG sol:

↳ movement ↑↓→←

↳ 4 way DFS

↳ from each node & mark visited

↳ do for all ~~visited~~ unvisited nodes

↳ go to a node during DFS only when it is 1

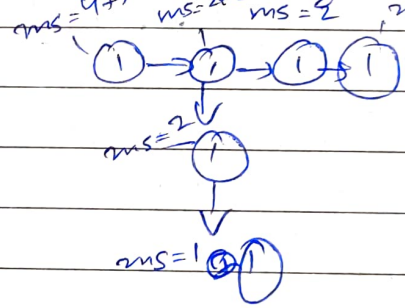
```
1, 0, 0, 1, 0
1, 0, 1, 0, 0
0, 0, 1, 0, 1
1, 0, 1, 1, 1
1, 0, 1, 1, 0
```

2 cond: val = 1 & ~~not~~ node not visited

↳ of all island values {2, 1, 8, 2} in above example

↳ select max one → not store all just take max

↳ at each call return size



42. Graph sequence → takes 2D matrix

↳ increasing path : ~~get~~ strictly greater than previous

↳ 4 way ↑↓←→ DFS

↳ my sol : DP (given →) ~~just like largest island~~

↳ find max path

↳ call 4 way DFS

↳ PNG : try every pos : use DP to store max path from each node if come in path then add its val, same like my sol.

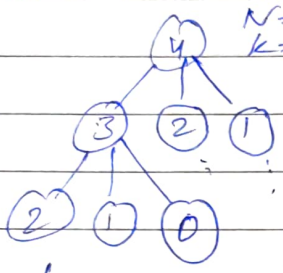
Date ___/___/_____
Page No. _____

DP : → top down : recursion → start from final opt that needs to returned
eg :- Opt(n, W) , call on n-1
fib(n) = fib(n-1) + f(n-2)

→ Bottom up : table format using 2 loops → i & j
eg :- fibonacci, i = 0 to n
fib[i] = fib[i-1] + fib[i-2]
fib[0] = 0
fib[1] = 1

"PNB" DP → NK ladders

43.- N-K ladders : no. of ways can climb N size ladders



N = 4
K = 3

↳ can take steps ∈ of atmost k
↳ f(n) = f(n-1) + ... f(n-k)
↳ base case : when n = 0, return 1
↳ use DP, optimal substructure
↳ top down
↳ bottom up : 1 step 1 way, 2 ways, (2 from 1, from 2)

1	1	2	4	7
0	1	2	3	4

0 step 1 way ← substract all k & add their val

↳ dp[N] = dp[n-1] + dp[n-2] .. dp[n-k]
↳ 3rd approach : optimise bottom up
↳ using subarray sum
2 DP[n-1] - DP[n-k+1] → O(1)
↳ so final O(n)
↳ no more k computation
↳ bcz in given k, it is also const, bcz if k input then prob

TC, SC : O(NK)

"PNB" → DP "Rod cutting"

44. Rod cutting problem : Rod of size N → cut ~~k~~ atmost size N, each ~~N~~ profit assoc. w. it, find max profit

↳ eg :- N = 8 = k

1	2	3	4	5	6	7	8
1	5	8	9	10	17	17	20

Max profit = 22 → 6 + 2

↳ recursive & DP call for each f(n-1) ... f(n-n)
take max

Top down → Opt(n) = Max (Price[i] + Opt(n-i)) ↓ for i to n select

Bottom up → for len = 1 to n
for curr = 1 to len
ans = max(ans, price[curr] + dp[len - curr])

↳ for each len, take max of remaining, total cuts upto len can be taken + dp of remaining

45. Array jumps : array of +ve integers, each element shows almost jumps can be taken from that no.
↳ find min no. of jumps
↳ eg :- [3, 4, 2, 1, 2, 3, 7, 1, 1, 1, 2, 5]
O/P = 4
(My soln) 3 → 4 → 2 → 7 → 5
↳ Recursive + DP (Top down)
↳ for an ele, call for i+1 to arr.length
opt(i) = min (1 + opt(j))
↳ save min val for that ele. in DP[dp[i]]

base case :
if (i == n-1) {
return 0;
}

"frog jump" PNB DP

16. Frog's min cost :
↳ n stones
↳ jump cost : |hi - hj|
↳ find min cost
↳ recursive DP : either take stone or take another
↳ 30 10 60 10 60 50
0 1 2 3 4 5 6
|30 - 60| + |60 - 60| + |60 - 50| = 40

for j = 0 to n
opt(i) = min(|h - hi| + opt(j))
dp[0] = 0
dp[i] = ?

for i = n-1 to 1
opt(n) = min(|hn - hi| + opt(i))
↳ when can jump anywhere

bottom up
↳ here only 2 jumps
↳ base case :
dp[0] = 0
dp[1] = |dp[1] - dp[0]|
dp[i] = min { dp(i-1) + |h[i] - h[i-1]|, dp(i-2) + |h[i] - h[i-2]| }

Top down : opt(n) = min { |h[n-1] - h[n]| + opt(n-1), |h[n-2] - h[n]| + opt(n-2) }
↳ base case : opt(0) = 0

