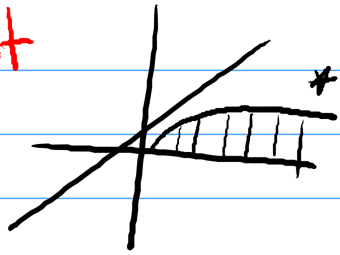
JoeyIssa

Everything You need to Know about Big O!



$O(1) > O(\log n) > O(n) > O(n\log n) > O(n^2) > O(n^3) > O(2^n)$

1.
```
int sum()
{
  sum=0;
  for(int i=0; i<10; i++)
  {
      sum = sum+1;
  }
  return sum;
}
```

0, ... 10 @ max 10 times

$O(1)$

```
for i = 1 to n do
    j = n
    while j ≥ 1 do
        Print (i, j)
        j = ⌊j/2⌋
    end while
end for
```

$1 \ldots n \rightarrow O(n)$

$j = n \geq 1$

↳ $n/2$ everytime

$O(\log_2 n)$

$\underline{O(n \log n)}$

$j--=1$ $O(n) \rightarrow O(n^2)$

Input : The number n is an integer greater than one. Assumed

```
Sum = 0
i = 1
while i ≤ n do
    j = 1
    while j ≤ 100 do
        Sum = Sum + 1
        j = j + 2
    end while
    i = i + 1
end while
```

$1 \ldots n \rightarrow 1, 2, 3 \ldots n$ $O(n)$

$1 \ldots 100 \rightarrow 1 \rightarrow 100$ $O(100)$

$O(c)$

$O(1)$

$O(n) * O(1)$

$\rightarrow O(n)$

Joey Issa

```
i=1;
while(i<=n){
        i=2*i;
}
```

1 ...... n

2*i $O(\log_2 n)$

0 .... n

```
for(i = 0;i < n; i++)
    for(j = 0; j< n; j++)
        sum += i*j;
```

sum →

0 .... n

$O(n) * O(n)$

$\rightarrow O(n^2)$

```
int sum = 0;
for (int i = 1; i < N; i *=2)
{
    for (int j = 0; j < N; j++)
    {
        ArrayX[j]= ArrayY[i];
        sum++;
    }
}
```

1 ... N i*2 $O(\log_2 n)$

0 ... N j++ $O(n)$

×

$O(n \log n)$

Bonus !!

Input: The number n is divisible by 4.

for $i = 2$ to n **do**
  for $j = 0$ to n **do**
    Print (i, j)
    $j = j + \lfloor n/4 \rfloor$
  end for
end for

i : 2 ..... n $O(n)$

j : 0 .... n $O(n)$

4

$j = j + [n/4]$

n

$O(1) \cdot O(n) \rightarrow O(n)$

Find the complexity of an algorithm that determines a patient between 100 Normal people using big O notation.

0 100

$O(100) \longrightarrow O(1)$

$\frac{n}{2}$ $\frac{\infty}{2} = \infty$

$i = 0$ $j = 0$ $K = 0$

```
int mycode(int n) {
    int i, j, k = 0;
    for (i  = n/2; i <= n; i++)
        for (j = 2; j <= n; j = j * 2)
            k = k + 2;
    return k;
}
```

n

$i = n : i \leq n : i{+}{+}$

$2 \ldots n$ $\underline{j * 2}$ $O(n \log n)$

$O(\log_2 n)$

$i = n : i \leq n : i{+}{+}$ $O\left(\frac{n}{2}\right) \to O(n)$

Joey Issa

```java
/** Returns true if there are no duplicate elements in the array. */
public static boolean unique1(int[ ] data) {
  int n = data.length;
  for (int j=0; j < n−1; j++)
    for (int k=j+1; k < n; k++)
      if (data[j] == data[k])
        return false;                    // found duplicate pair
  return true;                           // if we reach this, elements are unique
}
```

0 .... n-1

0 ... n

j .... n

k .... j+1

j	k
0	1
1	2
3	4
n-1	n

0 .... n

0 .... n

$O(n^2)$

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

$O(n)$ geometric

$O(n^2)$

$\frac{n^2 - n}{2}$

$\frac{n^2}{2} - \frac{n}{2}$

PreOrder, InOrder, PostOrder

CLR LCR LRC

LR

R-8.22 Draw a binary tree *T* that simultaneously satisfies the following:

- Each internal node of *T* stores a single character.
- A *preorder* traversal of *T* yields EXAMFUN.
- An *inorder* traversal of *T* yields MAFXUEN.

