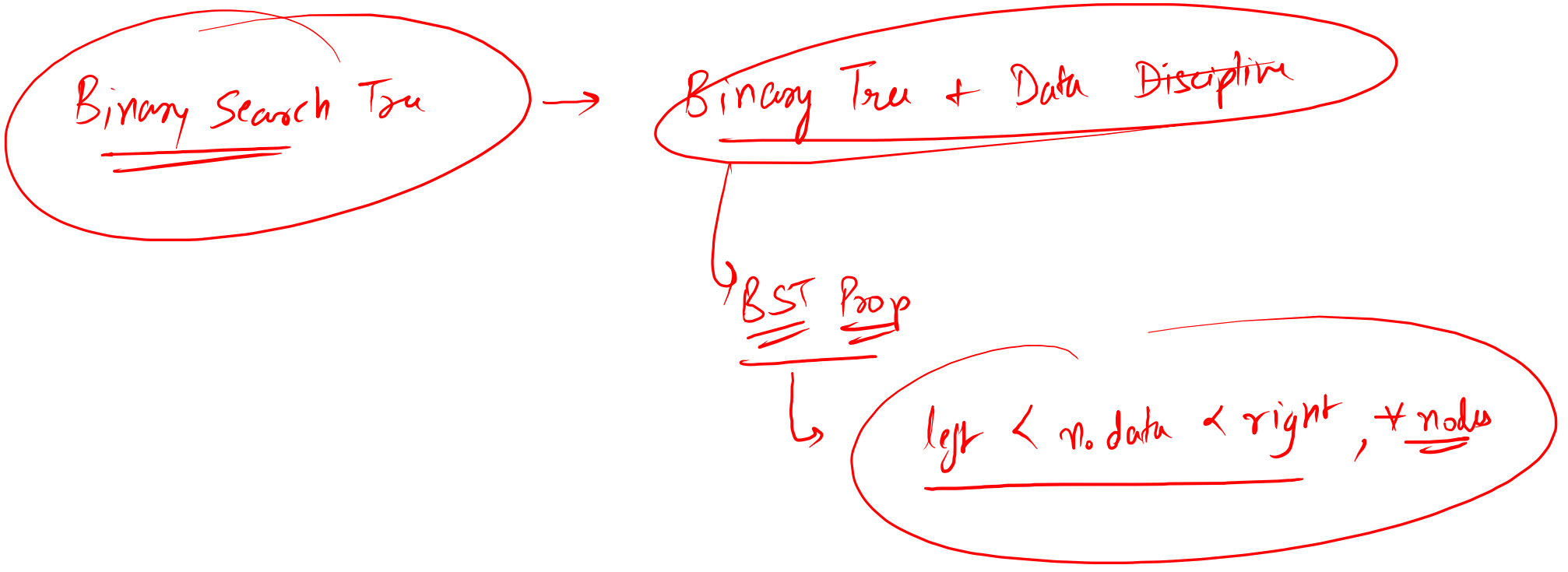
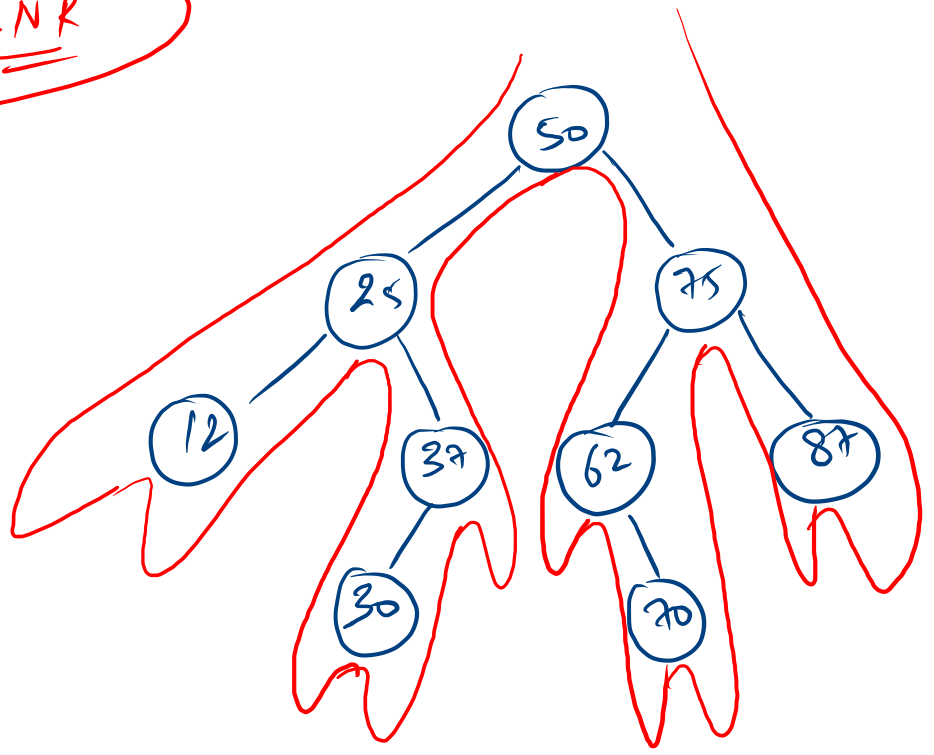
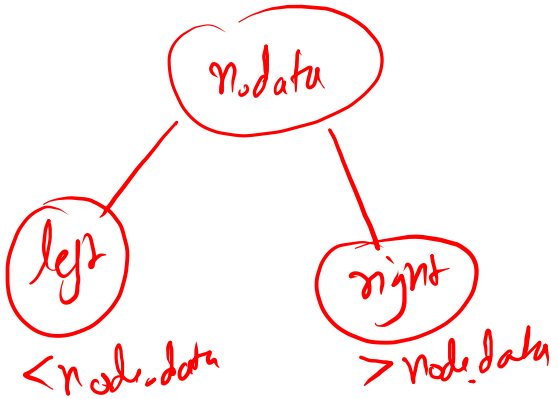
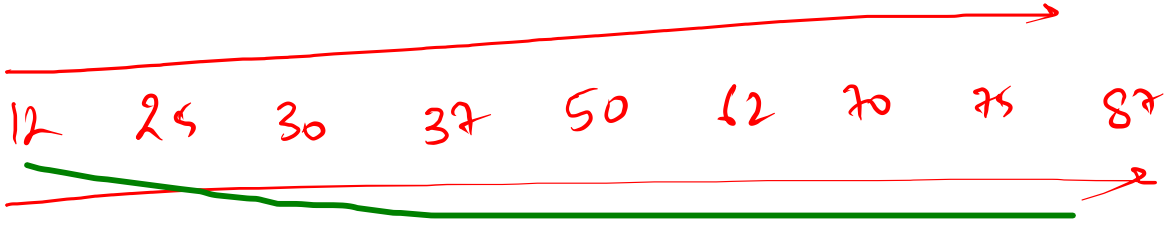

