

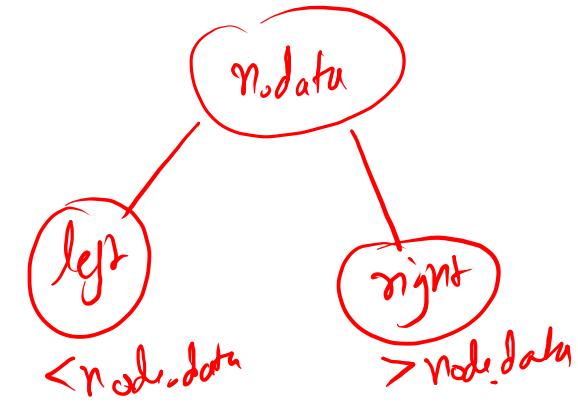
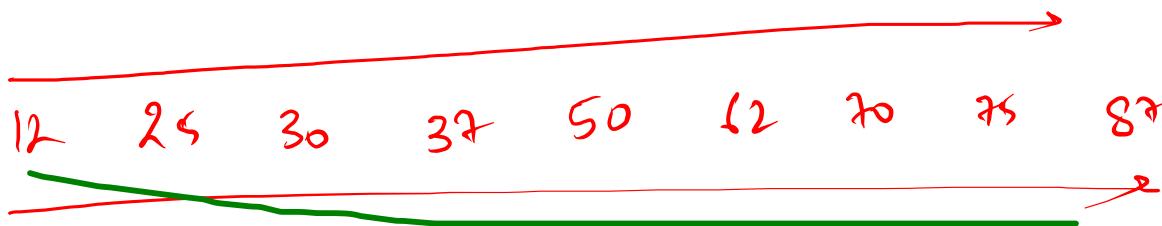
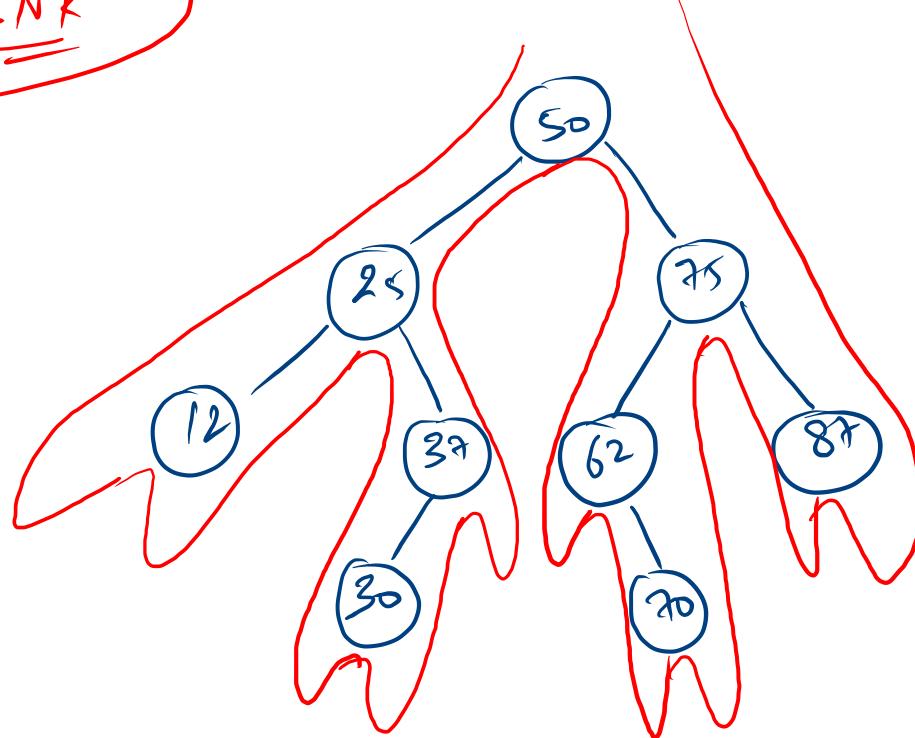
Binary Search Tree

Binary Tree + Data Discipline

BST Prop

$\text{left} < \text{node} < \text{right}$, $\forall \underline{\text{node}}$

Inorder \Rightarrow LNR



algo

→ Display (Structure)

→ Sum \Rightarrow (Structure)
Min \Rightarrow (Data)

Max \Rightarrow (Data)

Size (S)

find (D)

LCA ()