IN LCR

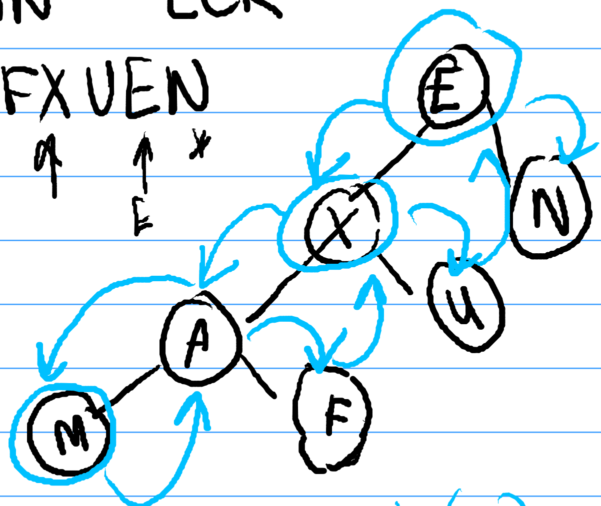
pre EXAMFUN

CLR

1st

MAFXUEN

E



check if have time

in : MAFXUEN

pre : EXAMFUN

print ( )

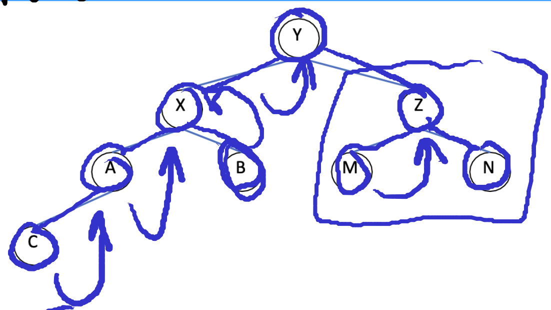
preorder (node→left)

preorder (node→right)

Joey Issa

Inorder → LCR (print)   PreOrder → CLR   PostOrder → LRC

Traversals



In: CAXBYMZN

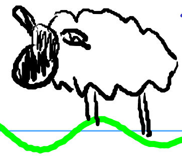
LCR

In(node → left) → in(Z → M)
print
In(node → Right)   in(Z → N)

pint(M)
print(Z)
pirint(N)

Heaps

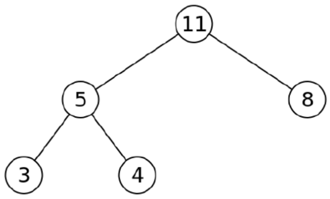
Bah!



→ sheep!

Joey Issa

Question 11 [3 points] Consider the following max-heap:



and the following operations :

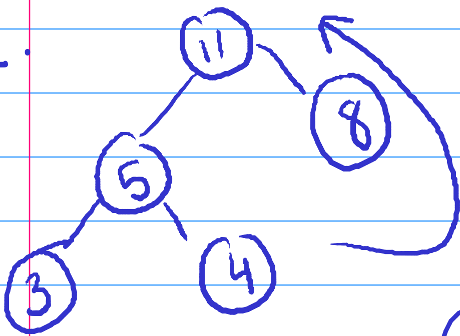
`insert(key)` : insert a node with the value key in the heap

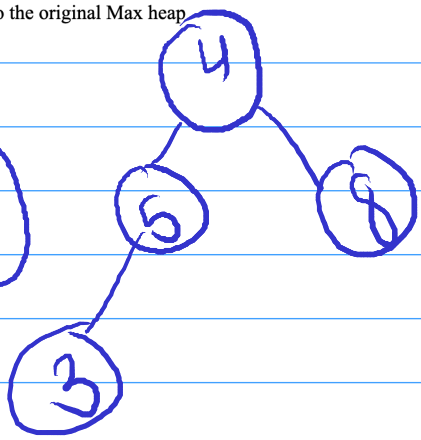
`removeMax()` : delete and return the element of maximum key in the heap

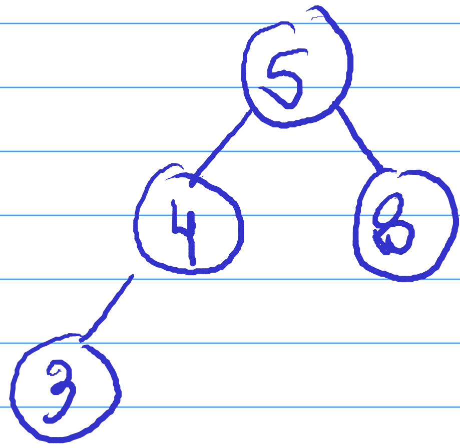
1- Draw the Max heap after Remove Max

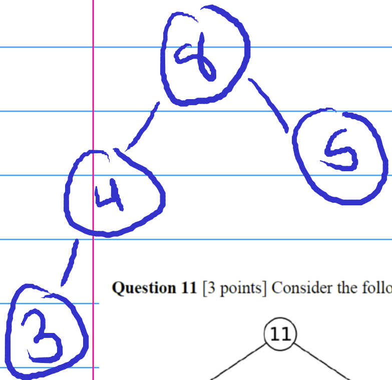
2 – Draw the Max Heap after you insert 14 to the original Max heap

1.

