Sortering

DATAMATIKER-B

2021-05-05 LYNGBY

Agenda

Recap - Binary Trees

Sorting Algorithms

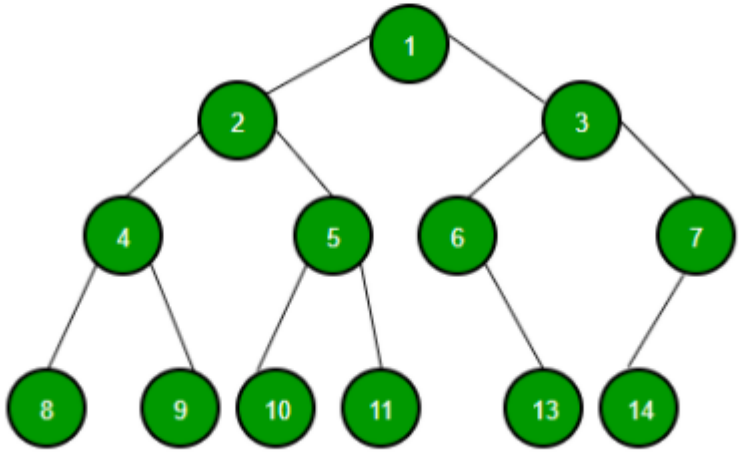
Exercises

Recap

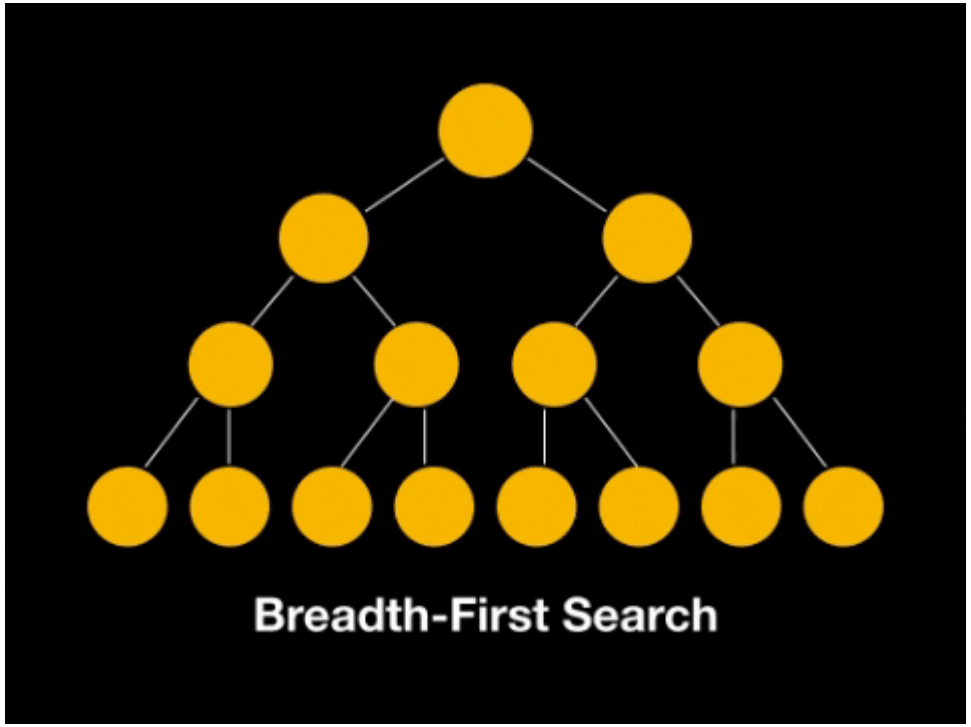
Recap - Binary Trees

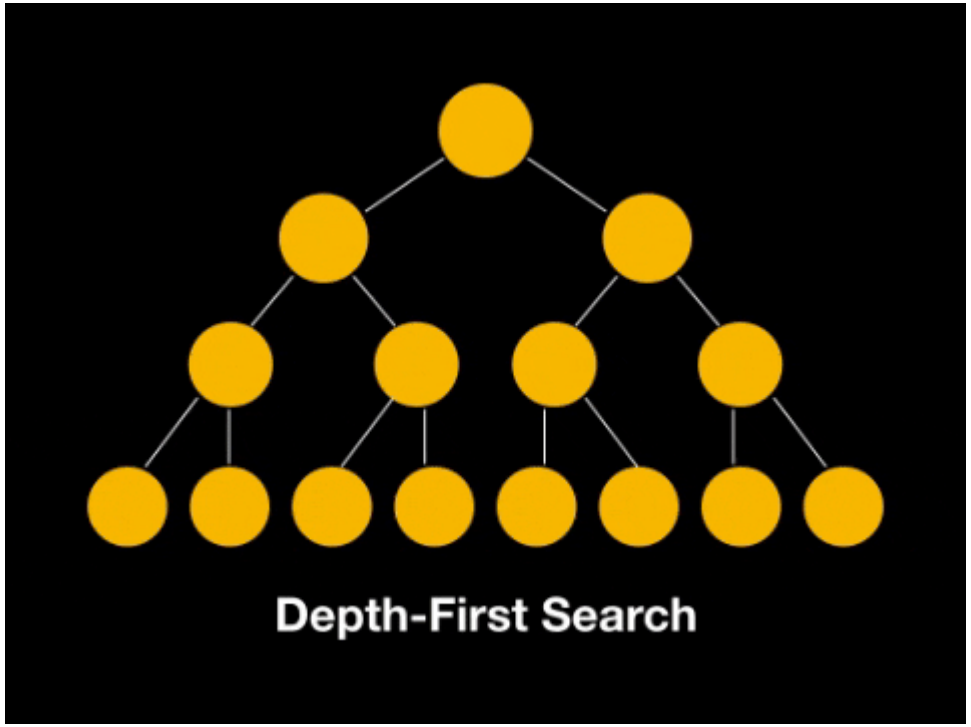
Binary Tree Data Structure