nodeToRoot path

Diameter

Binary Tree

n

n

n

n

n

n

n

n

n

n

Binary Search Tree

n

n

optimal

optimal

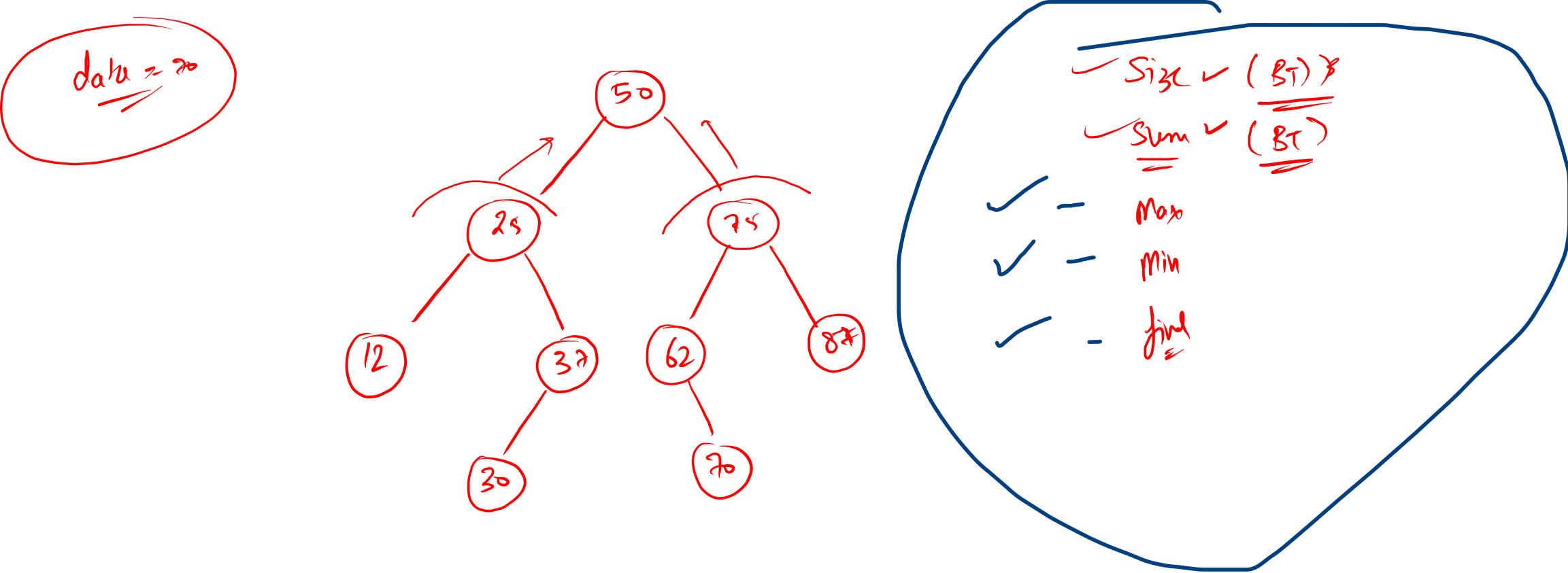
n

optimal

~

~

n



$\checkmark \text{Size } \checkmark (\text{BT})$

$\checkmark \text{Sum } \checkmark (\text{BT})$

$\checkmark \text{Max}$

$\checkmark \text{Min}$

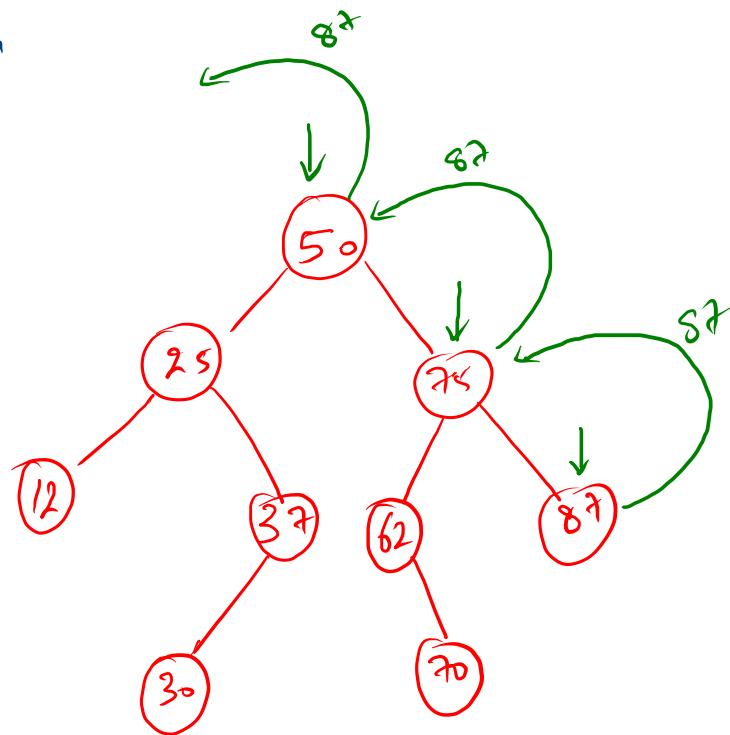
$\checkmark \text{find}$

19

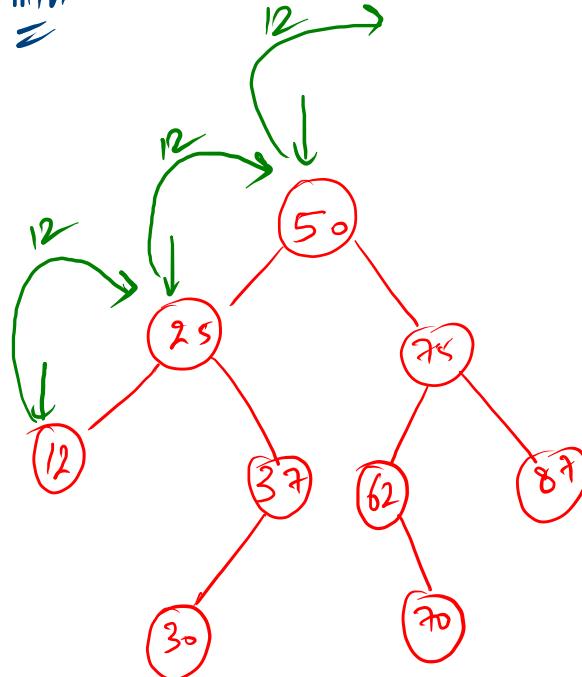
50 25 12 n n 37 30 n n n 75 62 n 70 n n 87 n n