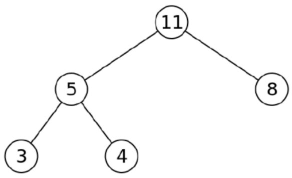


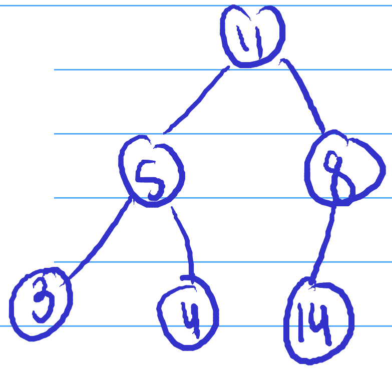


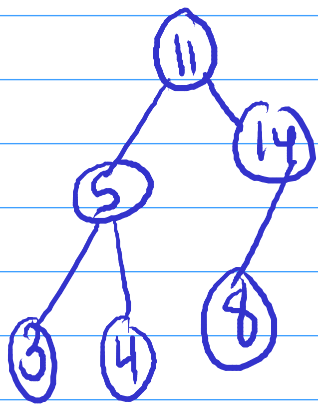




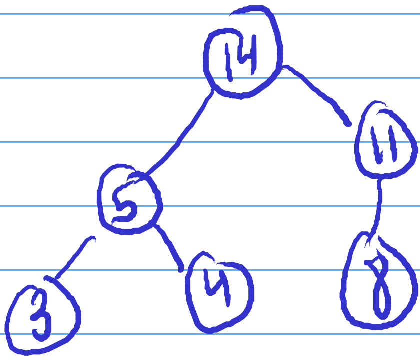
Question 11 [3 points] Consider the following max-heap:





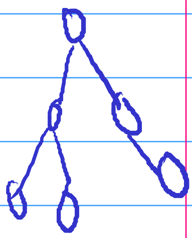


Insert 14

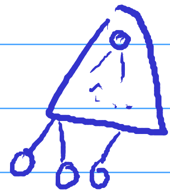


Joey Issa

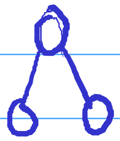
Complete, full, Perfect << Fast Review >>



Complete – Perfect tree from Height $(h-1)$ w >1 leaves @ h from left out



full — each node has two children



formulas

Perfect – (full + Complete) full binary tree with all leaves @ the same level

$h = \#$

$n = \#$

$i = (n-1)/2$ * $e = i + 1$ $n = i + e$ $n = 2e - 1$

internal nodes, # nodes, external nodes, total nodes

$$\boxed{n = 2^{h+1} - 1}$$ perfect trees

can assume it's full too

if you have a P. tree w heigh $= 4$ how many node?

$n = 2^4 - 1$ $n = 15$

How many external

$n = 2e - 1$ $e = 8$

$\frac{n+1}{2} = e$

$e = i + 1$

$e - 1 = i$

How many internal $i = 7$

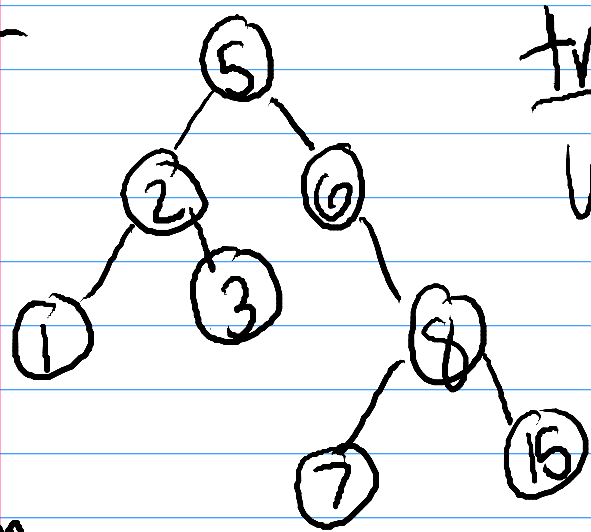
Joey Issa

You have numbers in this order: 5, 2, 6, 8, 15, 1

Create BST, Min Heap, Max heap, Avl

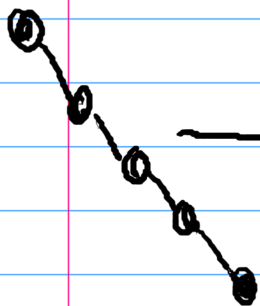
BST

1, 3



trick Questions

What is the worst complexity of finding a node in a tree?

