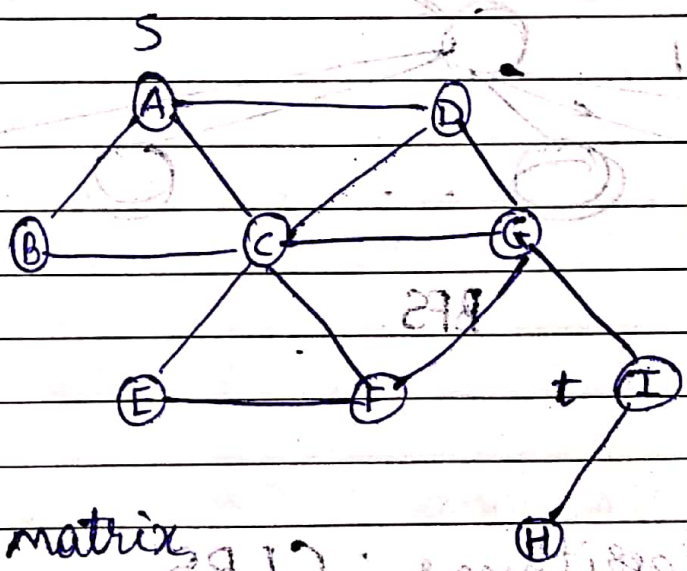
Artificial Intelligence

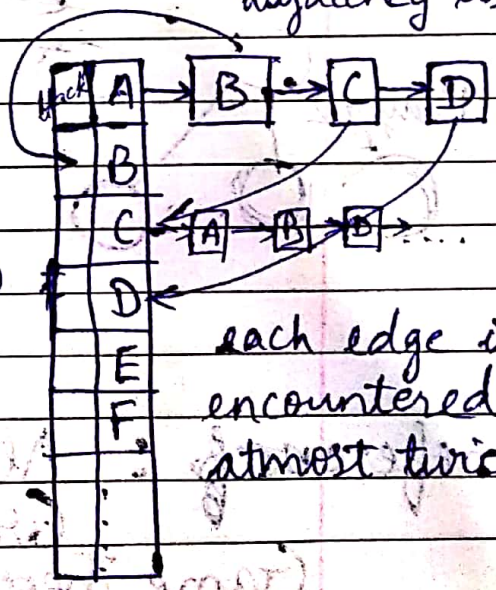
09/01/20

Binary Search
Tree Search
Graph Search



Breadth First Search

adjacency list



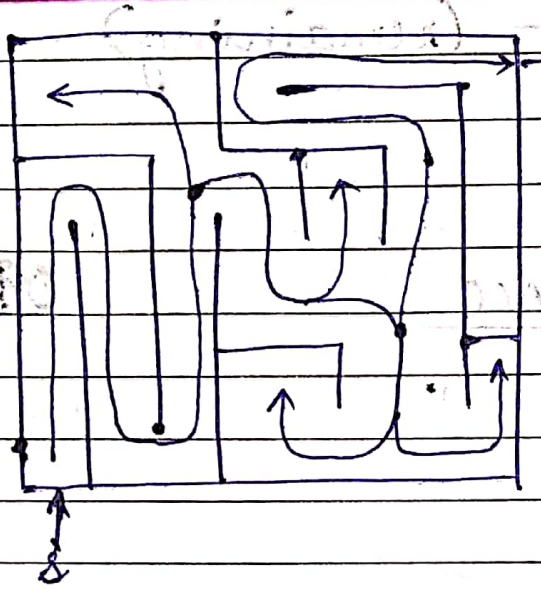
each edge is encountered atmost twice

adjacency matrix

	A	B	C	D	E
A		1			
B	1				
C				1	
D			1		
E					

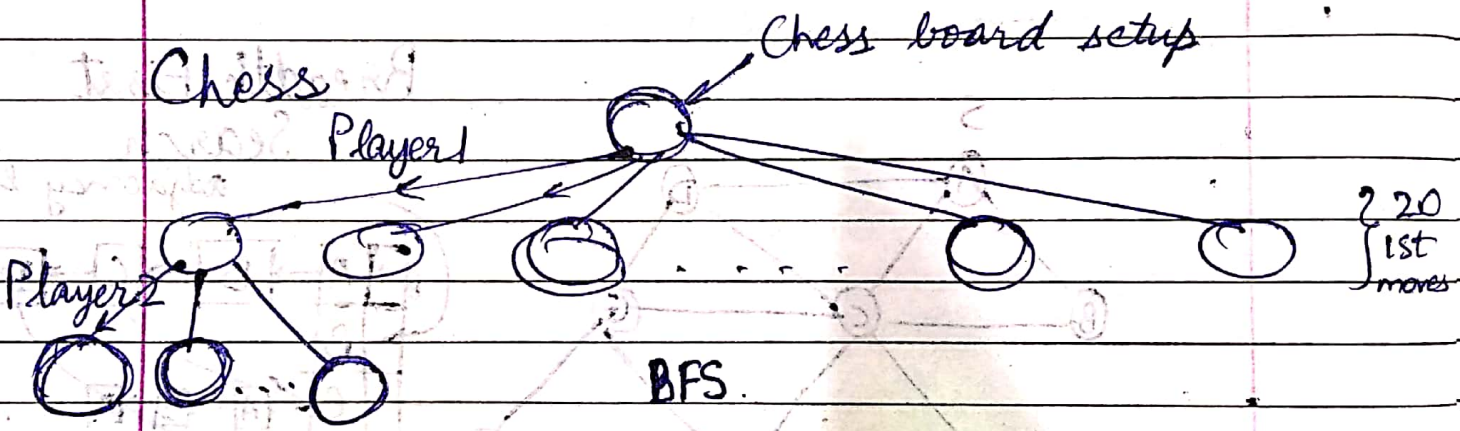
unvisited vertex → white
visited and exhausted vertex → black
visited and active vertex → red

$O(|V| + |E|)$ (i.e. $|V| \leq |E|$) connected
↳ bcoz state space/graph is ^



Depth first Search (DFS)

$O(|V|^2)$



references : Algorithms : CLRS
Graph Theory : Douglas West
Game Theory : Algorithmic Game Theory (CMU lecture notes)
AI : AI, a modern approach Peter (Norvig), Stuart J. Russell

Assignments: Groups: 5-7 per group

loops,
conditionals,
pointers,
arrays,
functions,
recursions,
BFS, DFS.