70

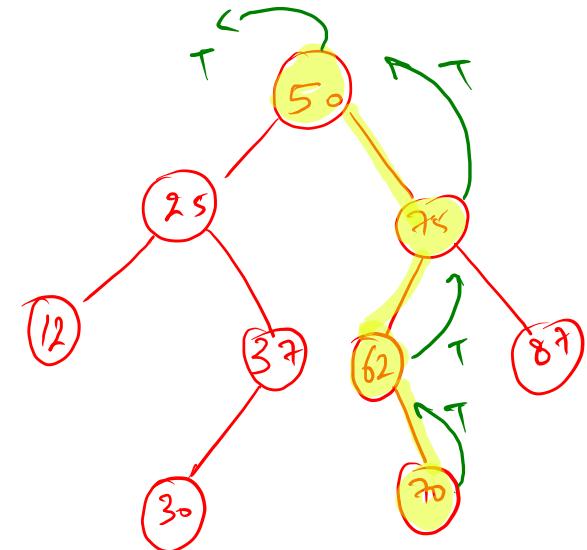
Max

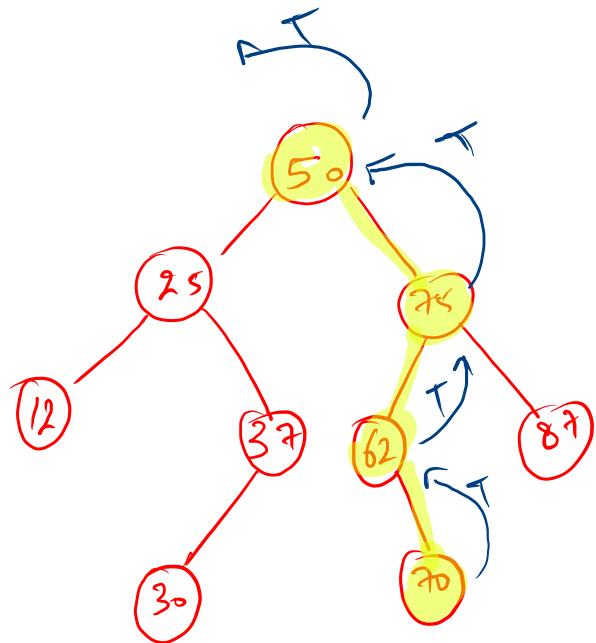


Min



Find, data = 70





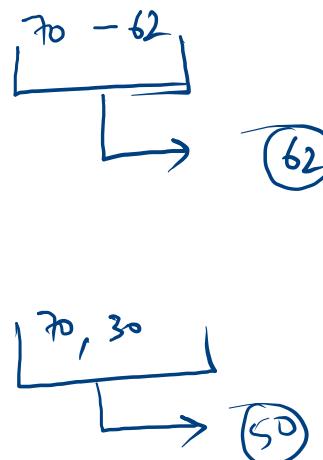
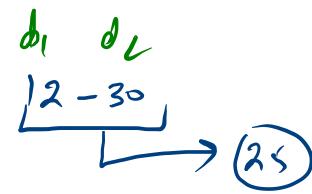
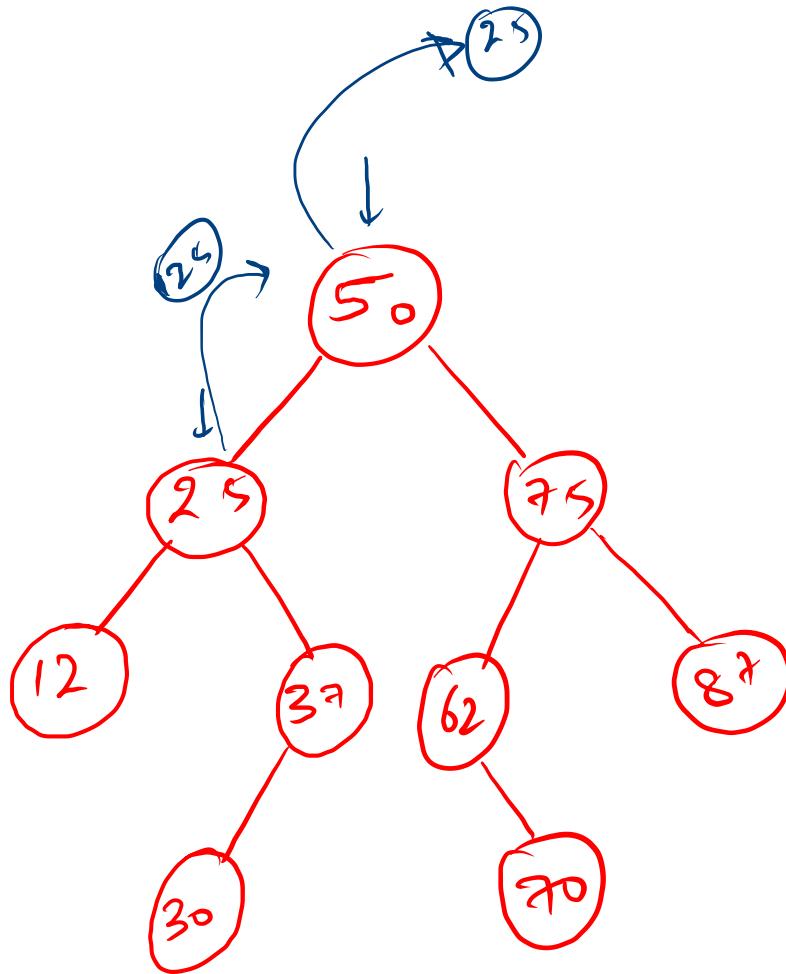
```

public static int max(Node node) {
    if(node.right != null){
        return max(node.right);
    }else{
        return node.data;
    }
}

public static int min(Node node) {
    if(node.left != null){
        return min(node.left);
    }else{
        return node.data;
    }
}

public static boolean find(Node node, int data){70
    if(node == null){
        return false;
    }
    if(data > node.data){
        return find(node.right,data); return find(node.right,data);
    }else if(data < node.data){
        return find(node.left,data); return find(node.left,data);
    }else{
        return true;
    }
}