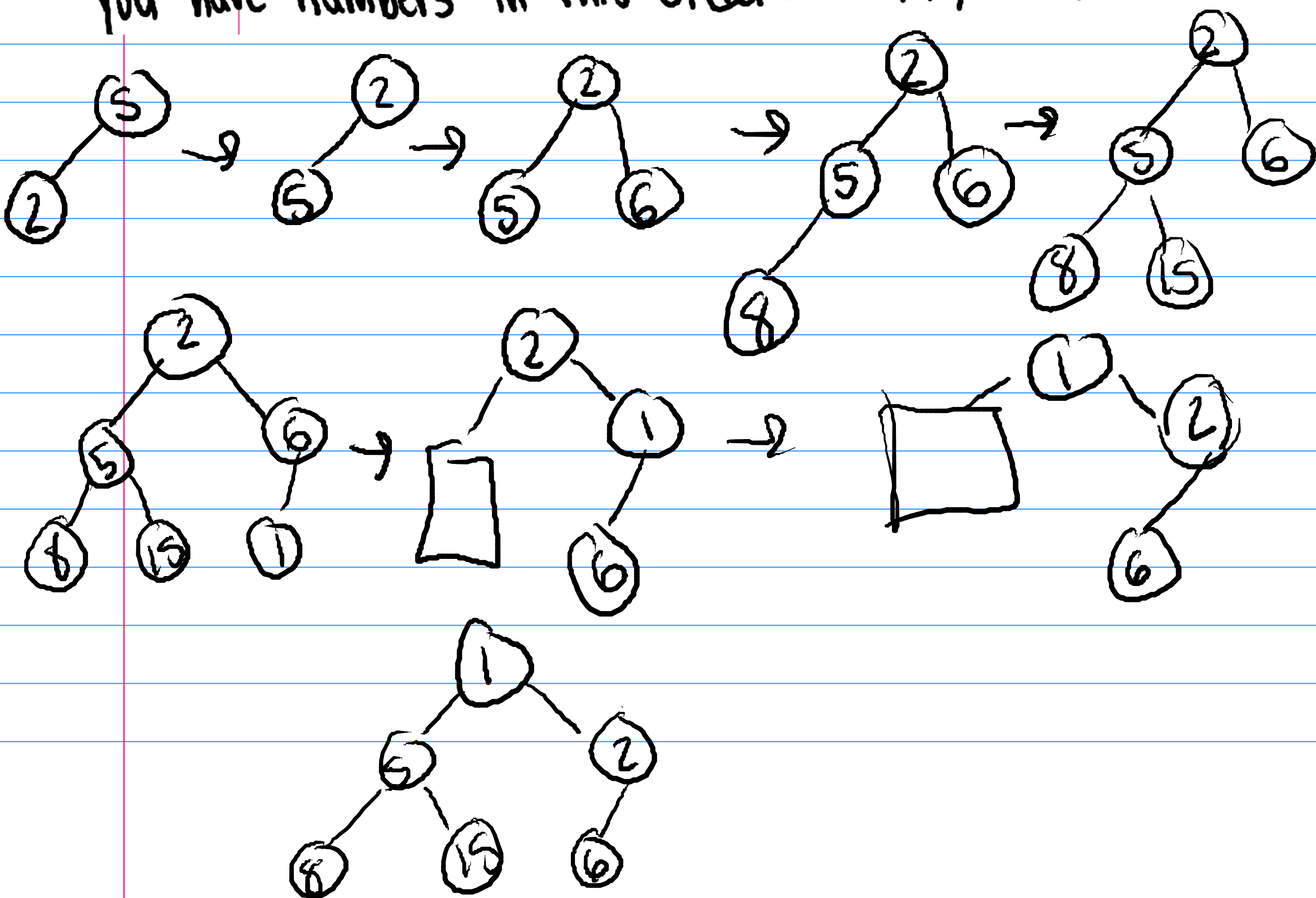


→ $O(n)$

What ...... BST

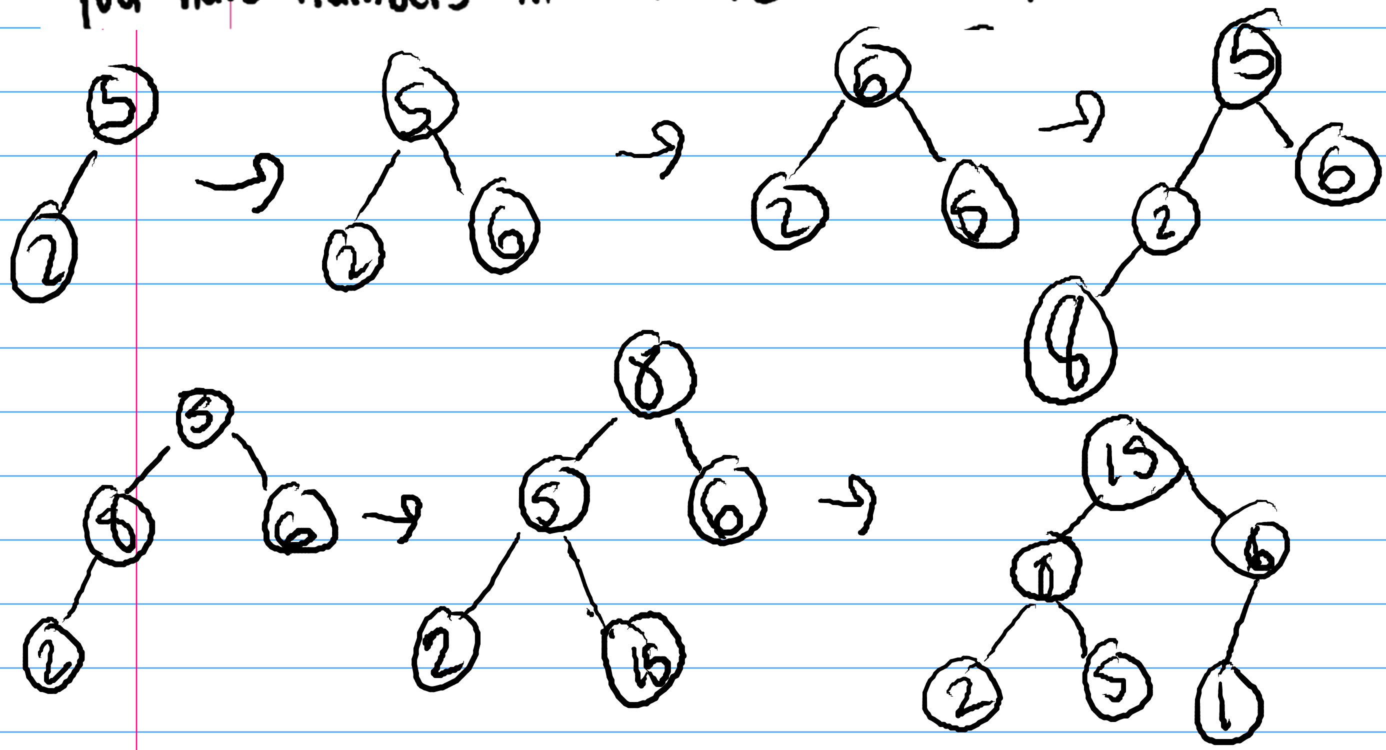