A tree whose elements have at most 2 children is called a binary tree. Since each element in a binary tree can have only 2 children, we typically name them the left and right child.

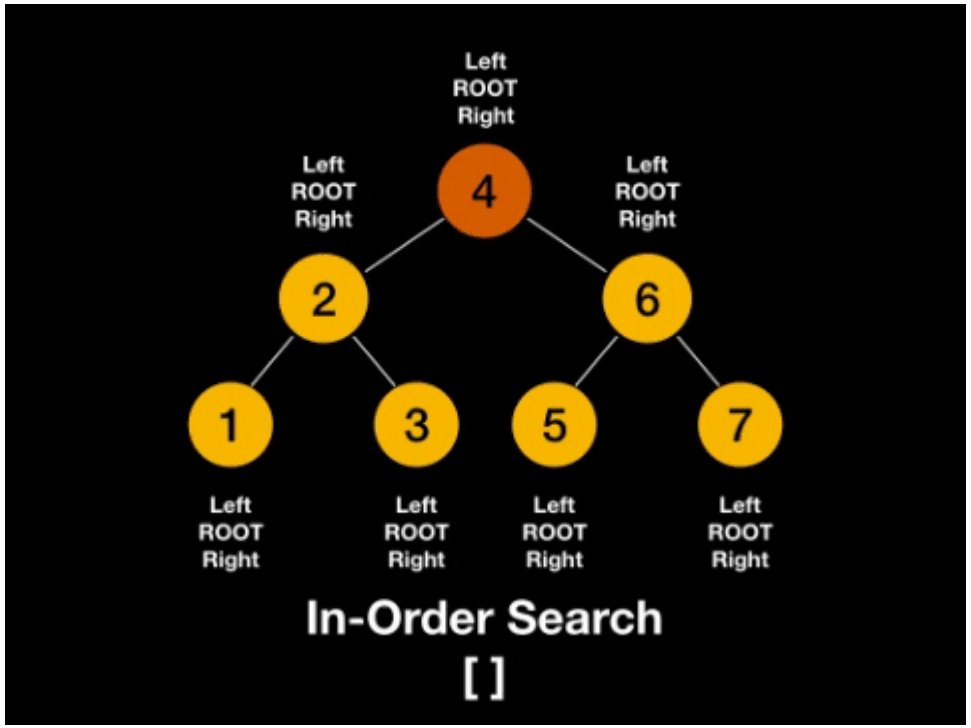


Recap - Traversal – BFS or DFS?