Assignment 1: Program a clever interactive tic-tac-toe player

Only single program/file. No modules can be used

Stop when someone wins or when empty cells are exhausted

```
int victory_check(---)
cout << ....
arr[3][3]

int sum=9
int flag=0
while (sum>0 and flag=0)
{

}
```

Make an optimal program s.t. the player never loses for full marks.

Page
Date

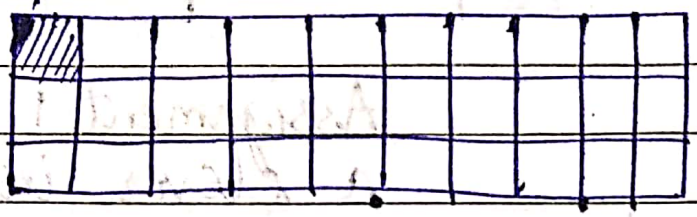
One who eats this loses.

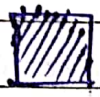
Prime

Chomp

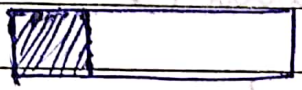
① bite off a rectangle

② leave a rectangle

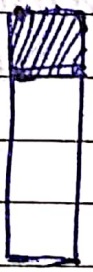




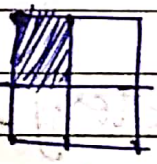
Player 1 loses



Player 1 Wins

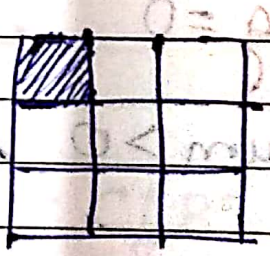


Player 1 Wins



Player 2 Wins

Player 1 Wins



Player 2 Wins

General Formula:

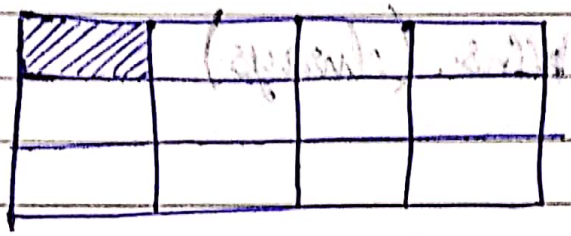
when bar is not a square : Player 1 Wins

" " " square : Player 2 Wins

last Week Q: height of AVL : $w \log n$

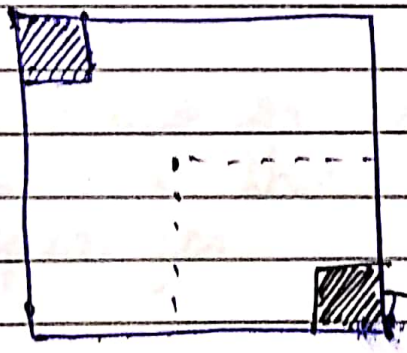
Crazy Chomp ① bite off everything to the right and below of a piece.

4 x 3



→ Proof that 1st player always wins.

Assume Player 2 has winning strategy ↗ there exists.



if ∃ a winning strategy for player 2, player 1 can bite it off in 1st move.

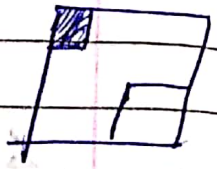
→ Assume this as player 1 first move.

Same Proof is valid for Chomp.

Chomp:

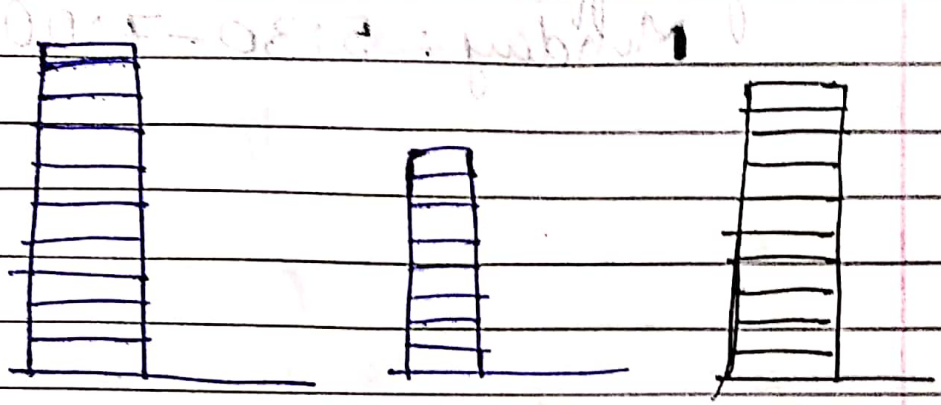
bite such that in total all pieces to the right and below a piece has to be bitten off.

final structure after each move:



Player 1 Wins. (always)

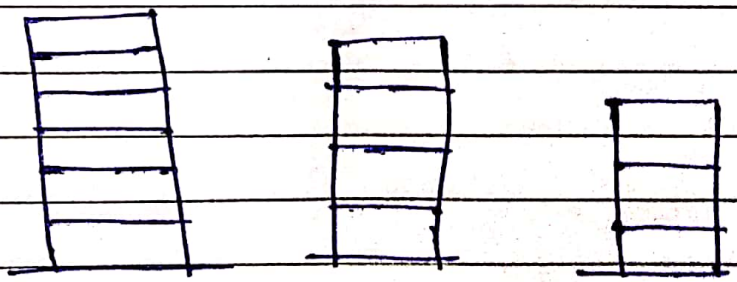
Nim:



1) Players alternate in removing the coins.
2) Player may remove as many coins as he wants but only from a single heap per move
3) Player to remove the last coin wins

if two heaps are equal the first player removes the third heap and becomes the equalizer.

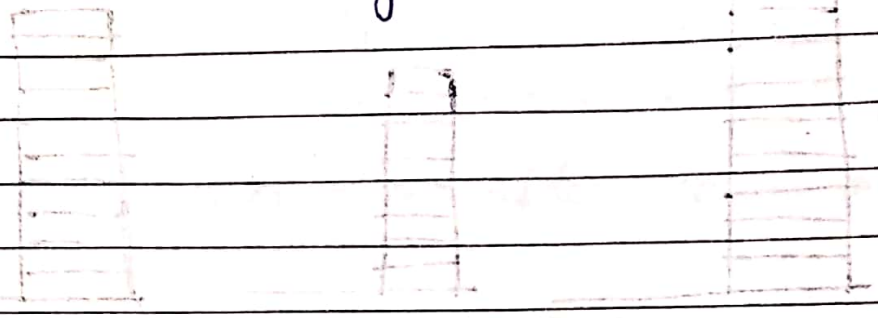
ex 6 4 3

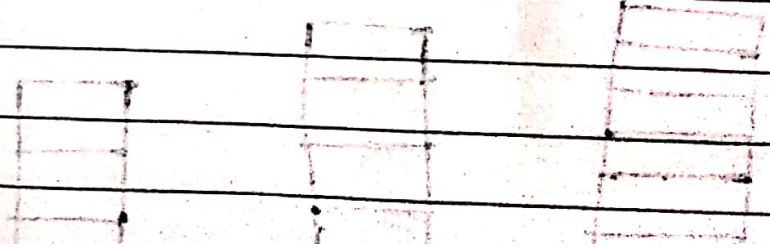


1) Make the XOR zero → Invariant
2) Manipulate the LARGEST heap → Method

Coding Difficulties:

Mohday: 5:30-7:00 p.m.