→ $O(\log n)$

You have numbers in this order: 5, 2, 6, 8, 15, 1

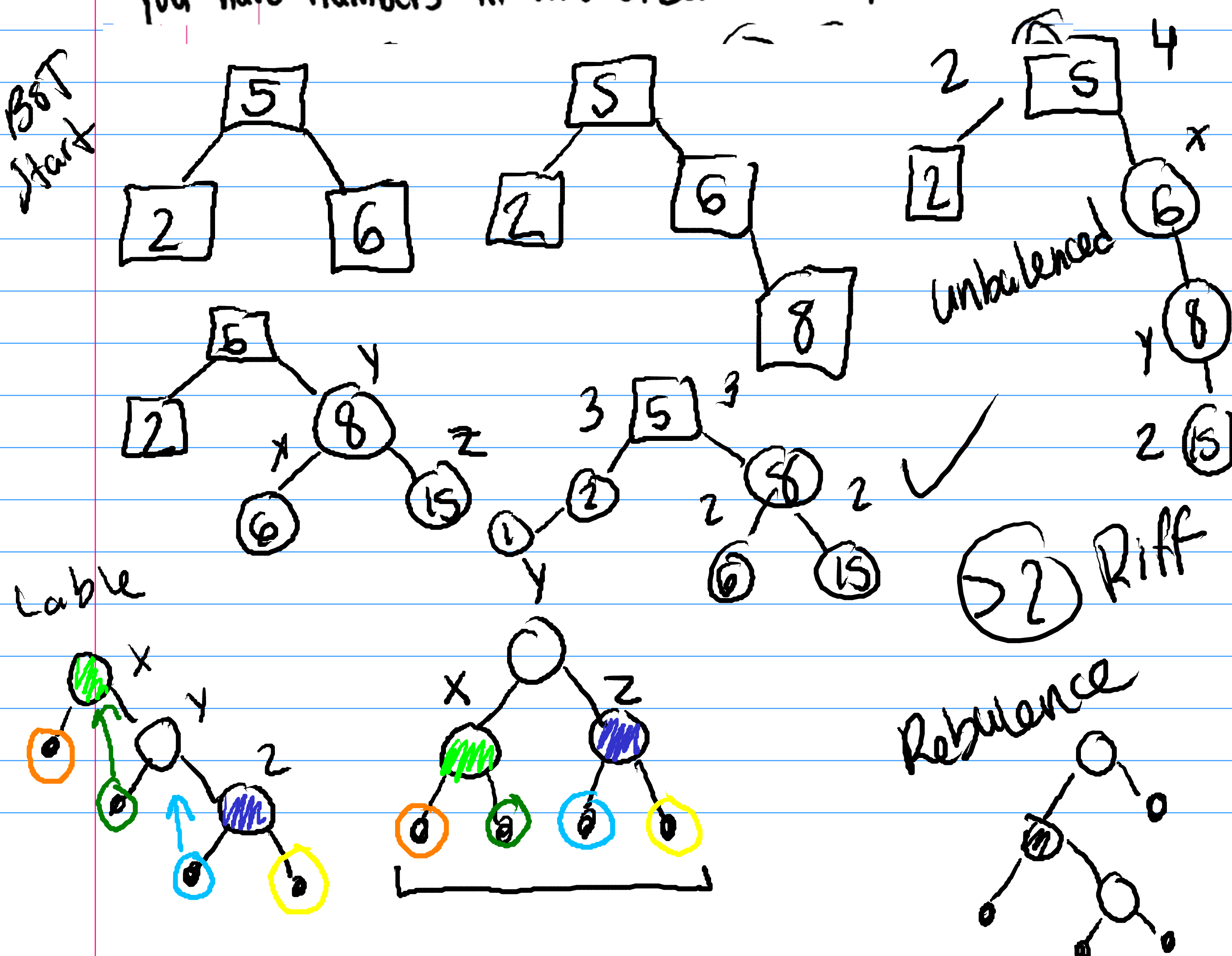