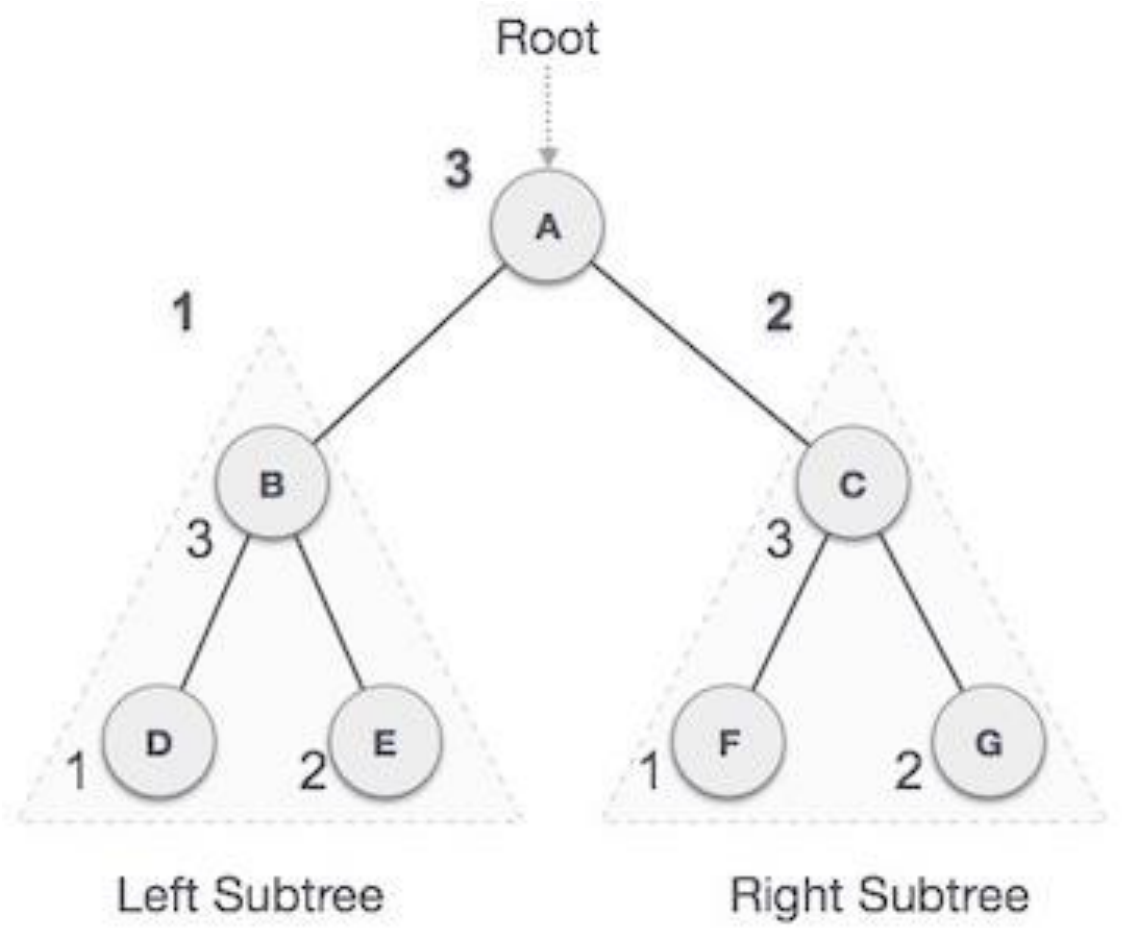


Recap - Traversal



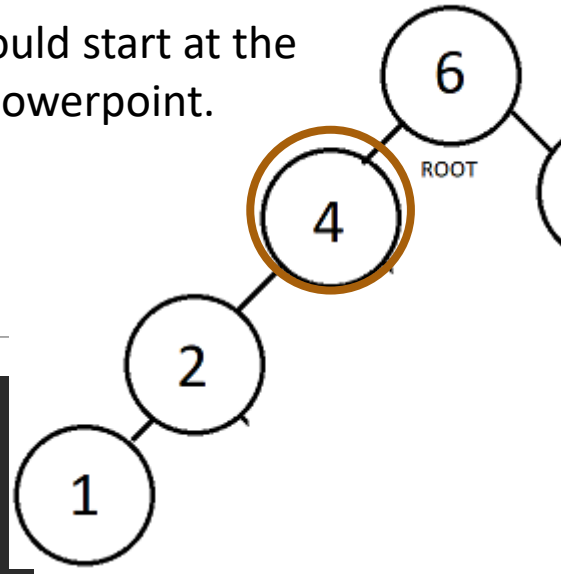
Note:
In-Order = { left, ROOT, right };
Pre-Order = { ROOT, left, right };
Post-Order = { left, right, ROOT };

Recap - Recursion



Recap - Recursion

Obviously, it should start at the root note, but powerpoint.