Binary Search Tree
Binary Tree + Data Discipline
BST Prop
left < n.data < right, ∀ nodes

Inorder → L N R


50
25
75
12
37
62
87
30
70

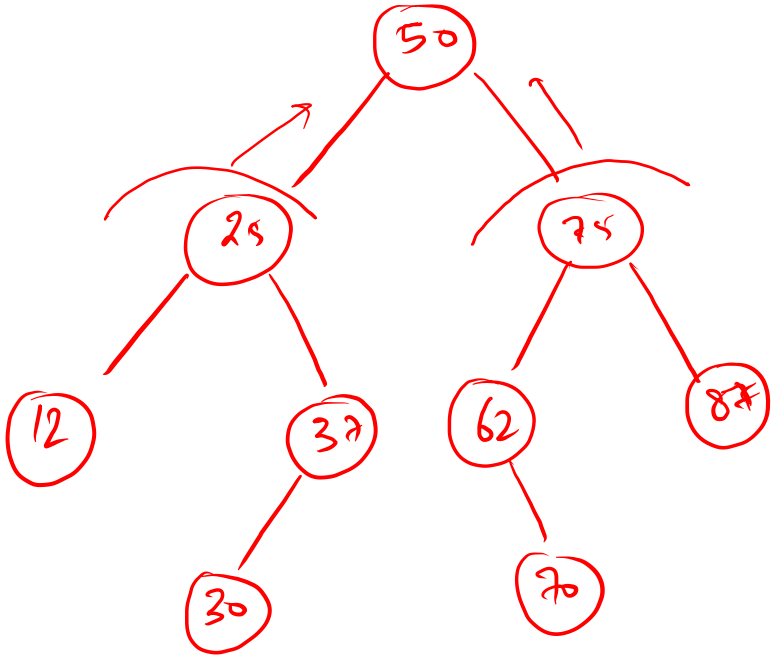

n.data
left
right
<node.data
>node.data


12
25
30
37
50
62
70
75
87

algo

	Binary Tree	Binary Search Tree
→ Display (Structure)	n	n
→ Sum → (Structure)	n	n
Min → (Data)	n	optimal
Max → (Data)	n	optimal
Size (S)	n	n
find (D)	n	optimal
LCA ( )	n	=
node To root path	n	=
Diameter	n	n

data = x



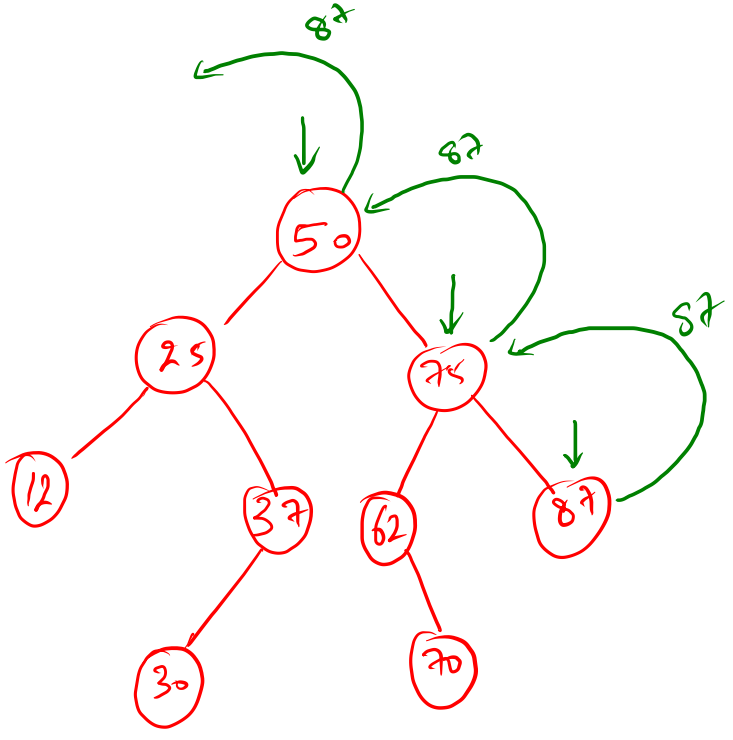
- Size ✓ (BT) ✗
- Sum ✓ (BT)
- ✓ - Max
- ✓ - Min
- ✓ - find

19

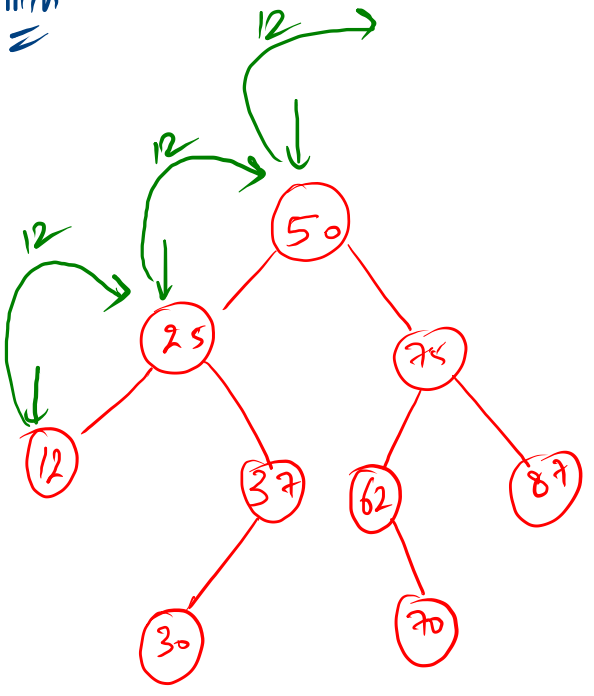
50 25 12 n n 37 30 n n n 75 62 n 70 n n 87 n n