Joey Issa

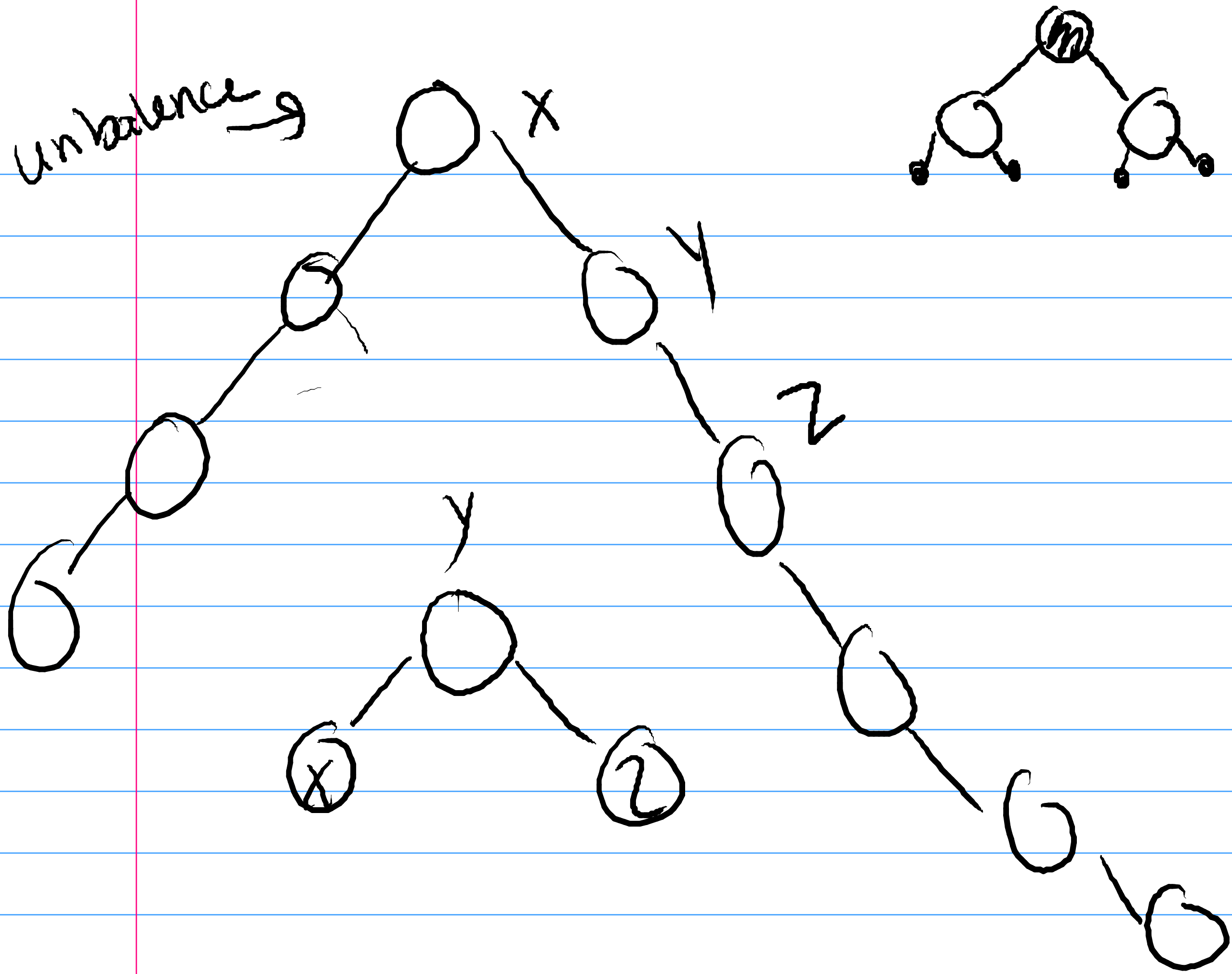
You have numbers in this order: 5, 2, 6, 8, 15, 1