```

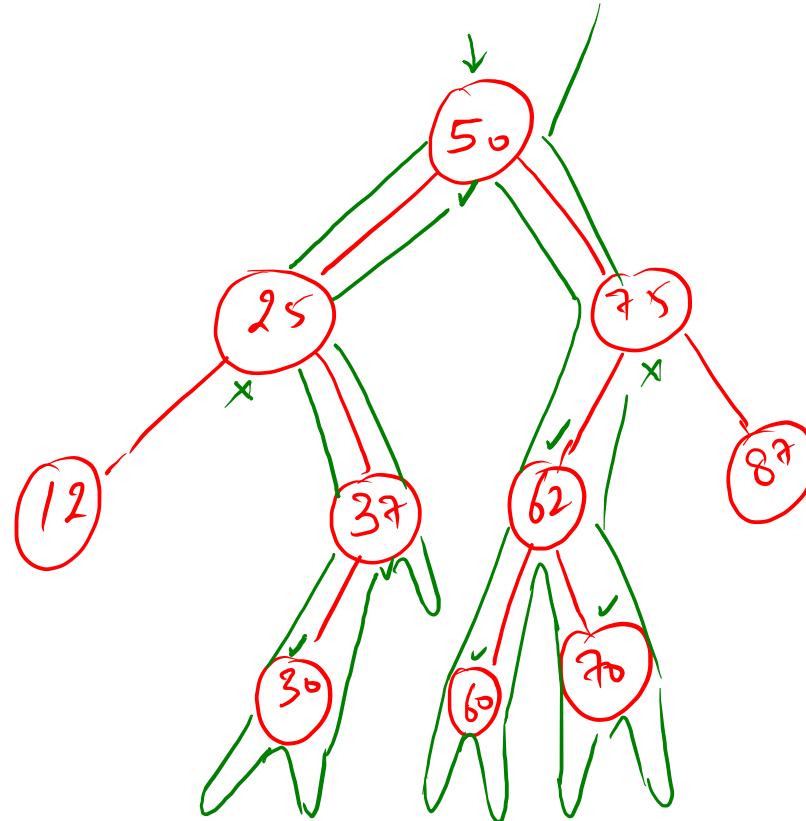


```

public static int lca(Node node, int d1, int d2) {
    if(d1 < node.data && d2 < node.data){
        return lca(node.left,d1,d2);
    }else if(d1 > node.data && d2 > node.data){
        return lca(node.right,d1,d2);
    }else{
        return node.data;
    }
}
  