70

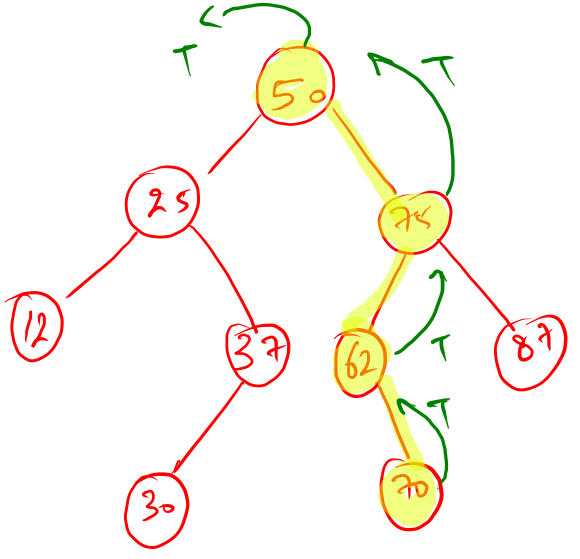
Max

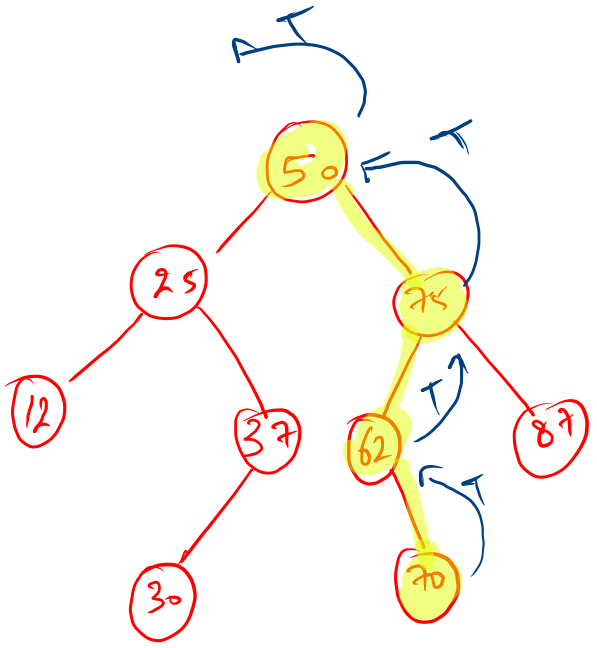


Min

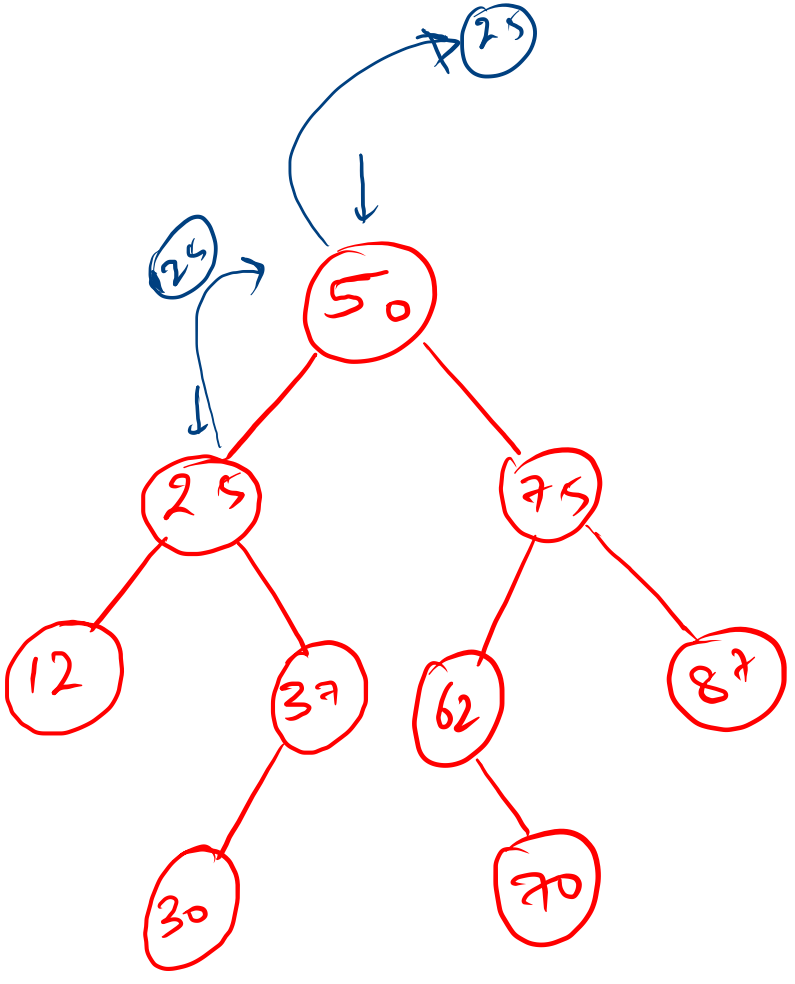


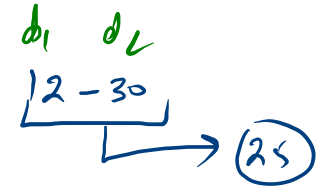
Find, data = 70

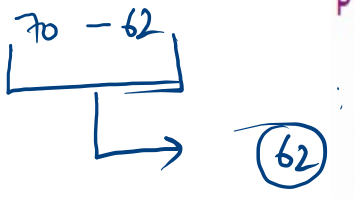


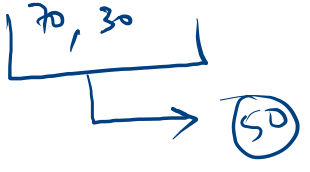



```java
public static int max(Node node) {
  if(node.right != null){
      return max(node.right);
  }else{
      return node.data;
  }
}

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){
  if(node == null){
      return false;
  }
  if(data > node.data){
      return find(node.right,data);
  }else if(data < node.data){
      return find(node.left,data);
  }else{
      return true;
  }
}
```