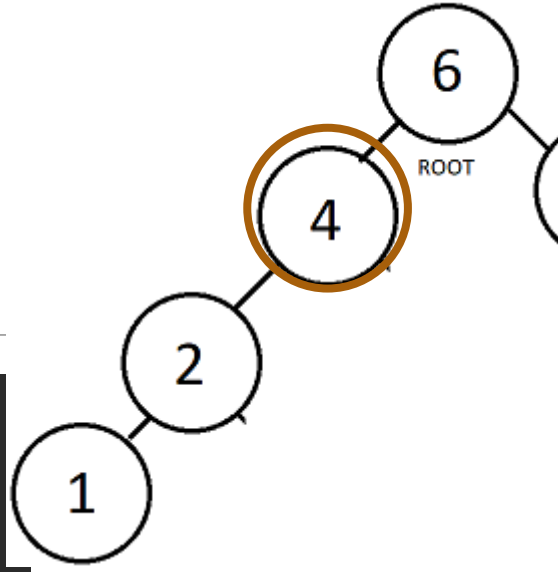



```java
private static void PrintBinaryTreeDFS(Node node)
{
    if (node != null) {
        PrintBinaryTreeDFS(node.left);
        System.out.print(" " + node.getValue());
        PrintBinaryTreeDFS(node.right);
    }
}
```

```
Output:
[]
```

Recap - Recursion

```
private static void PrintBinaryTreeDFS(Node node)
{
    if (node != null) {
        PrintBinaryTreeDFS(node.left);
        System.out.print(" " + node.getValue());
        PrintBinaryTreeDFS(node.right);
    }
}
```

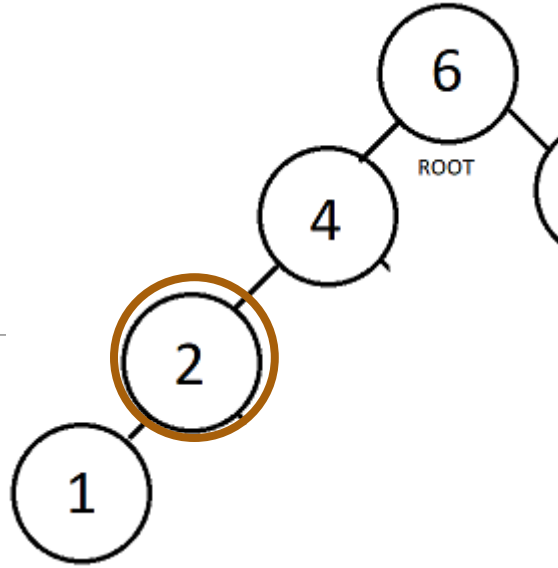



```
Output:
[]
```

Recap - Recursion

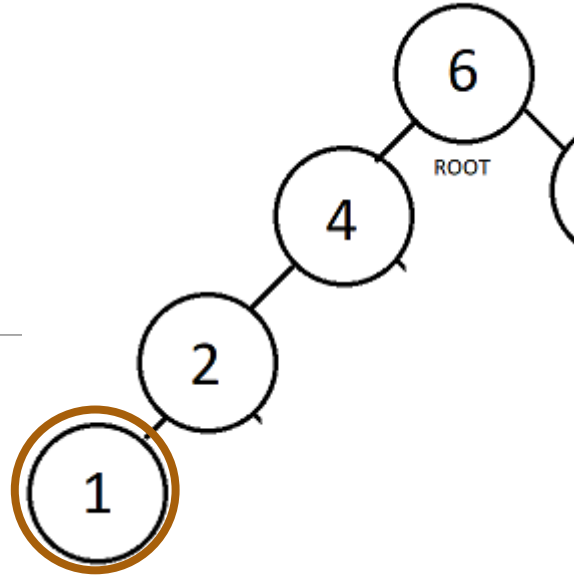
```
private static void PrintBinaryTreeDFS(Node node)
{
    if (node != null) {
        PrintBinaryTreeDFS(node.left);
        System.o
        PrintBin
    }
}
```

```
private static void PrintBinaryTreeDFS(Node node)
{
    if (node != null) {
        PrintBinaryTreeDFS(node.left);
        System.out.print(" " + node.getValue());
        PrintBinaryTreeDFS(node.right);
    }
}
```




```
Output:
[]
```