```

Point in orange

$$\begin{cases} l_0 = 2^7 \\ h_i = 72 \end{cases}$$



30
37
60
62
70

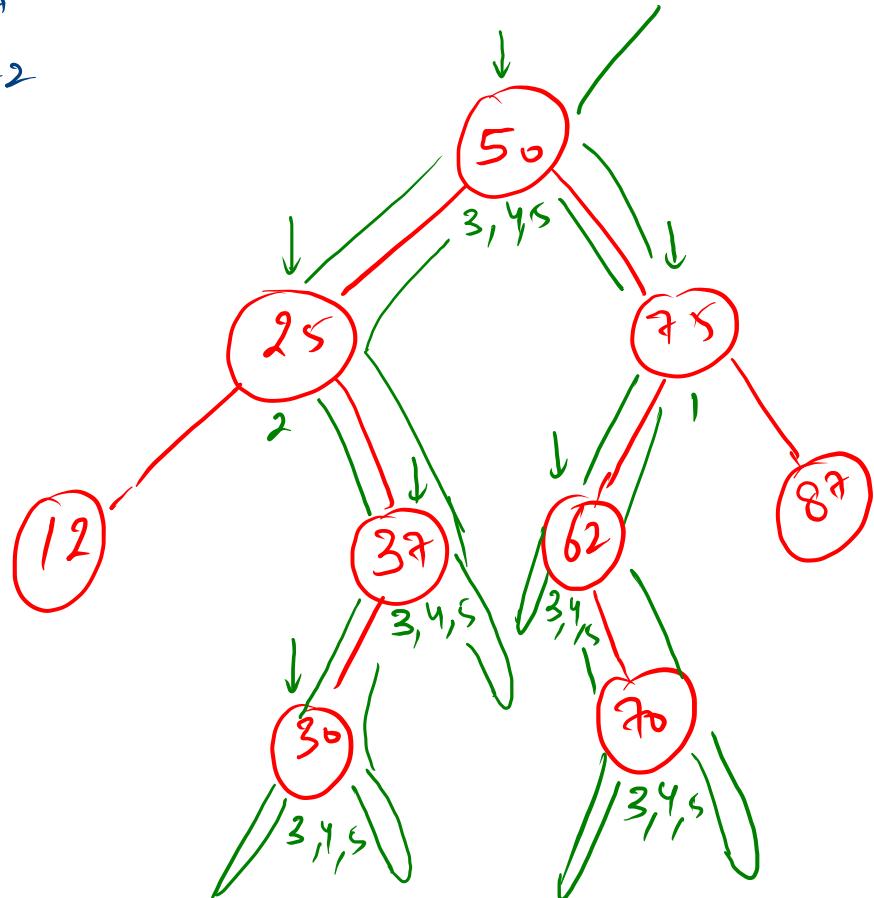
21

50 25 12 n n 37 30 n n n 75 62 60 n n 70 n n 87 n n

12

65

$$d_1 = 2^9$$
$$d_2 = 72$$



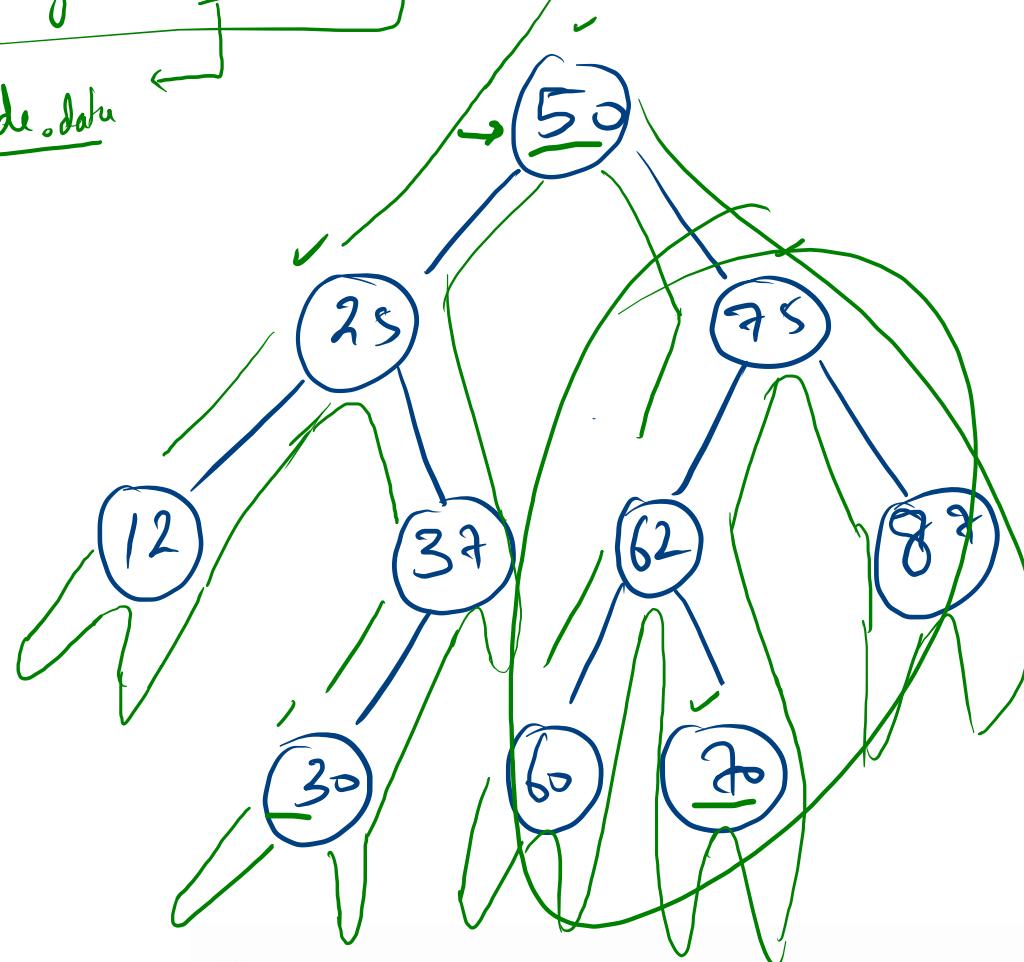
30
37
50
62
75

X

```
public static void pir(Node node, int d1, int d2) {
    if(node == null){
        return;
    }
    if(d1 < node.data && d2 < node.data){
        pir(node.left,d1,d2); -①
    }else if(d1 > node.data && d2 > node.data){
        pir(node.right,d1,d2); -②
    }else{
        pir(node.left,d1,d2); -③
        System.out.println(node.data); -④
        pir(node.right,d1,d2); -⑤
    }
}
```