Assignment :

if (2, 2) = blank
occupy

We start :

① Start with (2,2)

if reply' = diagonal || non-diag.
reply = adjacent diagonal

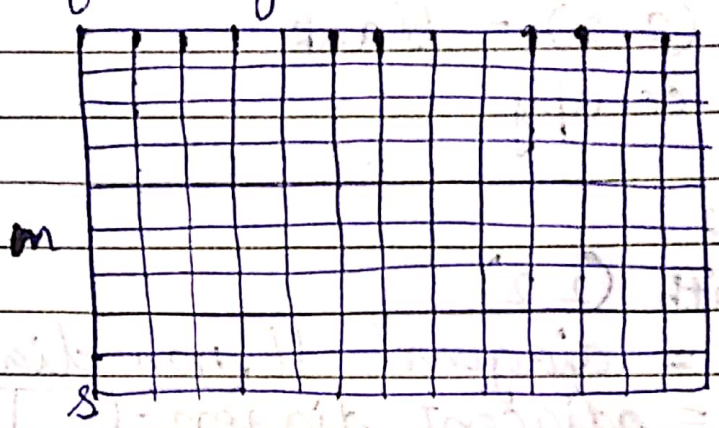
consider special case for winning strategy

Player 2 Starts

① if (2,2)
reply = diag.
& check for 3 (2 in a line) on each step.

Page
Date 10/01/20

Path finding



no. of shortest paths : $\binom{m+n}{n}$

→ Best first search
→ A* search

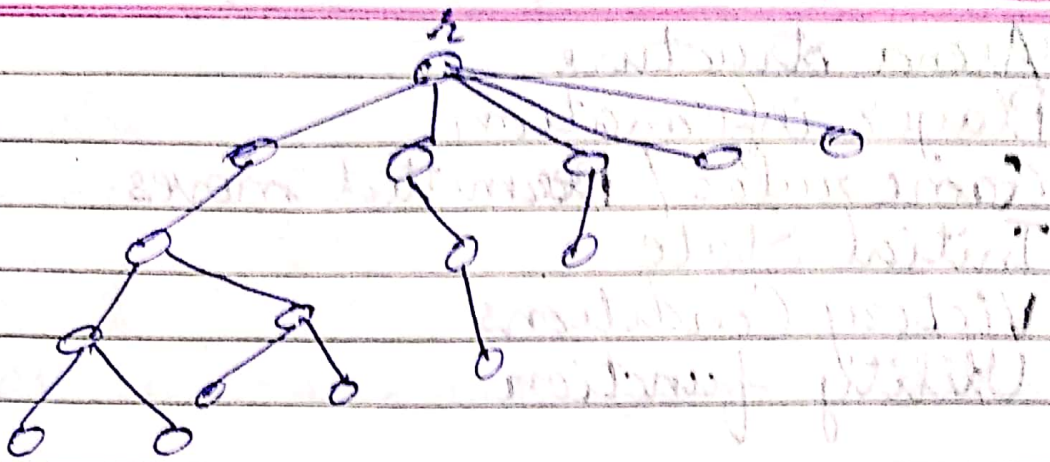
Given a graph, eccentricity of a vertex v is the longest path among the shortest paths from v to all other vertices

Radius of G: smallest eccentricity
Diameter of G: largest eccentricity.