Recap - Recursion

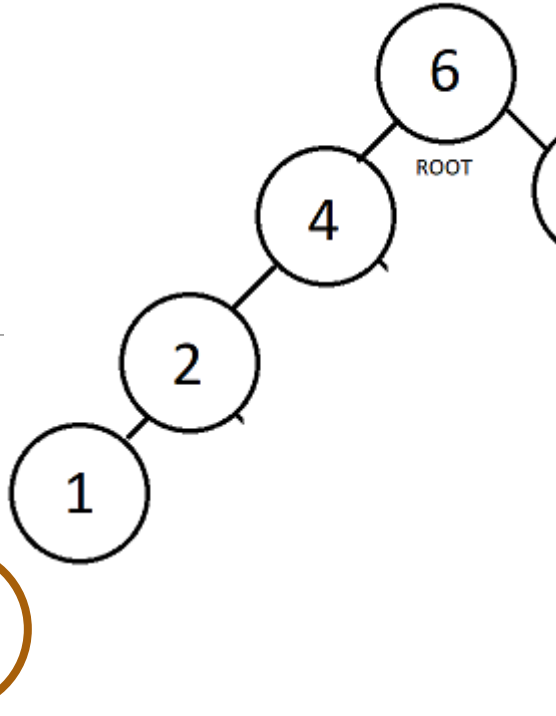



```
private static void PrintBinaryTreeDFS(Node node)
{
    if (node != null) {
        PrintBinaryTreeDFS(node.left);
        System.out.print(" " + node.getValue());
        PrintBinaryTreeDFS(node.right);
    }
}
```

Output:

[]

Recap - Recursion



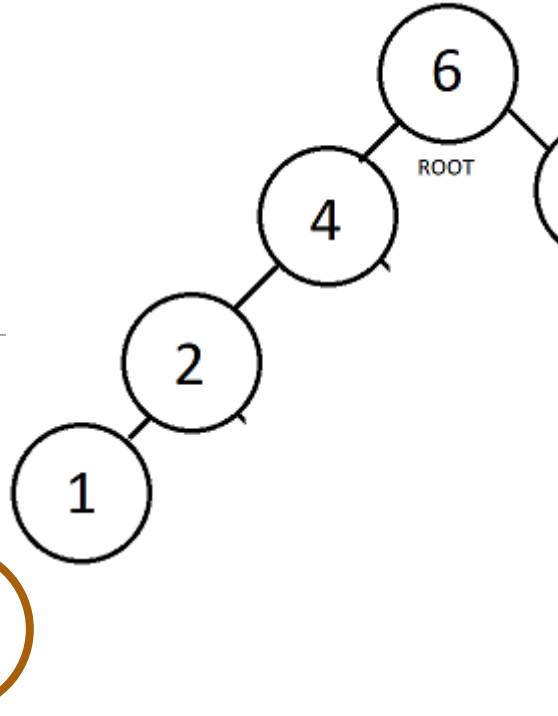

```
private static void PrintBinaryTreeDFS(Node node)
{
    if (node != null) {
        PrintBinaryTreeDFS(node.left);
        System.out.print(" " + node.getValue());
        PrintBinaryTreeDFS(node.right);
    }
}
```

```
private static void PrintBinaryTreeDFS(Node node)  node: null
{
    if (node != null = false ) {   node: null
        PrintBinaryTreeDFS(node.left);
        System.out.print(" " + node.getValue());
        PrintBinaryTreeDFS(node.right);
    }
}
```

Output:

[]

Recap - Recursion



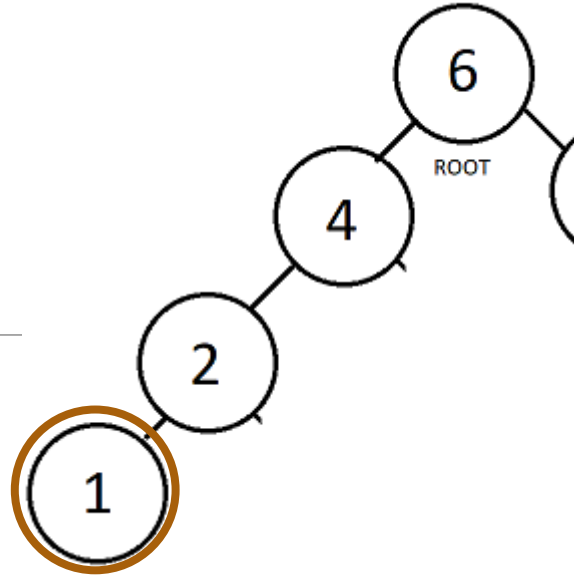

```
private static void PrintBinaryTreeDFS(Node node)
{
    if (node != null) {
        PrintBinaryTreeDFS(node.left);
        System.out.print(" " + node.getValue());
        PrintBinaryTreeDFS(node.right);
    }
}
```

Output:

[]

Recap - Recursion

```
private static void PrintBinaryTreeDFS(Node node)
{
    if (node != null) {
        PrintBinaryTreeDFS(node.left);
        System.out.print(" " + node.getValue());
        PrintBinaryTreeDFS(node.right);
    }
}
```