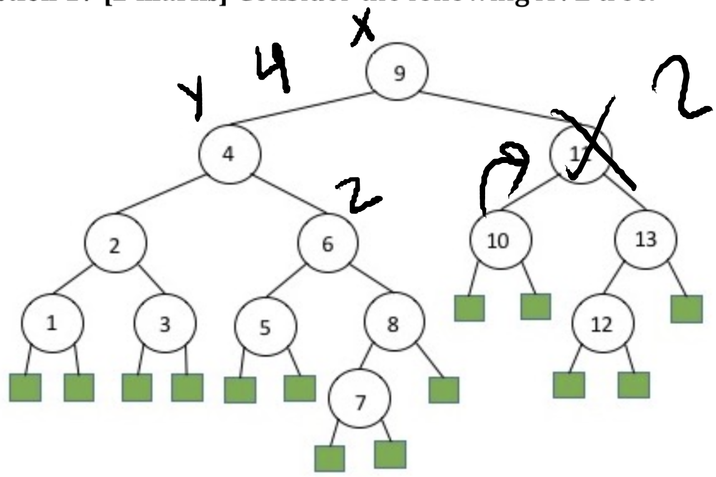
You have numbers in this order: 5, 2, 6, 8, 15, 1

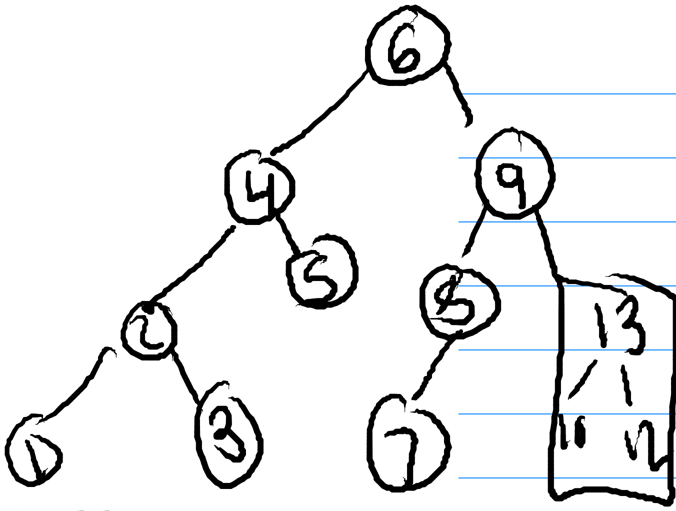


unbalance →

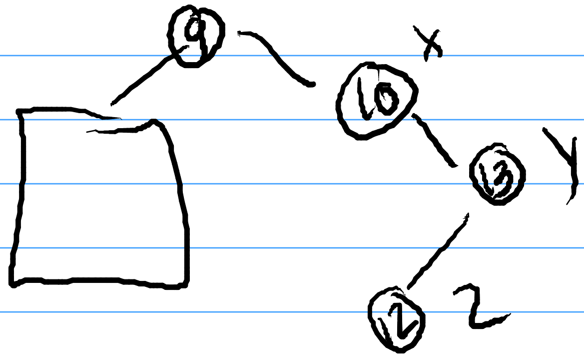


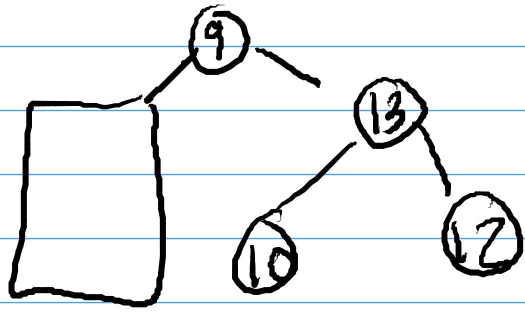
Question 17 [2 marks] Consider the following AVL tree.



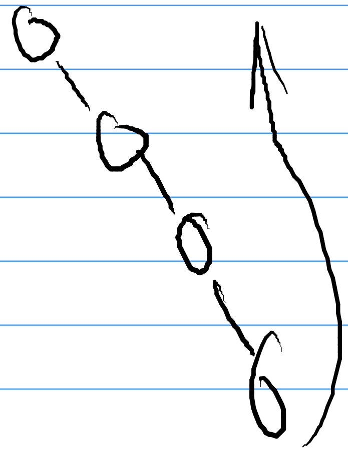


Show the relevant steps (tree transformations) after **the operation delete 11**, assuming the first step in the deletion operation selects **element 10 to substitute the deleted element 11.**