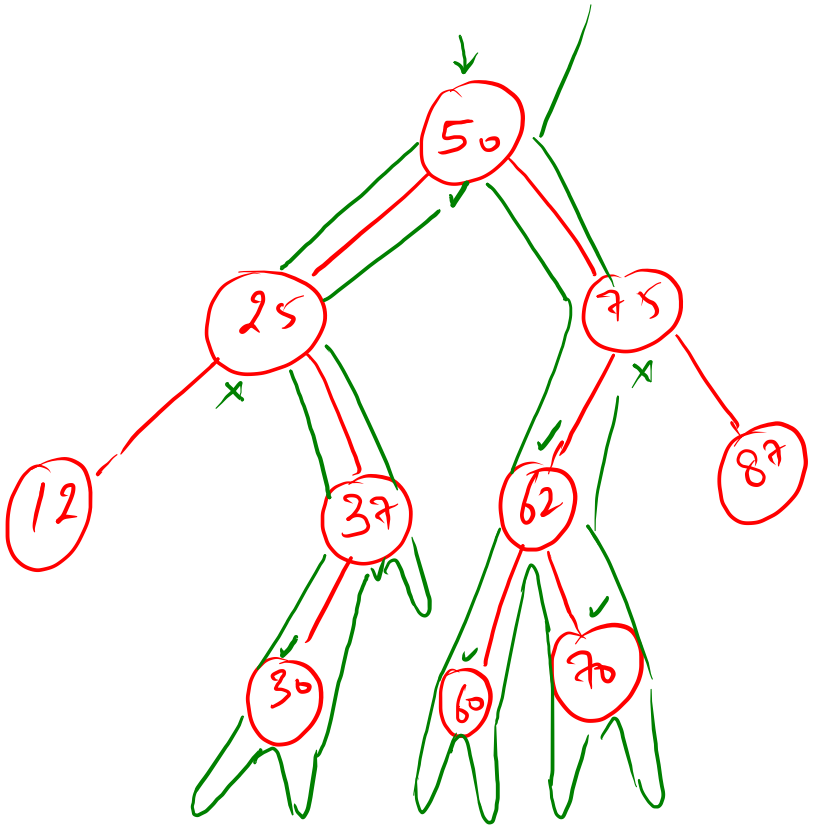





```java
public static int lca(Node node, int d1, int d2) {
  if(d1 < node.data && d2 < node.data){
    return lca(node.left,d1,d2);
  }else if(d1 > node.data && d2 > node.data){
     return lca(node.right,d1,d2);
  }else{
     return node.data;
  }
}
```