47
48. Max non adjacent sum → arr of +ve integers, max sum of non adjacent ele

my sol:

eg:- 6, 10, 12, 7, 9, 14 : order matters

opt(n) = max { arr[n] + opt(n-2), opt(n-1) } ans 32

↳ top down

↳ bottom up (DP): dp[i] = max { arr[i] + dp[i-2], dp[i-1] }

↳ base case: if n ≤ 0, return 0 — we taking

↳ base case: dp[0] = arr[0]

dp[1] = max(dp[0], arr[1])

"box" "new" → DP

stacks

49. Box stacking → array of boxes (W, D, H) → in this order

↳ strictly smaller W, D, H → similar to LIS

↳ eg:- (2, 1, 2), (3, 2, 3), (2, 2, 8), (2, 3, 4), (2, 2, 1), (4, 4, 5) → 3 boxes

find largest size stack

↳ first sort based on height

↳ [(2,2,1), (2,1,2), (3,2,3), (2,3,4), (4,4,5), (2,2,8)]

y = [1, 2, 5, 4, 10, 8] → DP when stack, biggest stack earlier.

↳ to box that can fit

counting trees → DP

gate formula → distinct BST: $\frac{2nC_n}{n+1}$ ⇒ $n=3 \Rightarrow \frac{6C_3}{4} = \frac{20}{4} = 5$

search

50. Counting trees: → count the no. of binary trees

↳ formed using N nodes → numbered from 1 to N

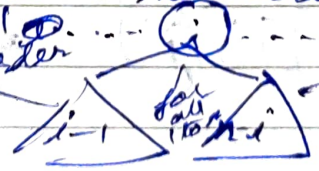
↳ eg:- 3 → 5

4 → 14

5 → 42

↳ Top down

$f(n) = \sum_{i=1}^{N} f(i-1)\, f(n-i)$



can be in any order

$= {}^{x}C_1 \cdot {}^{y}C_1$

$= x \cdot y$

$= f(i-1)\, f(n-i)$

↳ bottom up.

dp[0] = 1

dp[1] = 1

for (i = 2 to n)

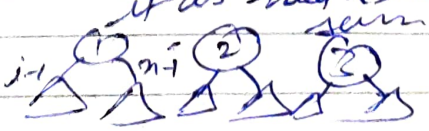
{ for (j = 1 to i)

dp[i] += dp[j-1] * dp[i-j]

}

this thing in iterative form

if n = 3 then for each elements (1,2,3) how many trees can be formed with it as root & then sum



"LCS" → PNG → DP

50 Longest common subseq: → given 2 str
find max size subseq (dont need to cont.)
asked in DAA paper
just like genetic seq.

→ APNG: exact same

my sol: (Top down) inc = exc = 0;
if X[i] == Y[j]
inc = Opt(X, Y) = 1 + Opt(X+1, Y+1)
else
exc = max { Opt(X, Y) = Opt(X+1, Y) shift X
{ Opt(X, Y+1) = Opt(X, Y+1)
→ final return max(inc, exc)

→ base case:
S1 == 0 or
S2 == 0

→ eg:- S1 = ABCD
S2 = ABEDG
⇒ ABD(3)

→ bottom up: S1

S2

	0	A	B	C	D
0	0	0	0	0	0
A	0	1	1	1	1
B	0	1	2	2	2
E	0	1	2	2	2
D	0	1	2	2	3
G	0	1	2	2	3

→ means in A & AB how many matches (in S_2, S_1)

← filled myself ✓ correct

S_2 & S_1 format means in A & A how many matches

means AB & A (S_1, S_2) how many matches

→ dp[i][j] = 1 + dp[i-1][j-1] if match
dp[i][j] = max { dp[i-1][j]
{ dp[i][j-1]

"countsubseq"
→ PNG → DP

Extramarks
Date ___/___/_____
Page No. _____

51 Counting subseq :- how many times S2 occurs in S1
↳ my sol:
↳ same as LCS but instead if else :
call both & sum all
↳ Base case when $S_2.b == 0$ return 1,
$a = b = c = 0$ else return 0

```
count(x,y) {
    if (x == y)
        a = count(x+1, y+1)      x = S1
    b = count(x+1, y);           y = S2
    ~~c = count(x, y+1);~~ (no need to shift S2.
    return a+b~~+c~~;
}
```

↳ PNG ~~direct~~ same ☆
↳ $f(i,j) = f(i-1, j-1) \longrightarrow S1[i] != S2[j]$
$f(i,j) = f(i-1, j-1) + f(i-1, j) \longrightarrow S1[i] == S2[j]$
↳ Base case : $i == -1$ & $j == -1$ or $j == -1$
return 1
$i == -1$
return 0
no else case or ~~for all val of~~ i,j return 0

↳ bottom up

↳ for i to m
for j to n
{ if (a[i-1] == b[j-1])
dp[i][j] = dp[i-1][j-1] + dp[i-1][j]
else
dp[i][j] = dp[i-1][j];

	∅	A	B	C
∅	1	0	0	0
A	1	1	0	0
B	1	1	1	0
C	1	1	1	1
E	1	1	1	2
	1	1	1	2