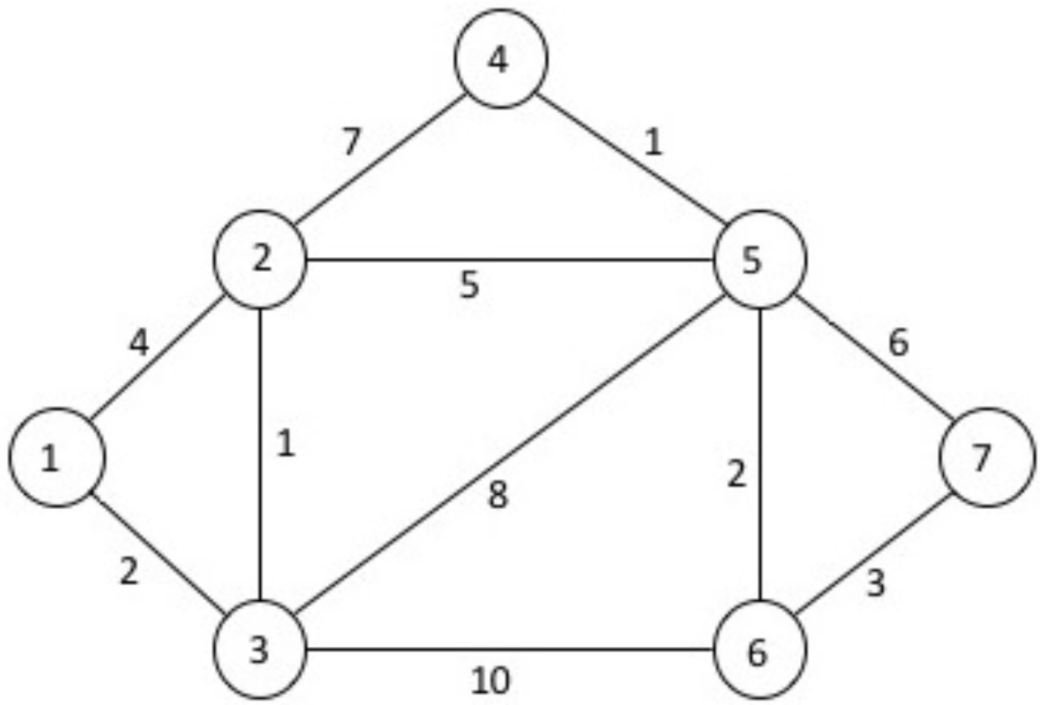
Which of the following alternatives is a true completion of the phrase:
The worst case running time of searching for a key in an AVL tree is (i) log n but the worst case running time of searching for a key in a binary search tree is (ii) O(n)



Dijkstra + Prim + Kruskal

BFS, DFS

Joey Issa



Joey Issa

Hashing

yum! 

JoeyIssa

Question 10 [1 point] Consider the following Hash table where insertions are done using the hash function h(k)= k mod 7, and collisions are resolved with **quadratic probing**.

0	1	2	3	4	5	6
	15	**1**	**8**		**5**	

Regarding the ordering of insertion in this table, which answer is CORRECT?

A) Key 1 was the last key to be inserted.
B) Key 8 was the last key to be inserted.
C) Key 15 was the last key to be inserted.
D) It is impossible to determine which was the last key among 8 and 5.
E) None of the above is correct.

Question 11 [1 point] Consider the following Hash table where insertions are done using the hash function h(k)= k mod 7, and collisions are resolved with **linear probing**.

0	1	2	3	4	5	6
	15	**1**	**8**		**5**	

What is the **average number of probes** A for searching an existing key in this table?

A) A=1
B) $1 < A < 2$
C) A= 2
D) A >2
E) None of the above is correct.

Question 12
Suppose you **insert element 2** in the table given in Question 11, still using linear probing. After this, you search for element 3. How many table positions must be probed until you conclude element 3 is not in the table?

A) 2 B) 3 C) 4 D) 5 E) 6 or more

Sorting Algorithms

Question 6 [1 point] The worst case running times of the following sort algorithms are:

	Insertion sort	Mergesort	Quicksort
A)	Θ(n log n)	Θ(n log n)	$\Theta(n^2)$
B)	$\Theta(n^2)$	$\Theta(n^2)$	Θ(n log n)
C)	$\Theta(n^2)$	Θ(n log n)	Θ(n log n)
D)	$\Theta(n^2)$	Θ(n log n)	$\Theta(n^2)$
E)	None of the above.		

Question 20 [3 points] Mergesort
Simulate the execution of the in-place Mergesort algorithm for the given array below. Suppose that at the end of each "merge" step, the algorithm prints a line with the current contents of the full array.
Show each printout of this algorithm, and highlight the part of the array where the merge has been done. The number of blank arrays below may be more or less than the amount you need to show (use the back if more space is needed).

30	10	50	20	15	7	2	12