Print in range

$[l_0 = 27, h_1 = 72]$

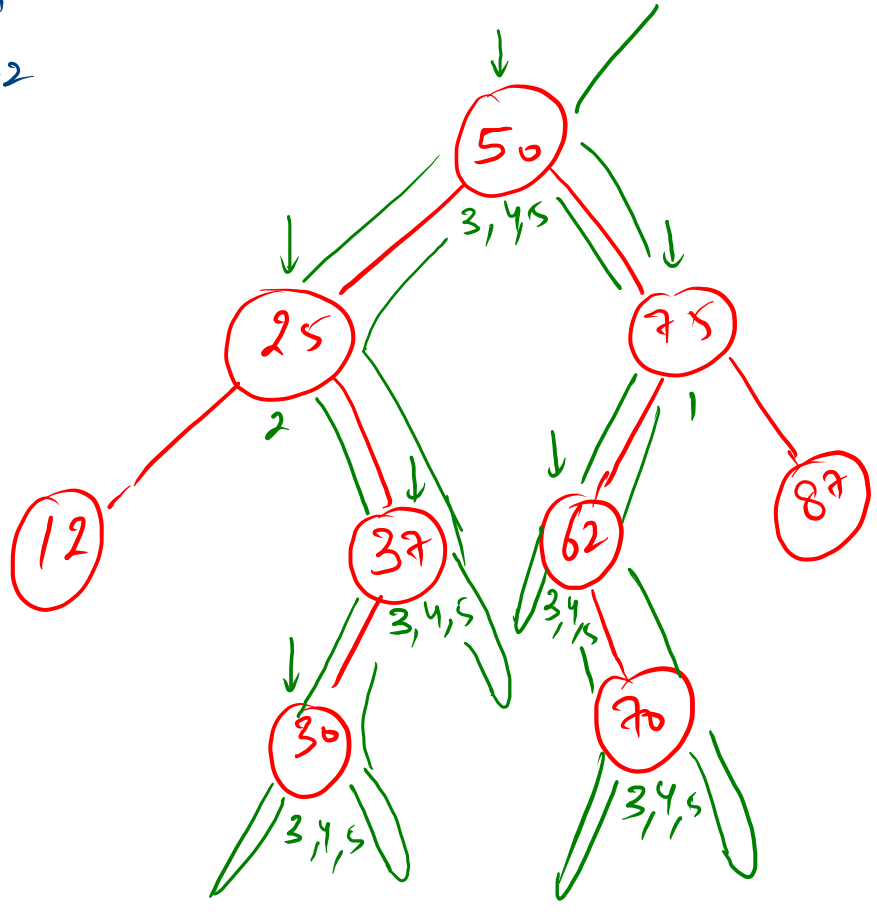


30
37
50
62
70

21
50 25 12 n n 37 30 n n n 75 62 60 n n 70 n n 87 n n
12
65

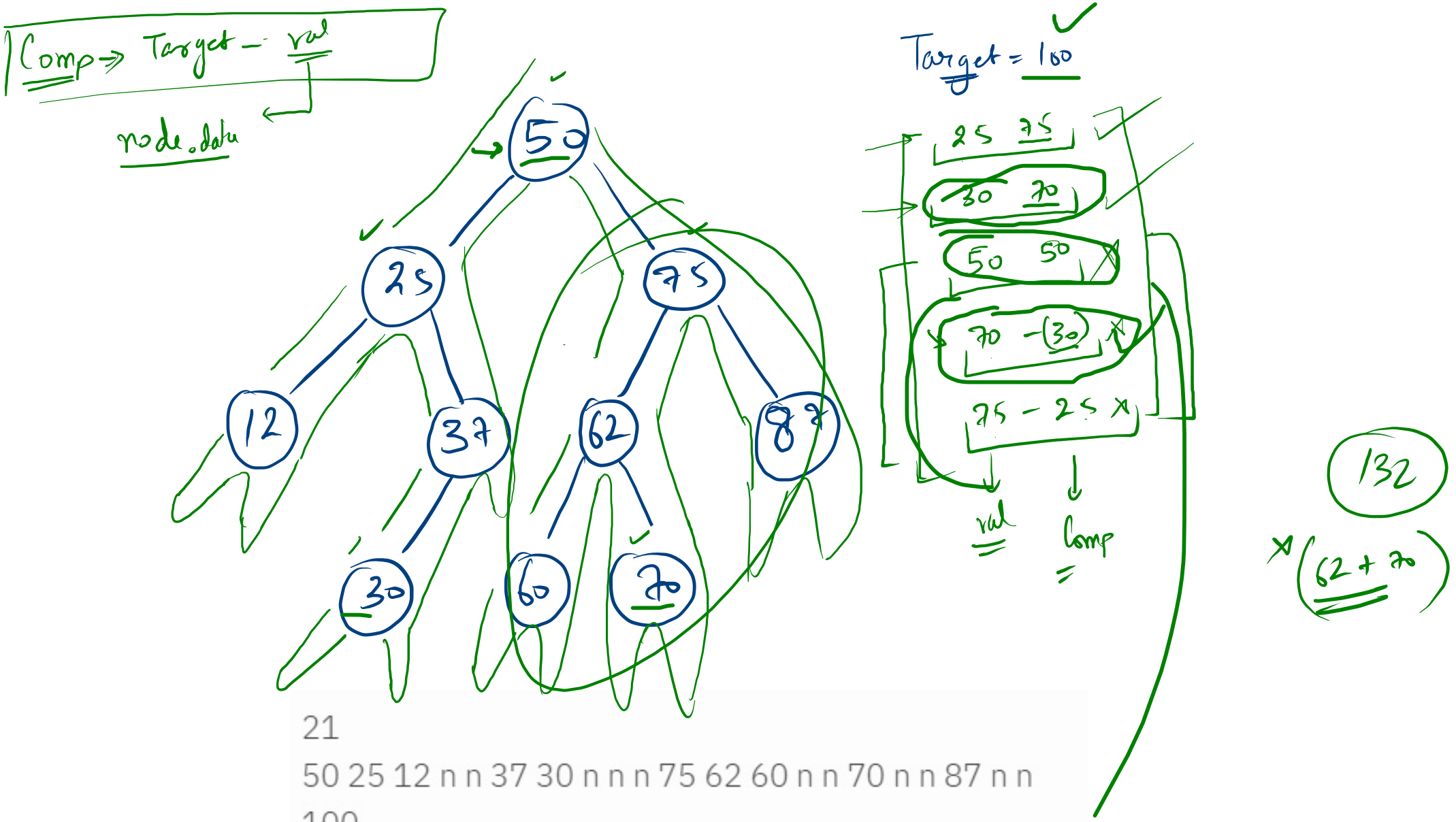
$d_1 = 27$

$d_2 = 72$



30
37
50
62
70

```java
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
    }
}
```