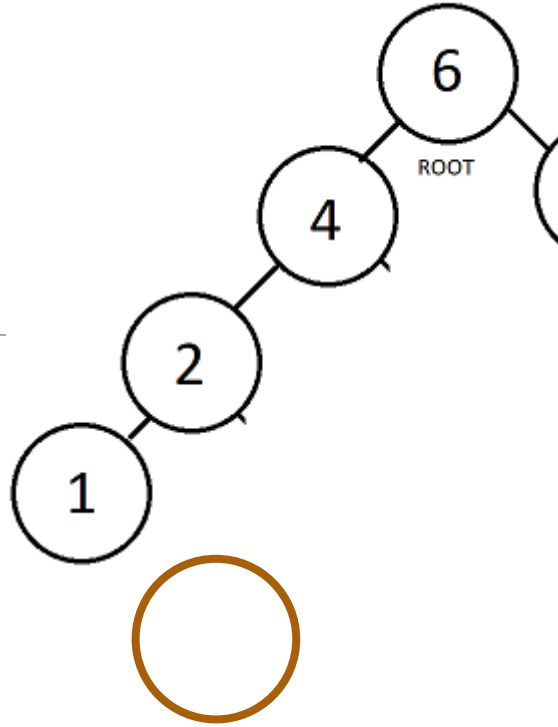



```
Output:
[1]
```

Recap - Recursion

```
private static void PrintBinaryTreeDFS(Node node)
{
    if (node != null) {
        PrintBinaryTreeDFS(node.left);
        System.out.print(" " + node.getValue());
        PrintBinaryTreeDFS(node.right);
    }
}
```

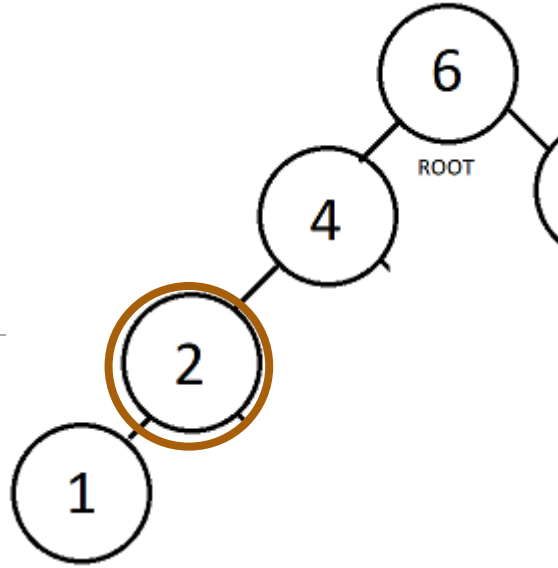
Output:
[1]



Recap - Recursion

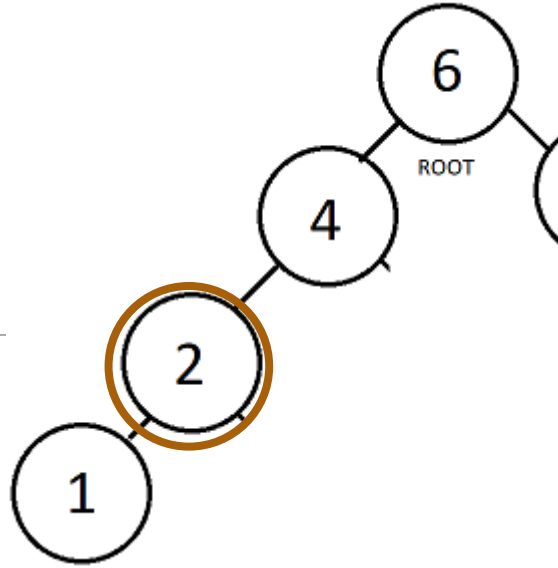
```
private static void PrintBinaryTreeDFS(Node node)
{
    if (node != null) {
        PrintBinaryTreeDFS(node.left);
        System.out.print(" " + node.getValue());
        PrintBinaryTreeDFS(node.right);
    }
}
```

Output:
[1]