Q. Given a tree, find its diameter efficiently.

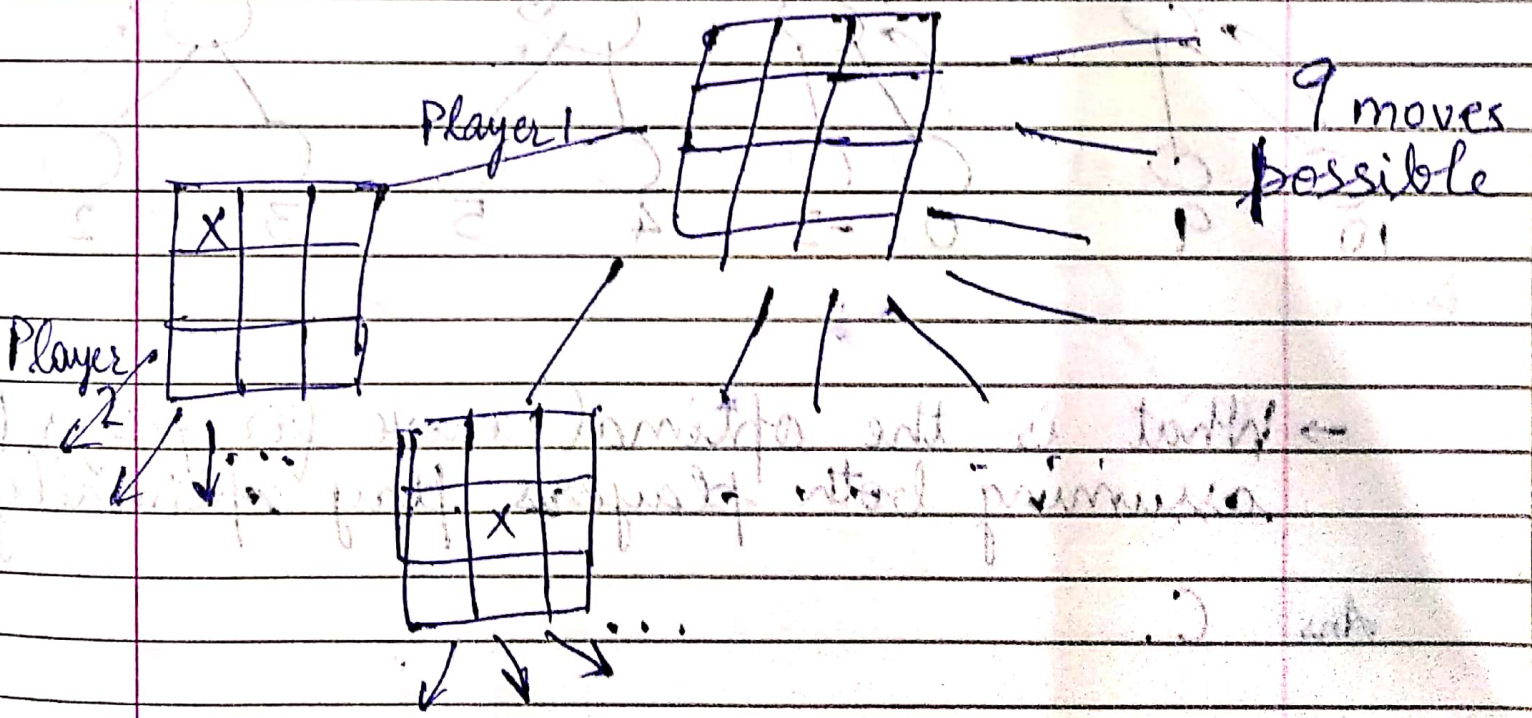
$O(|V|(|V|+|E|))$

d.f.s. on each vertex



the leaf farthest from the root is one end-point of diameter.
$O(|V|)$ required, to find the diameter.

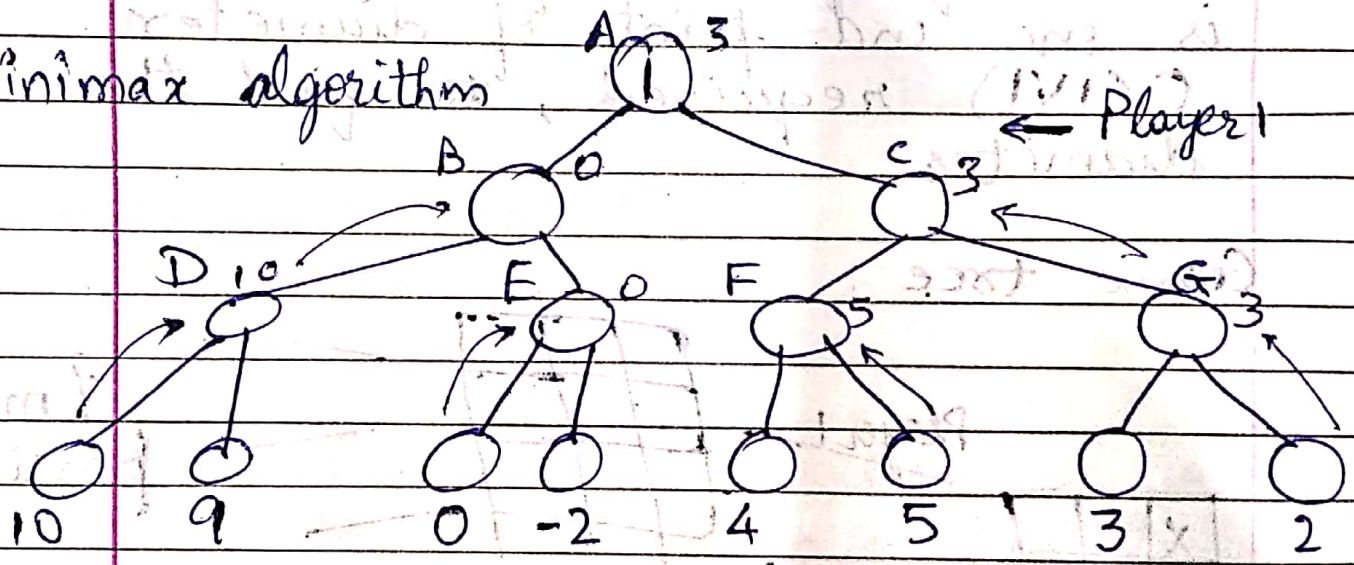
Game tree:



Arena structure
Player information
Game rules / Permitted moves
Initial State
Victory Conditions
Utility function

Adversarial Search

minimax algorithm



→ What is the optimal move for player 1 assuming both players play optimally?

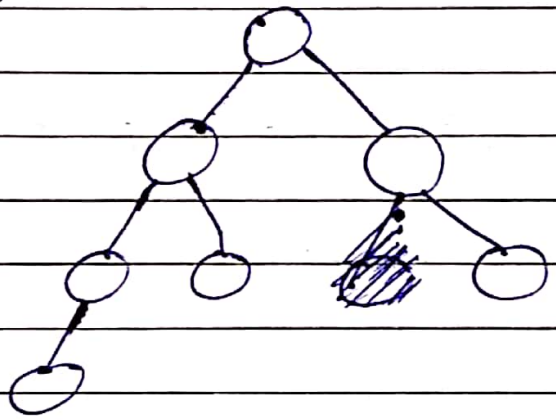
Ans C.

~~complete~~ AVL

Q Take balanced binary tree, such that height difference between adjacent nodes is at most 1.
For complete binary tree, $h = \lg n$.
Is it possible for such a tree to have height much greater than $\lg n$.