Comp \Rightarrow Target - val

node.data



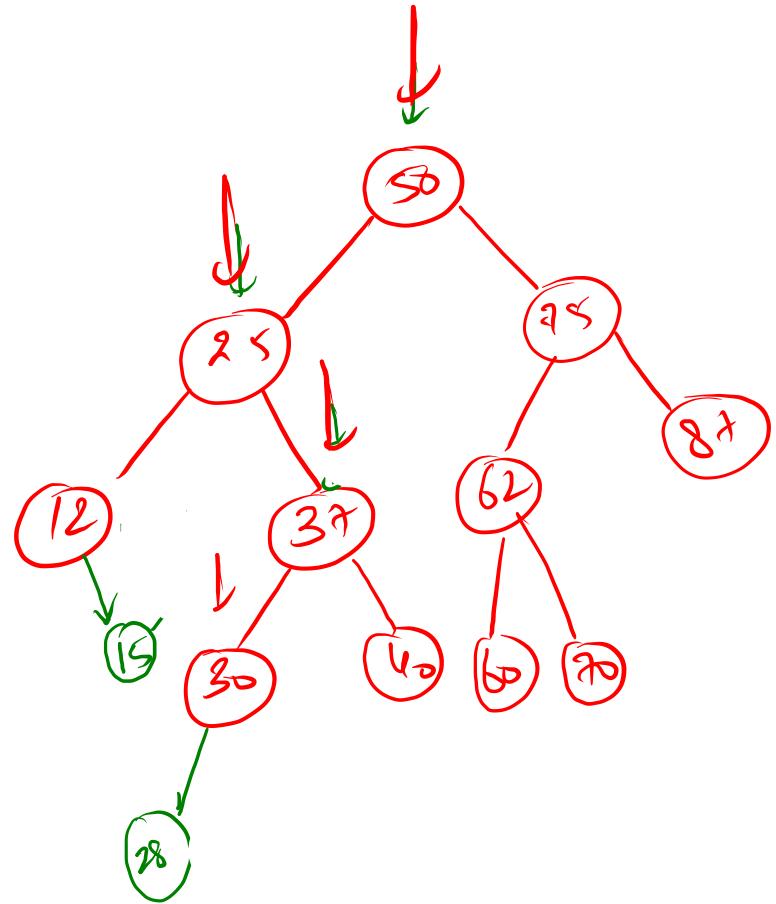
21

50 25 12 n n 37 30 n n n 75 62 60 n n 70 n n 87 n n
100

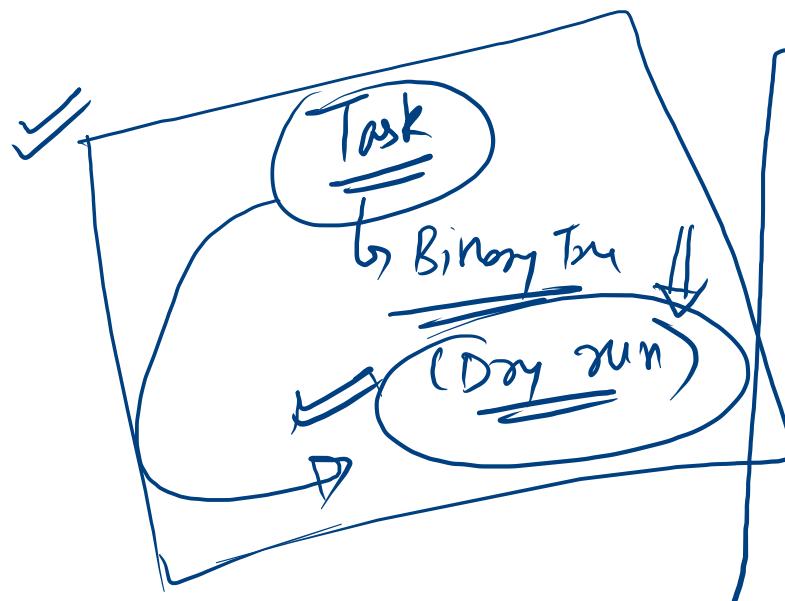
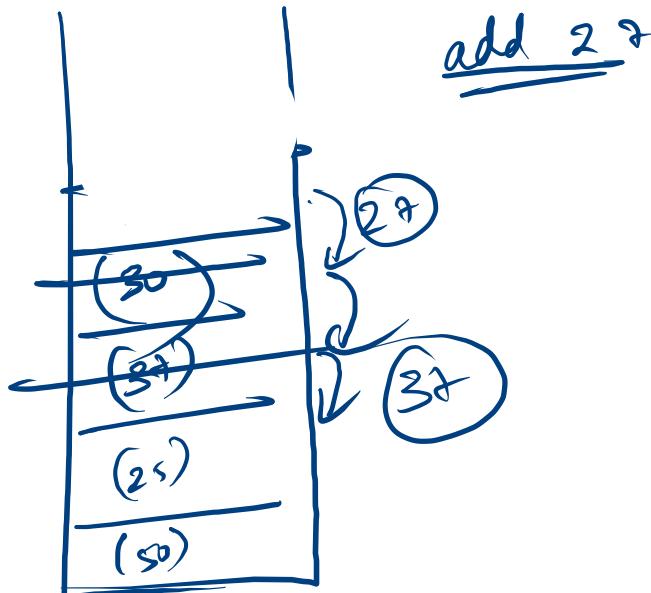
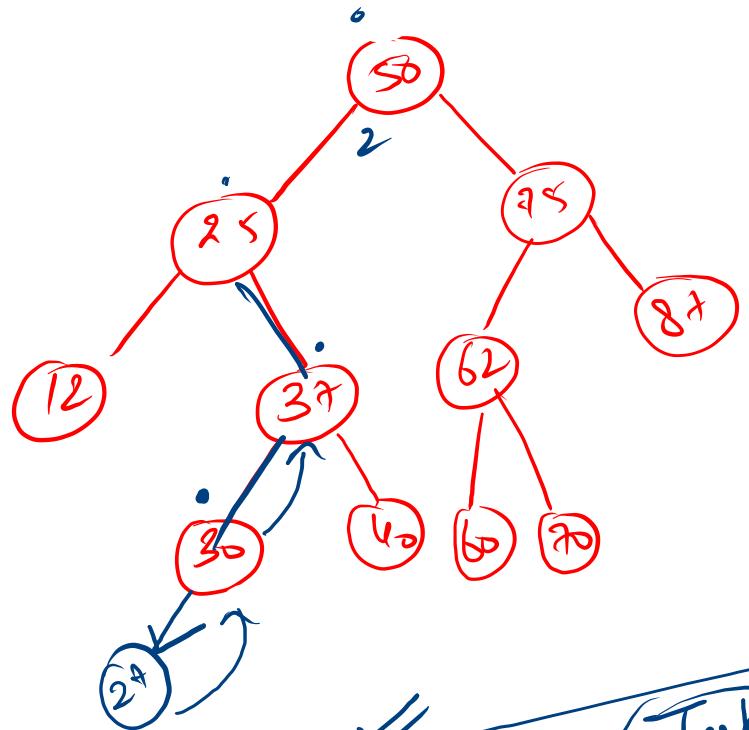
Target = 100 ✓

$$\begin{aligned}
 & 25 - 30 \\
 & 30 - 20 \\
 & 50 - 50 \\
 & 70 - (30) \\
 & 75 - 25 \\
 & \underline{\underline{val}} = \underline{\underline{comp}} =
 \end{aligned}$$