Recap - Recursion

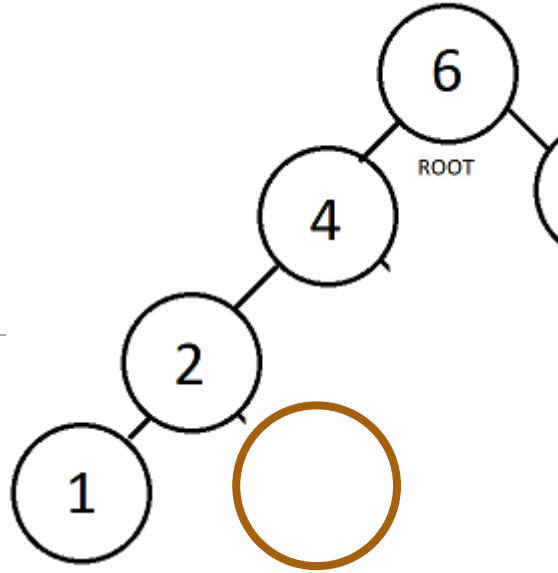
```
private static void PrintBinaryTreeDFS(Node node)
{
    if (node != null) {
        PrintBinaryTreeDFS(node.left);
        System.out.print(" " + node.getValue());
        PrintBinaryTreeDFS(node.right);
    }
}
```




```
Output:
[1 2]
```

Recap - Recursion

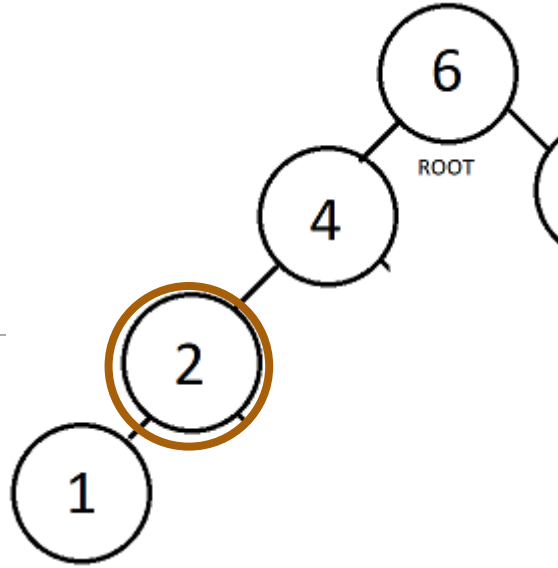
```
private static void PrintBinaryTreeDFS(Node node)
{
    if (node != null) {
        PrintBinaryTreeDFS(node.left);
        System.out.print(" " + node.getValue());
        PrintBinaryTreeDFS(node.right);
    }
}
```




```
Output:
[1 2]
```

Recap - Recursion

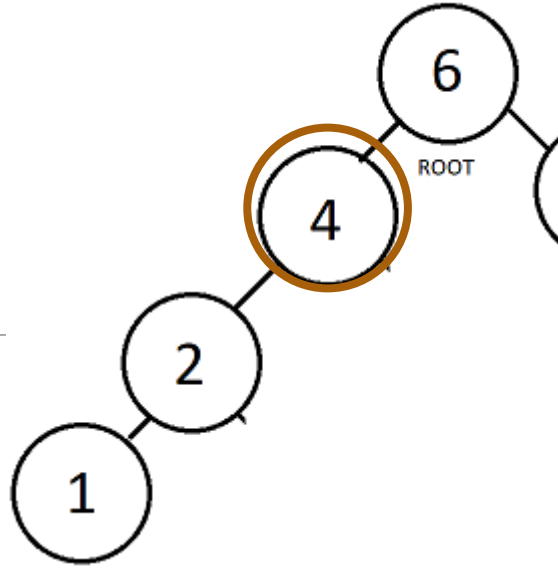
```
private static void PrintBinaryTreeDFS(Node node)
{
    if (node != null) {
        PrintBinaryTreeDFS(node.left);
        System.out.print(" " + node.getValue());
        PrintBinaryTreeDFS(node.right);
    }
}
```




```
Output:
[1 2]
```

Recap - Recursion

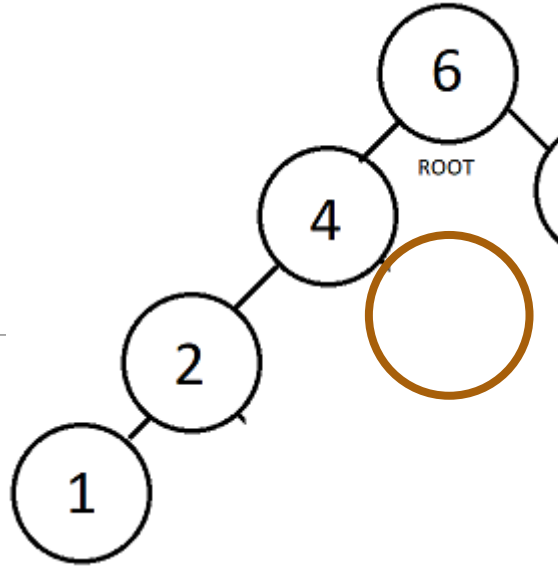
```
private static void PrintBinaryTreeDFS(Node node)
{
    if (node != null) {
        PrintBinaryTreeDFS(node.left);
        System.out.print(" " + node.getValue());
        PrintBinaryTreeDFS(node.right);
    }
}
```




```
Output:
[1 2 4]
```