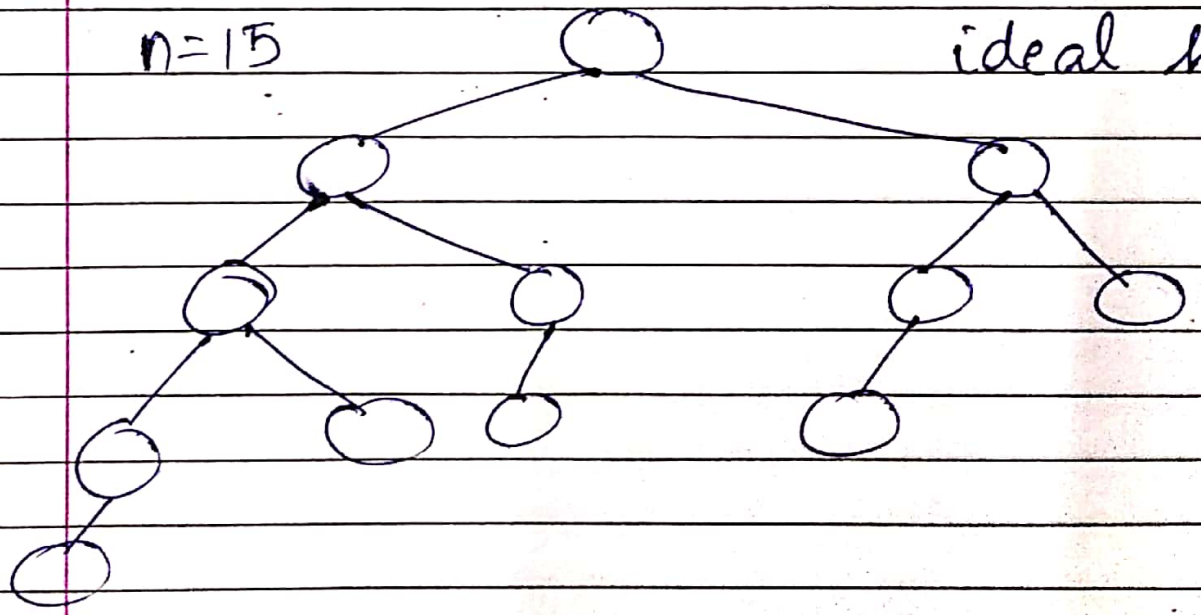
Sol

$n = 7$



ideally: $h = 2$
Here: $h = 3$

$n = 15$



ideal $h = 3$

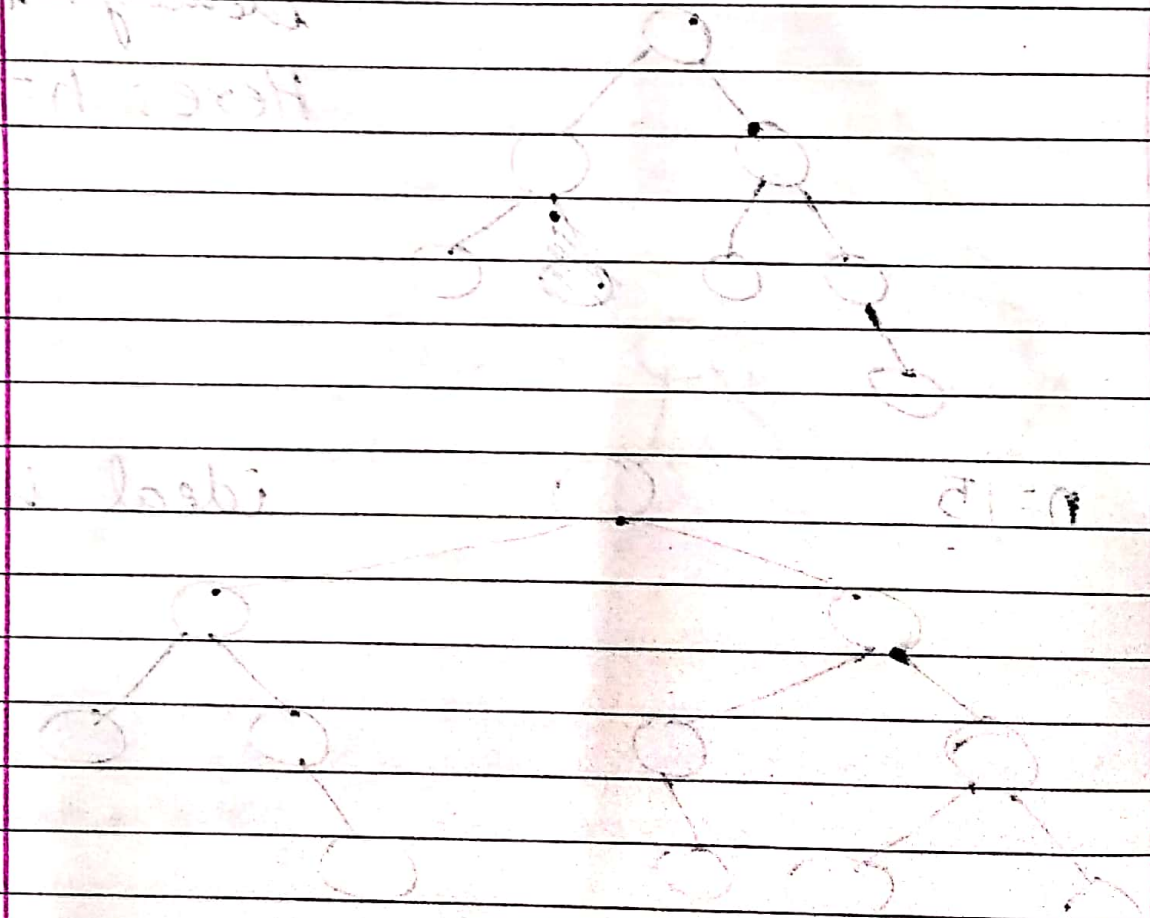
$h = 4$

The smallest AVL tree of height $h-1$ has less vertices than smallest AVL tree of height h.

Page
Date

Midsem:

1. Coding
2. Algorithmic Game Theory
3. Graphs
4. Whatever has been taught in class.



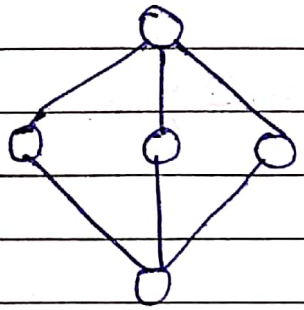
Page
Date 16/01/20

Install Ubuntu

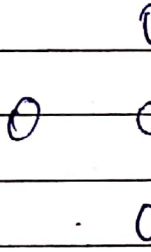
Q Party : 6 people
3 mutual friends
or 3 " strangers

Model it using a graph.

S

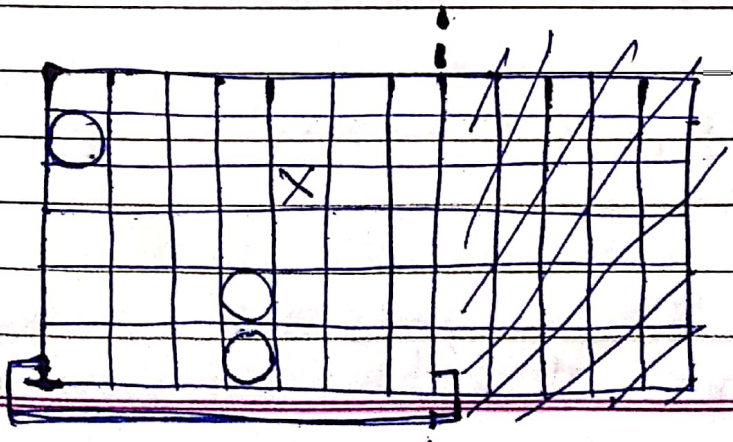


] mutual friends



] mutual strangers

Connect 4 Game (advanced version of tic-tac-toe)
Plot 4



Game theoratically, 1st Player has the winning strategy

Write a program for 2 players to play connect 4.

```
int main()
{
    int board[6][7];
    int i=0, j=0;
    while (i<6)
    {   j=0;
        while (j<7)
        {   ar[i][j] = 0;
            j= j+1;
        }
        i= i+1;
    }
    interact(board);
    return 0;
}
```

do not make a single function too long break it into multiple functions.

```
void interact(int board[][7])
{
    int ar[7]={5}; a, b, count=0;
    while(result(board)!=1)
    {
        cout<<"Player "<<count%2<<" move: "<<endl;
        if(count%2)cin>>a;
        {   board[a][ar[a]]=1;
        }
        else
        {   board[a][ar[a]]=2;
        }
        ar[a]--;
        count++;
    }
}
```

it is optional

```
void interact(int board[6][7])
{
    int full=0; pl=1;
    int win=0;
    while(full<42 and win==0)
    {   cout<<"Move for player "<<pl<<endl;
        choice(board, pl, full, win);
    }
}
```