132
X (62 + 20)



add 15
= 28
= 37
add 28

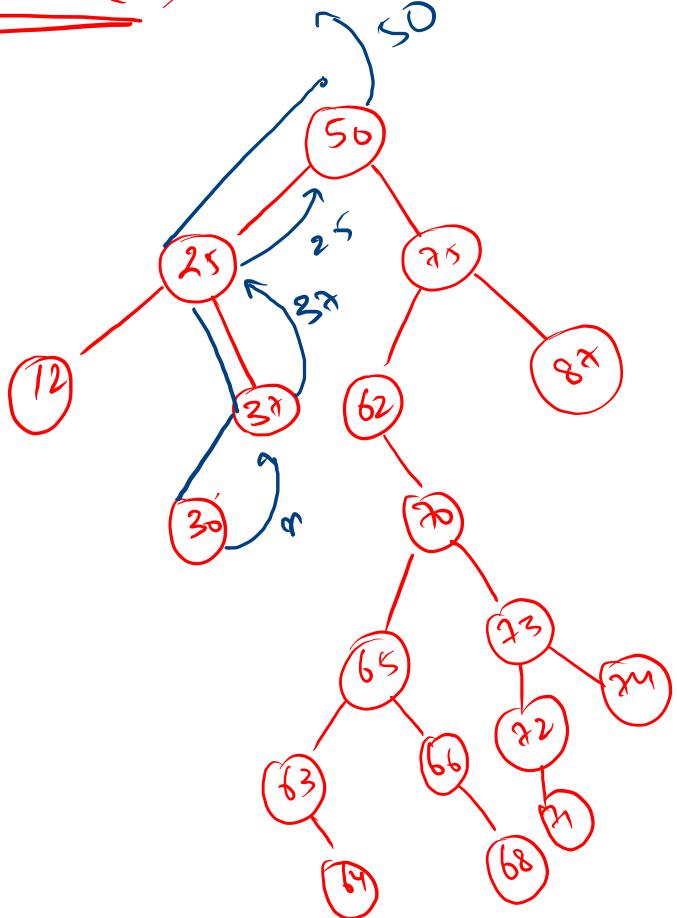


```

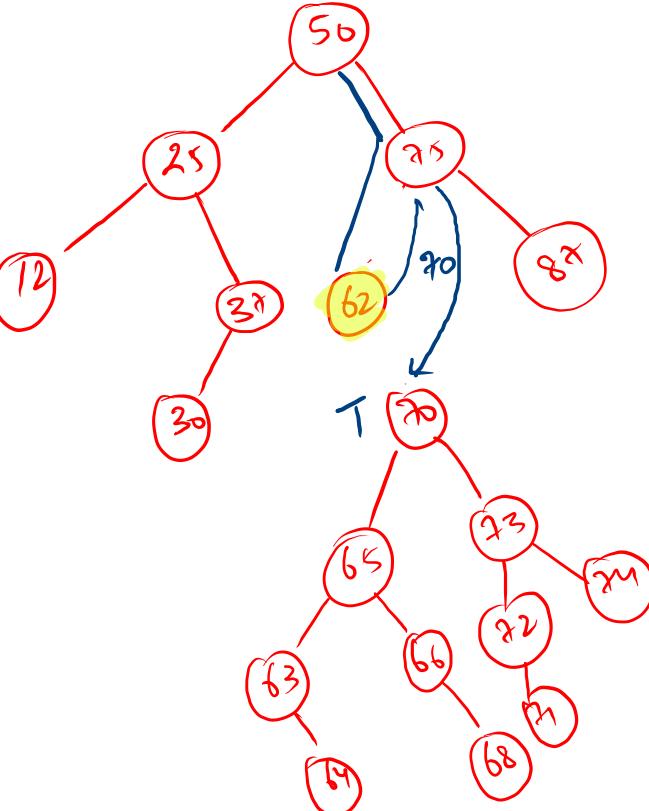
public static Node add(Node node, int data) {
    if(node == null){ ③
        return new Node(data,null,null); ①
    }
    if(data < node.data){ ①
        node.left = add(node.left,data); ②
    }else if(data > node.data){ ②
        node.right = add(node.right,data); ③
    }
    return node; ④
}

```

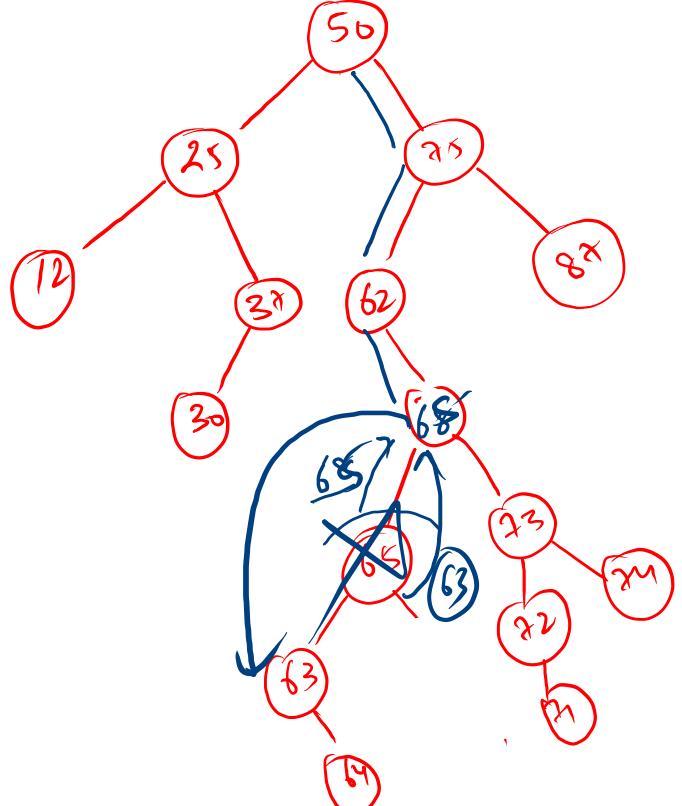
remove(30)