21

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

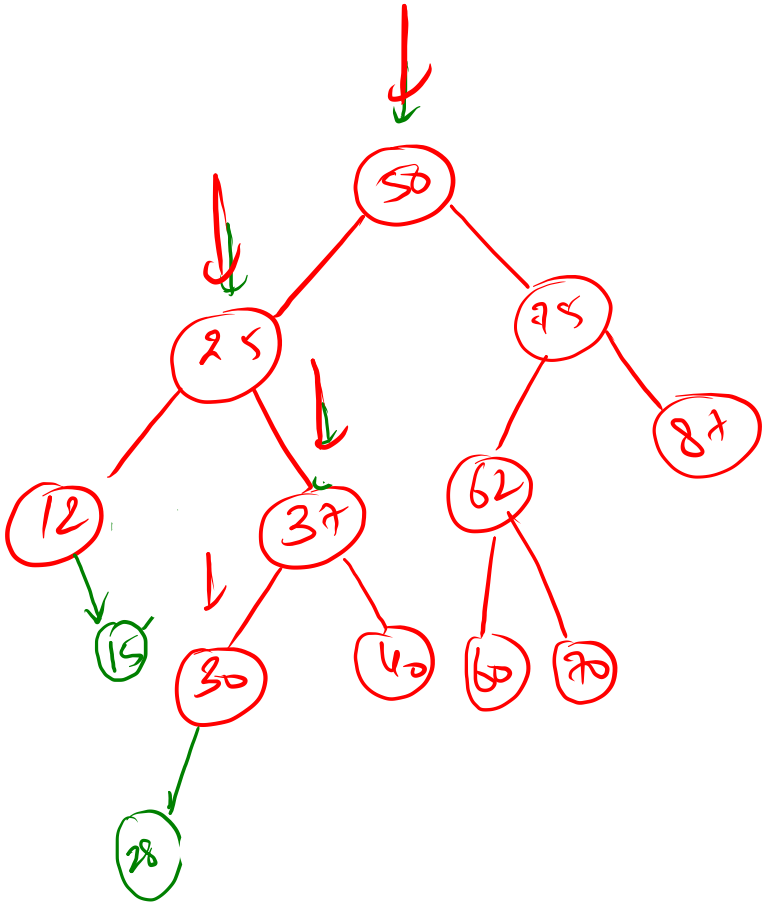
100

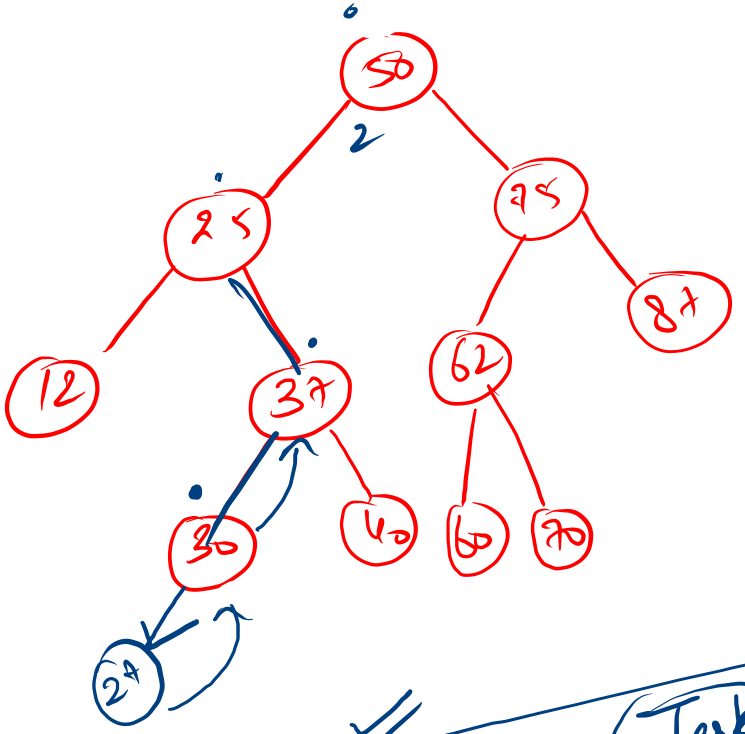
add 15

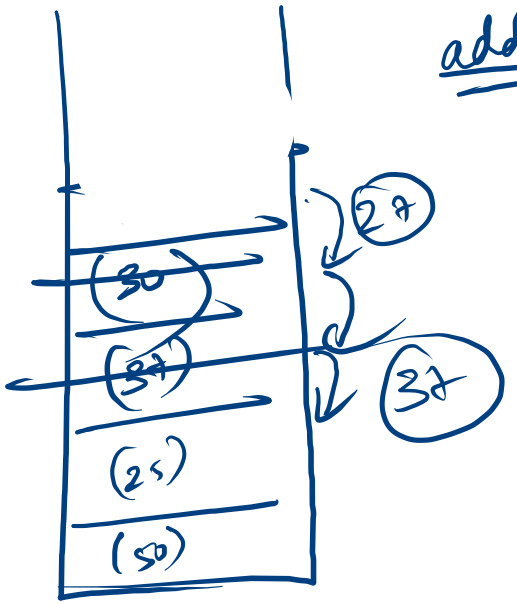
= 28

= 37

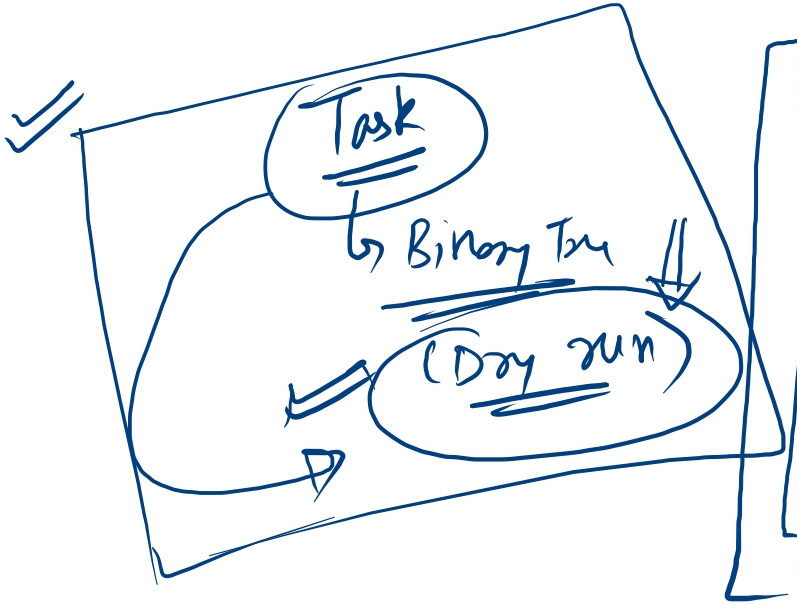
add 27







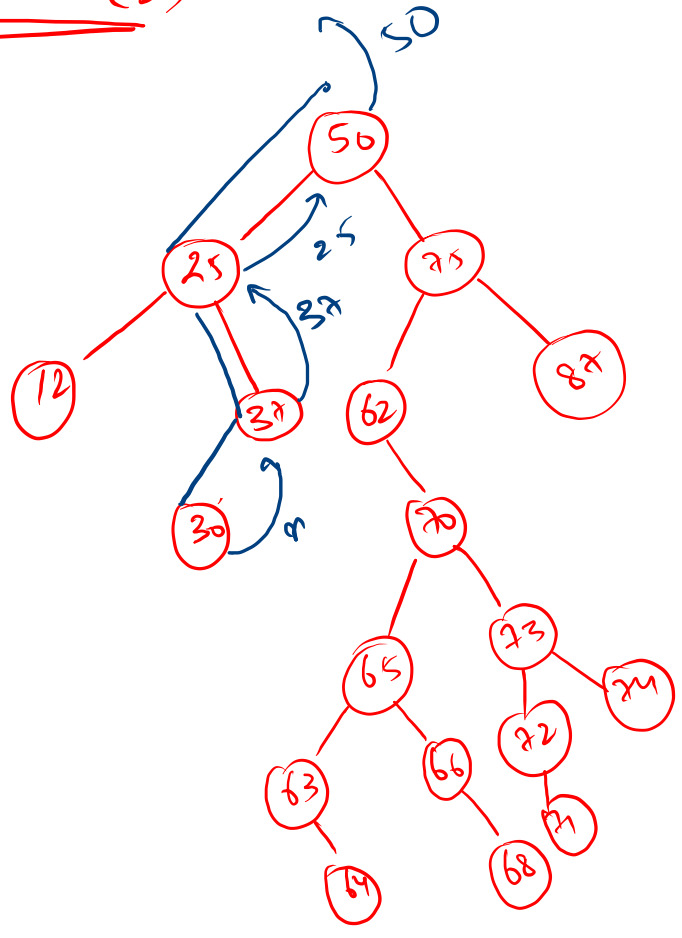
add 27



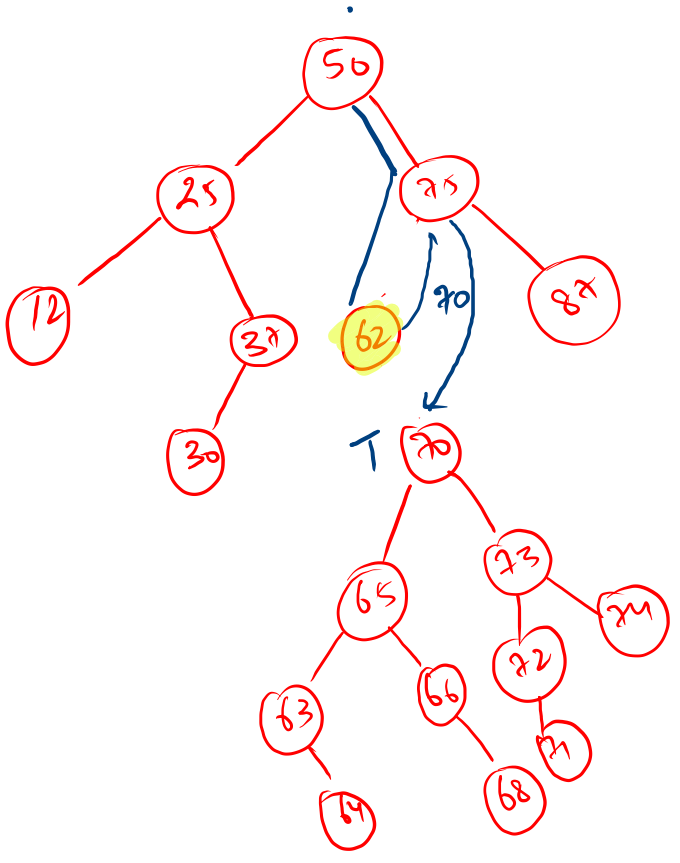

```java
public static Node add(Node node, int data) {
    if(node == null){
        return new Node(data,null,null);
    }
    if(data < node.data){
        node.left = add(node.left,data);
    }else if(data > node.data){
        node.right = add(node.right,data);
    }

    return node;
}
```