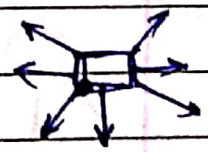
→ let choice call everything by reference.

aimove() can be added to implement computer's move.

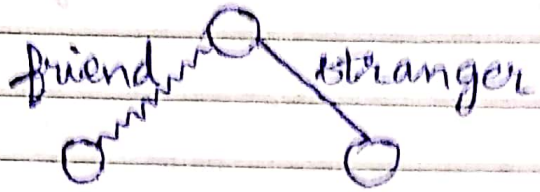
```
void choice(int board[6][7], int pl,
            int full, int win)
{
    if(wins(board)==1)
    {  win=1;
    }
    full++;
    pl^=3;
}

void choice((&board)[6][7], &*pl, &*full,
            &*win)
{
    int move;r;
    cin >> move;
    if(move>7 or move<1)   //column exists or not
    {  cout << "column number exceeds
       the scope of the board" <<endl;
       cout << "Move for player" << pl
            << endl;
       choice(board,&pl,&full,&win);
       return;
    }
  else if(ar[0][move-1] >0)  //column full
    {  cout<<"Column Full" <<endl;
```

```
cout << "Move for player" << pl << endl;
choice (board, &pl, &full, &win);
return;
}
else
{   r=5.
    while (board[r][move-1] >0)
        r=r-1;
    board[r][move-1] = pl;
    victory ((&board)[6][7], &win);
    full = full +1;                    ,&r, &move
    if (pl==1)
        pl=2;
    else
        pl=1;
}
}
```

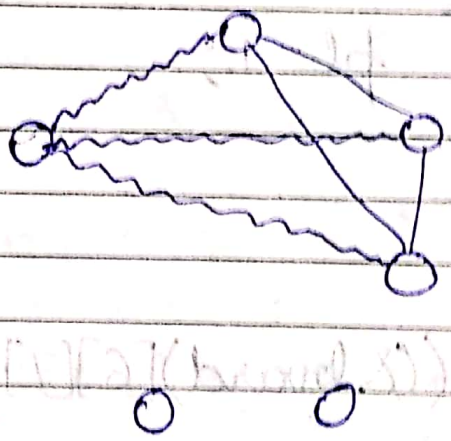
```
void victory ((&board)[6][7], *win, *r, *move)
```



6 people puzzle



PROOF:



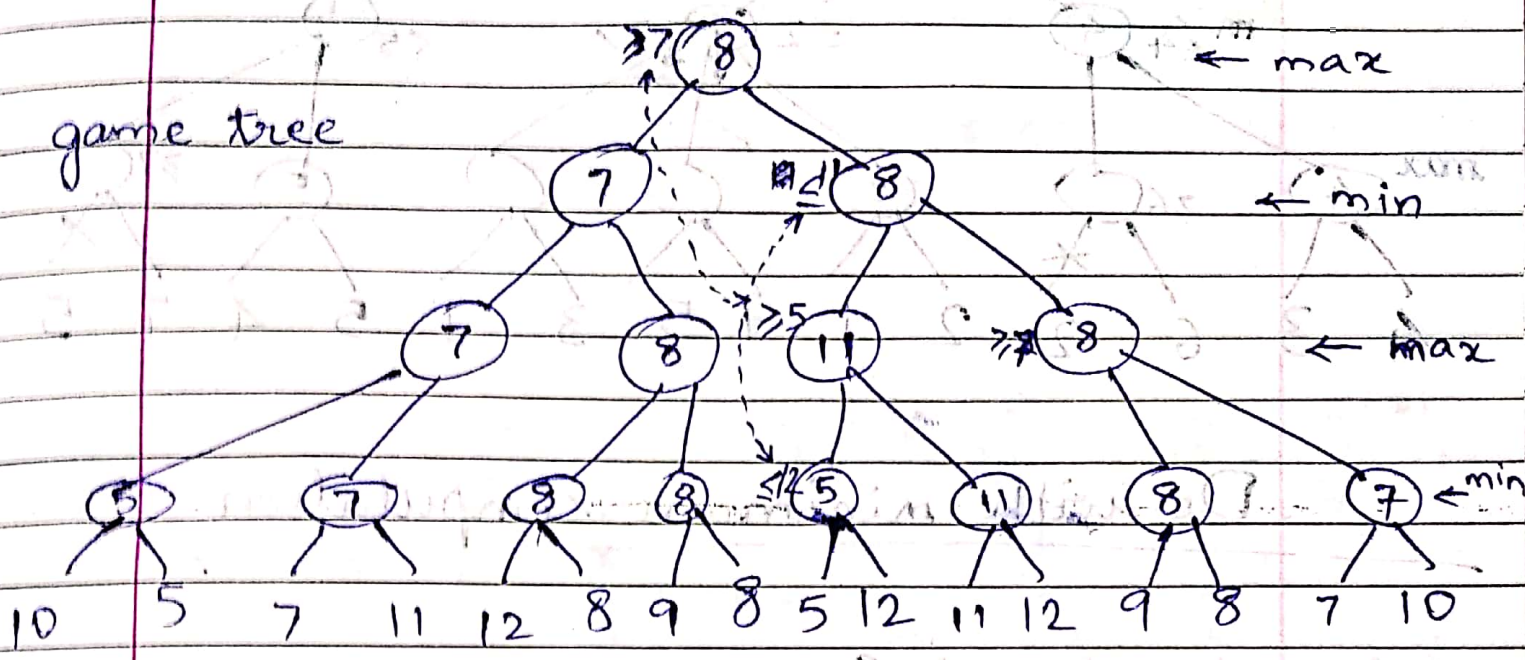
atleast 3 friend edges or atleast 3 stranger edges from a single vertex.