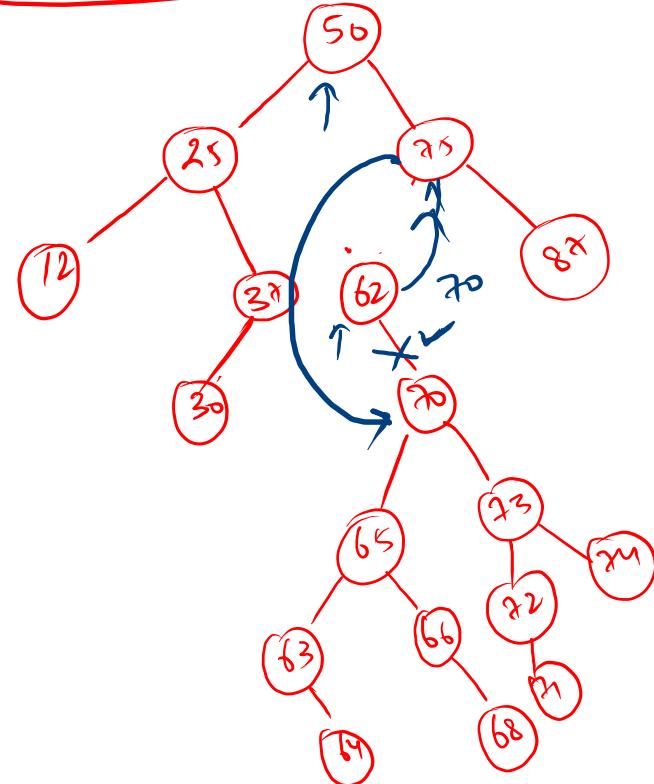
remove(62)



remove(70)



remove 6 2



```
if (node.left == null && node.right == null) {
    return null;
} else if (node.left == null) {
    return node.right;
} else if (node.right == null) {
    return node.left;
```

→ Keys are always unique



Key String	value Integer
India	100 200
UK	150
USA	180 185
China	160

① `HashMap<String, Integer> hm = new HashMap<>();`

HashMap Usage

functions

`hm.put(India, 100);`
`hm.put(UK, 150);`
`... (USA, 180);`
`... (China, 160);`
`hm.put(India, 200);`
`hm.put(USA, 185);`

put
insert → update

`hm.get(India) ⇒ 200`
`hm.get(PAK) ⇒ null`
`hm.containsKey(India) ⇒ True`
`hm.containsKey(PAK) = False`

import java.util.HashMap;

		<u>Key → Exists</u>	<u>Key → not exists</u>
Constant time	hm.put(Key, val) =	Update	insert.
Constant time	hm.get(Key) =	Returns val	Returns null
Constant time	hm.containsKey(Key) =	Returns true	Returns false
Constant time	hm.size() ⇒	[no. of elements in hm]	
Linear time <u>(n operations)</u>	hm.keySet() ⇒	[Returns all Keys in hm]	

↳ [Set □]


```

import java.util.HashMap;
public class hashMapIntro{
    Run | Debug
    public static void main(String[] args) {
        HashMap<String, Integer> hm = new HashMap<>();
        System.out.println(hm);

        hm.put("India", 200);
        hm.put("UK", 150);
        hm.put("USA", 180);
        hm.put("Dubai", 100);
        System.out.println(hm);
        hm.put("India", 250);
        System.out.println(hm);

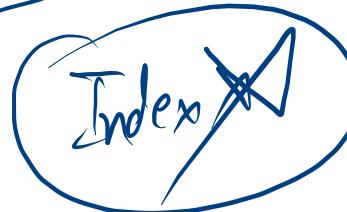
        System.out.println(hm.get("India"));
        System.out.println(hm.get("Pak"));

        System.out.println(hm.containsKey("India"));
        System.out.println(hm.containsKey("Pak"));

        System.out.println(hm.keySet());

        for(String key : hm.keySet()){
            System.out.println(key + "-->" + hm.get(key));
        }
    }
}

```



Aggressify@LAPTOP-I0ACH1PL MINGW64 /d/repository/pe
ged/3. HashMap (master)

```

$ javac hashMapIntro.java && java hashMapIntro
{}
{USA=180, UK=150, Dubai=100, India=200}
{USA=180, UK=150, Dubai=100, India=250}
250
null
true
false]
[USA, UK, Dubai, India] → Key Set
    USA-->180
    UK-->150
    Dubai-->100
    India-->250

```

Aggressify@LAPTOP-I0ACH1PL MINGW64 /d/repository/pe
ged/3. HashMap (master)

```

$ 

```


zmszeqxlzvheqwrofgcuntypejcxovtaqbnnqyqlmrwitz

Character	Integer
z	x2 3
m	x2
s	1
e	x2 3
q	x2 3 x5
x	x2
l	x2 3
v	x2
h	1
w	x2
y	x2
o	x2
f	1
g	1

c	x23	
u	1	
n	x2	
t	x23	
y	x2	
P	1	
j	1	
a	1	
b	1	
i	1	

inp = "abc aadce f"

Character	Integer
a	123
b	1
c	12
d	1
e	1
f	1

$$\text{max freq} = 3$$

$$\text{max freq char} \leftarrow a$$

```

HashMap<Character, Integer> hm = new HashMap<>();

for(int idx = 0 ; idx < inp.length() ; idx++){
    char ch = inp.charAt(idx);

    if(hm.containsKey(ch)){
        // update
        hm.put(ch, hm.get(ch)+1);
    }else{
        // insert
        hm.put(ch, 1);
    }
}

int maxFreq = 0;
char maxFreqChar = ' ';

for(char ch : hm.keySet()){
    int freq = hm.get(ch);

    if(freq > maxFreq){
        maxFreq = freq;
        maxFreqChar = ch;
    }
}

System.out.println(maxFreqChar);

```

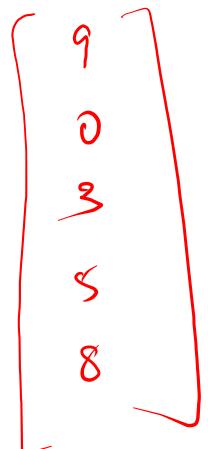
arr1

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

arr2

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
9	7	1	0	3	6	5	9	1	1	8	0	2	4	2	7	1	5

Integer	Boolean
5	true
9	true
8	true
0	true
3	true



Integer	Integer
1	x_2x_0
2	$\cancel{x} \cancel{2} \cancel{3} \cancel{2} 1$
3	1
5	x_0