C3 1 C1 2 C2 3 4

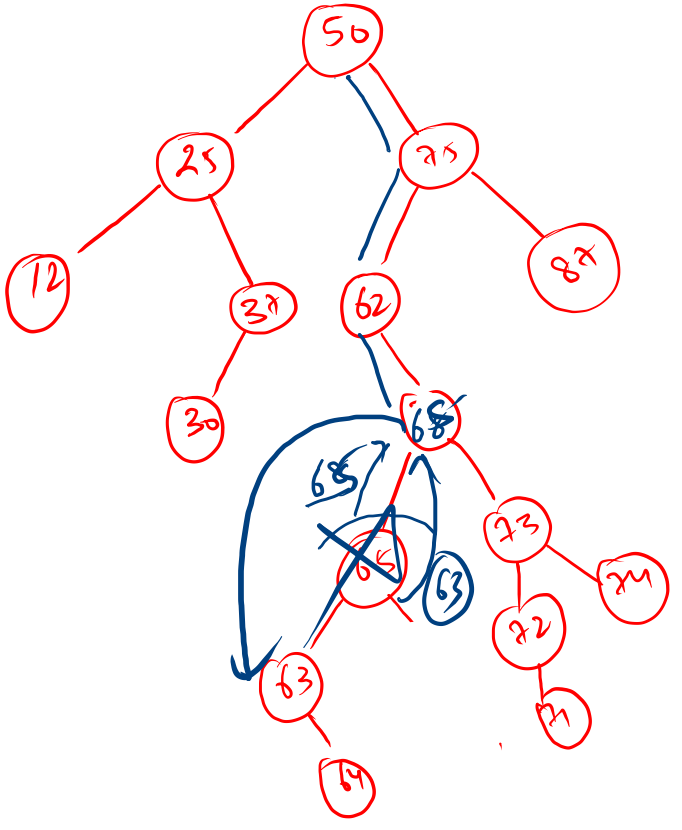
remove (30)



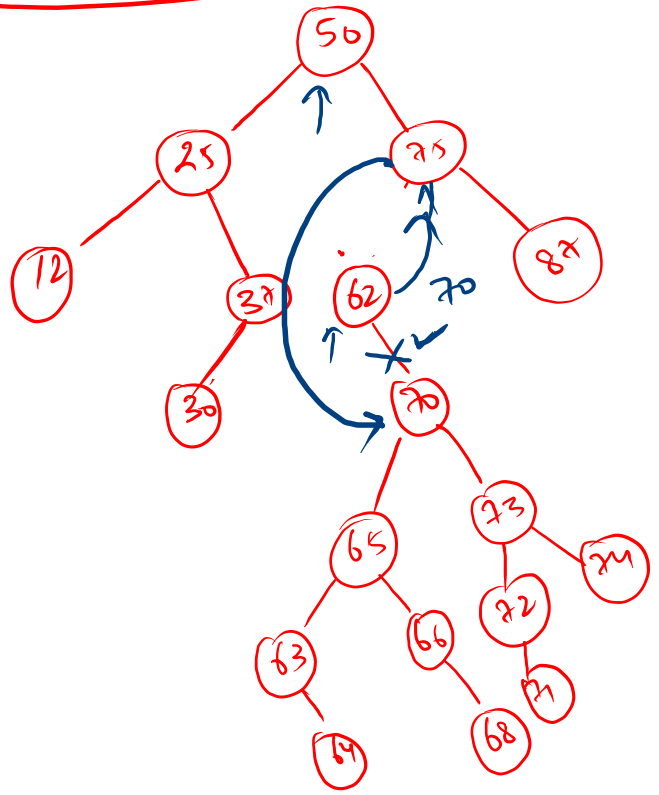
remove (62)



remove (70)



Scenario 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

HashMap → Usage

HashMap Usage

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

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

functions

```
hm.put(India, 100);
hm.put(UK, 150);
hm.put(USA, 180);
hm.put(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;

		Key → Exists	Key → not exists
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 (n operations)	hm.keySet() ⇒	[returns all Keys in hm]	

↳ Set<>

```java
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));
        }
    }
}
```

Index ✗

```
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]
USA-->180
UK-->150
Dubai-->100
India-->250

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

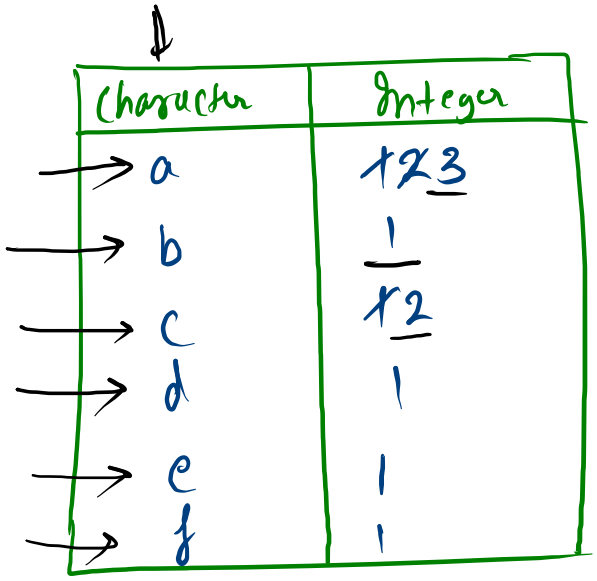
KeySet

zmszeqxllzvheqwrofgcuntypejcxovtaqbnqyqlmrwitc

Character	Integer
z	1 2 3
m	1 2
s	1
e	1 2 3
q	1 2 3 4 5
x	1 2
l	1 2 3
v	1 2
h	1
w	1 2
r	1 2
o	1 2
f	1
g	1

c	1 2 3
u	1
n	1 2
t	1 2 3
y	1 2
p	1
j	1
a	1
b	1
i	1

inp = " a b c a a d c e f "