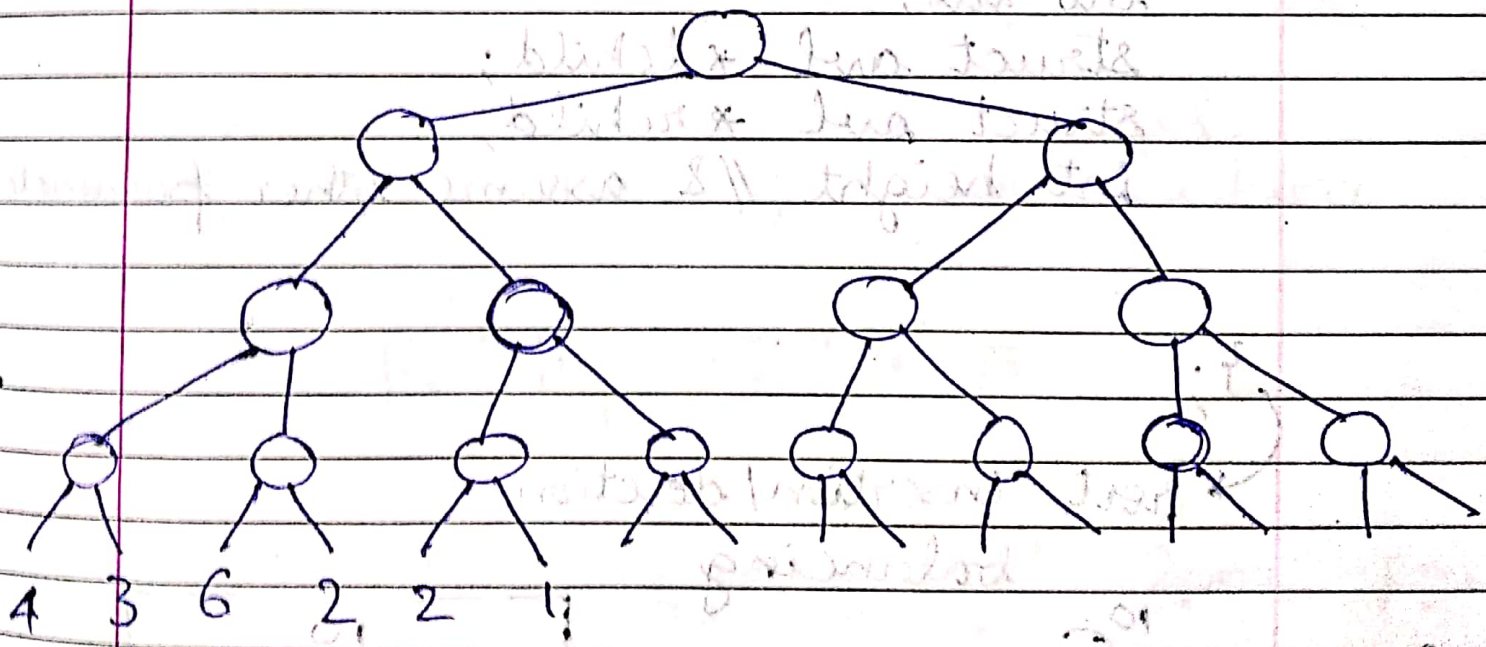
∴ a triangle is surely formed

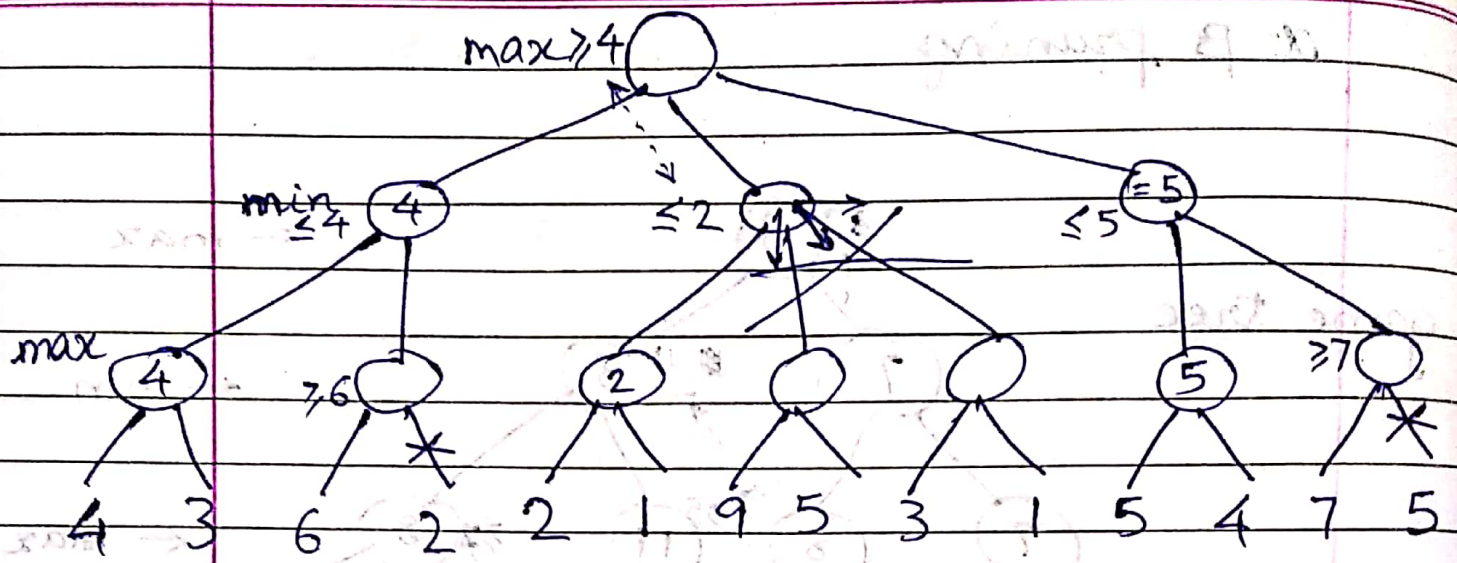
Page
Date 17/01/20

α-β pruning

game tree





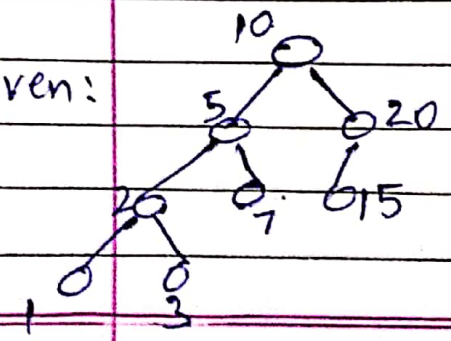


Do with minimum computation :

```
struct avl {
    int val;
    struct avl *lchild;
    struct avl *rchild;
    int height; // & assume other parameter
};
```

→ root insertion/deletion balancing

Given:



Print:

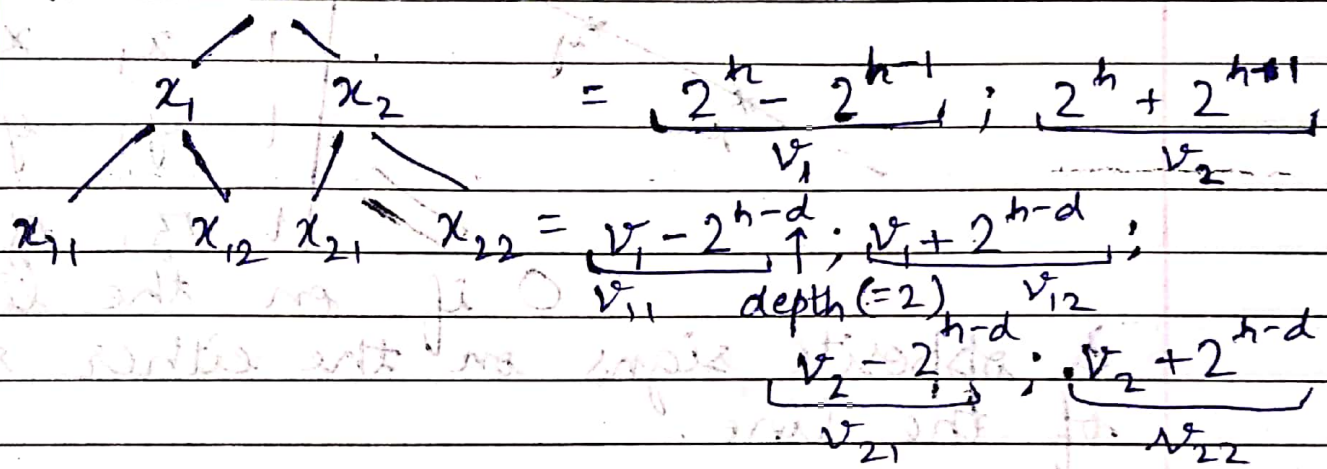
10
5 20
2 7 15
1 3

traversal can't be dfs, because we need to complete printing full line/ level before progressing further.

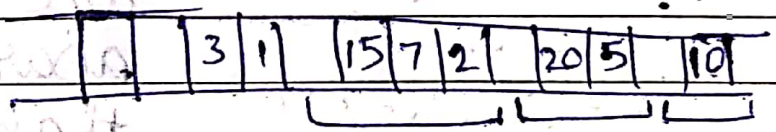
Let h = height of AVL tree.

CONCEPT: binary heaps

root x co-ord = 2^h



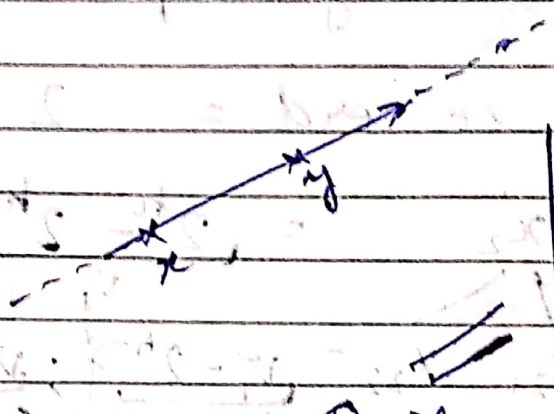
store in queue → level by level.
print in level after level fashion.



Next Week: Thursday – lab: 2pm – 6pm
23/01/20

Minimized trees.

Q Given three points x, y, z with their co-ordinates in the plane, determine whether z lies on, or to the left or right of the ray $\overrightarrow{xy}$.

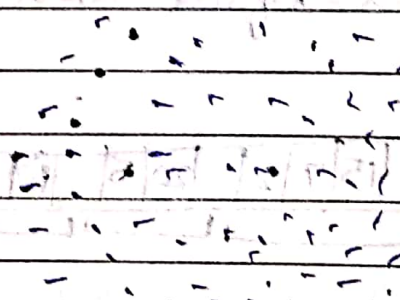


$$\begin{vmatrix} 1 & x_1 & x_2 \\ 1 & y_1 & y_2 \\ 1 & z_1 & z_2 \end{vmatrix}$$

0 if on the line

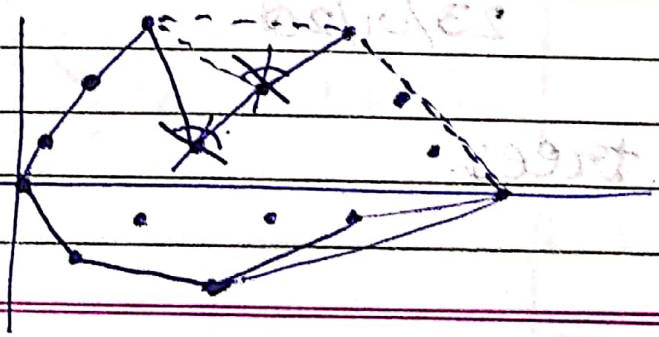
Δ opposite signs on the either side of the line.

Q. Find the pair of points which are farthest apart in less than $O(n^2)$ time.



Sol^n

Convex-hull



boundary found in $O(n \log n)$ time.

$O(n)$: by 2 pointer technique.

$O(n \log n)$: Pick every point one after other - do binary search on each point to find the farthest point.

$(n \times \log n)$

↑ each point — ↑ binary searching

designing a game tree.

Complexity : $O(n \log n + n \log n)$

$= O(n \log n)$ — convex hull / finding farthest points in convex hull

or $O(n \log n + n)$

$= O(n \log n)$

$\therefore O(n \log n)$

Page
Date 23/01/19

Laboratory

1. Primes upto n
2. Count the no. of primes upto n.
3. Implement above using sqrt method.
4. Implement above by storing primes in array : sieve of eosthosenes
5. Input 10 elements in array, print them.
6. Input a number n. Declare array of size n. int *arr = new int[n]. Randomly generate n numbers (rand()), sort them by swapping
7. Find the number of swaps required to sort the complete array.
8. Improve above algorithm to O(n log n) complexity : Merge Sort.
9. Write 2 player code for Connect4 Discussed in previous class.