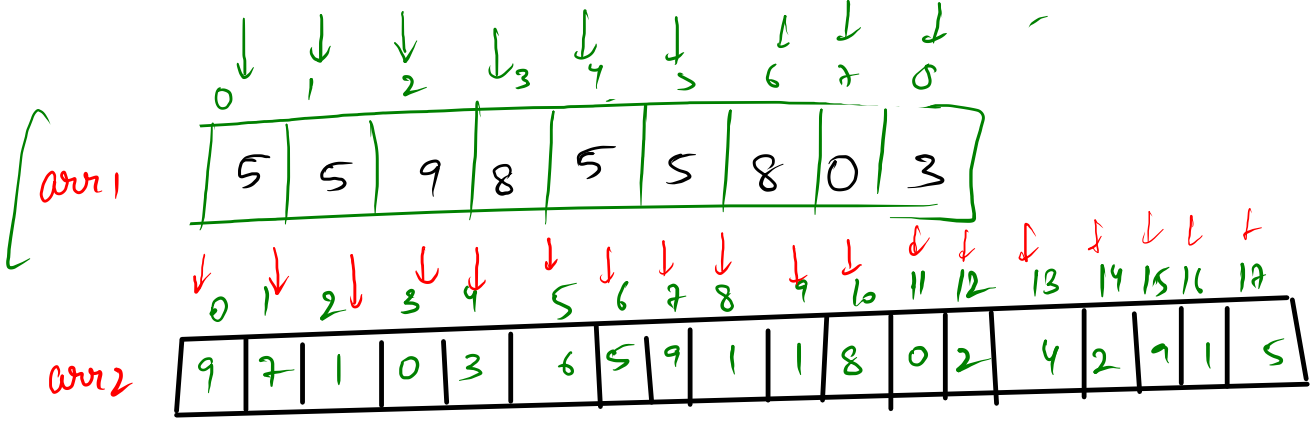


maxFreq = 3

maxFreqChar = a

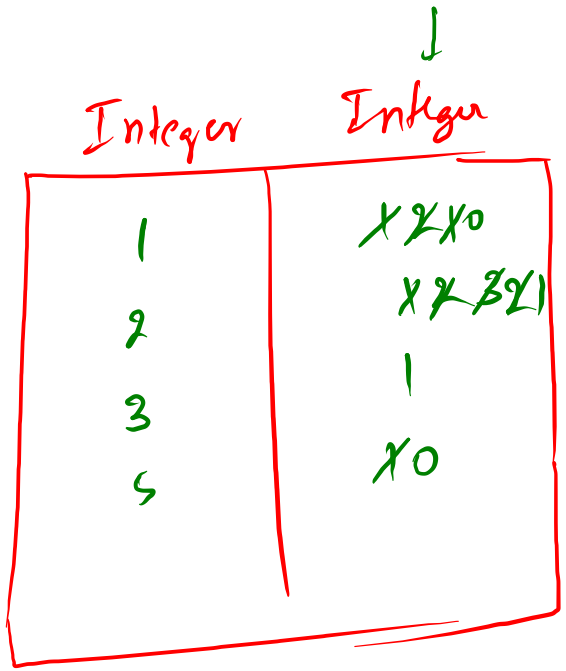
```java
HashMap<Character,Integer> hm = new HashMap<>();

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);
```

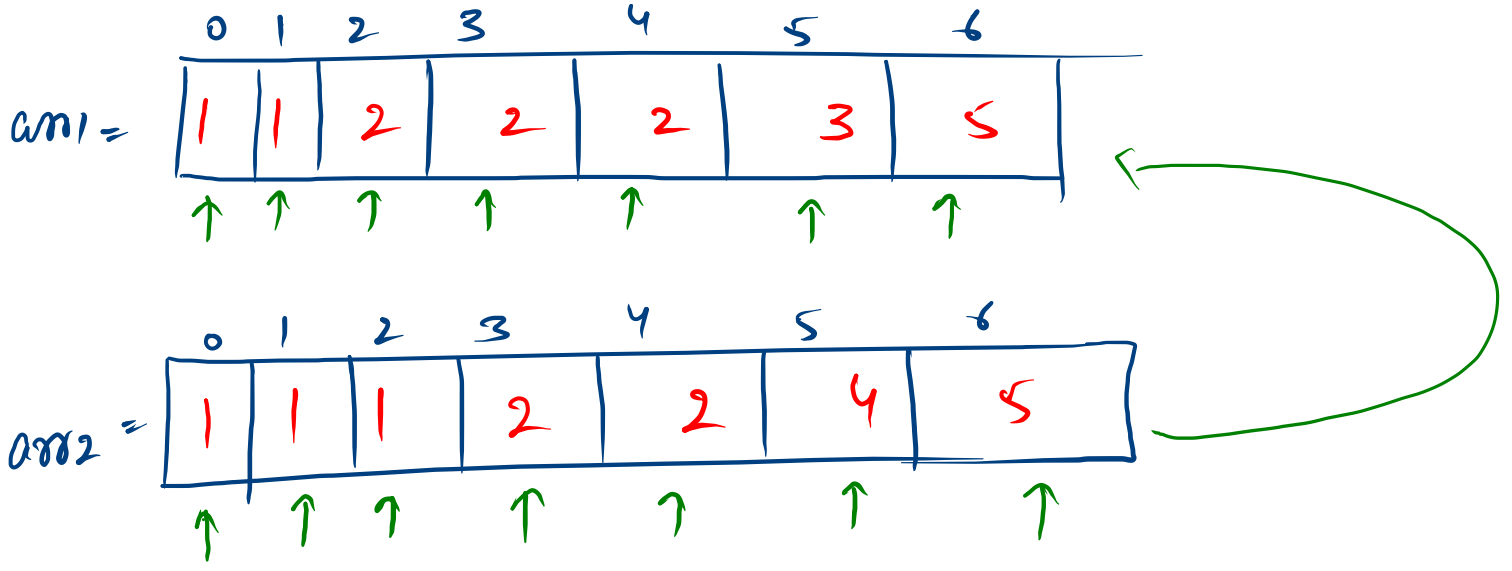
a

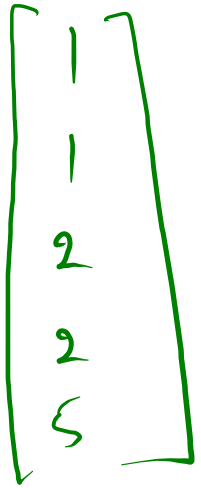


Integer	Boolean
~~5~~	~~true~~
~~9~~	~~true~~
~~8~~	~~true~~
~~0~~	~~true~~
~~3~~	~~true~~

9
0
3
5
8


Integer
Integer
1
2
3
5


arr1 =
0 1 2 3 4 5 6
1 1 2 2 2 3 5
arr2 =
0 1 2 3 4 5 6
1 1 1 2 2 4 5


1
1
2
2
5

Hashmap → [Longest Consecutive seq of elements]

BT → [Is balanced]

BST → [Replace with sum of larger, Constructor]

Heap → 1

Hash & Heap → 1

Graph → 3

Mon Tues Wed