Recap - Recursion

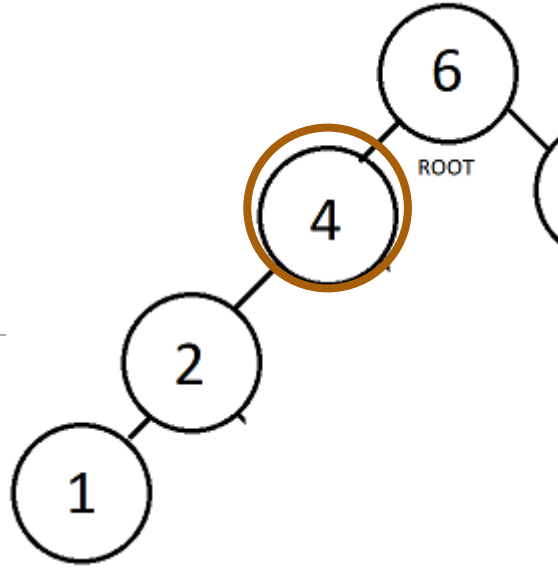
```
private static void PrintBinaryTreeDFS(Node node)
{
    if (node != null) {
        PrintBinaryTreeDFS(node.left);
        System.out.print(" " + node.getValue());
        PrintBinaryTreeDFS(node.right);
    }
}
```




```
Output:
[1 2 4]
```

Recap - Recursion

```
private static void PrintBinaryTreeDFS(Node node)
{
    if (node != null) {
        PrintBinaryTreeDFS(node.left);
        System.out.print(" " + node.getValue());
        PrintBinaryTreeDFS(node.right);
    }
}
```




```
Output:
[1 2 4]
```

Note: obviously there would be another recursion, so the root not would also be printed in this code example, but at this point, I suffered dead by powerpoint.

Sorting algorithms

Why?

Oftentimes, sorting will reduce the complexity of a problem. Remember the video shown yesterday with the google interview –the complexity of his solution arose, when there was no longer guaranteed a sorted input.

Sorting (numerical or lexicographical)

```
Collections.sort(list);
Collections.sort(list, Collections.reverseOrder());
```

Collections.sort

Uses a Merge Sort Algorithm

Other common sorting algorithms:

- Selection sort
- Bubble sort
- Insertion sort
- Quick sort
- Heap sort

Why is this relevant?

Most of the times, it really isn't!

We're talking possible performance improvements that might be really tiny. But the bigger the dataset, the bigger the performance impact.

Sometimes, even though the performance impact might actually be worhtwhile, it is still used in a context where there might be enough time to have a poor performance-wise sorting algorithm.

https://www.youtube.com/watch?v=ZZuD6iUe3Pc&ab_channel=ViktorBohush

Pause

Comparators

So what if our data we need to sort isn't numerical nor lexicographical?

Well, then Comparators!

https://www.tutorialspoint.com/java/java_using_comparator.htm

Comparators

```
// Overriding the compareTo method
  public int compareTo(Dog d) {
     return (this.name).compareTo(d.name);
  }

  // Overriding the compare method to sort the age
  public int compare(Dog d, Dog d1) {
     return d.age - d1.age;
  }
```

Heap Sort

Input 35 33 42 10 14 19 27 44 26 31

https://www.tutorialspoint.com/data_structures_algorithms/heap_data_structure.htm

Exercises: Implement sorting algorithms

Input array = { 2, 7, 9, 15, 6, 3, 10, 14, 13, 1, 4, 5, 12, 8 };

YES – The code for all these can be found on the internet, but PLEASE try it yourself first! the purpose is to figure it out on your own (in groups)!

1. Bubblesort
https://www.tutorialspoint.com/data_structures_algorithms/bubble_sort_algorithm.htm

2. Insertion sort
https://www.tutorialspoint.com/data_structures_algorithms/insertion_sort_algorithm.htm

3. Selection sort
https://www.tutorialspoint.com/data_structures_algorithms/selection_sort_algorithm.htm

4. Quick sort
https://www.tutorialspoint.com/data_structures_algorithms/quick_sort_algorithm.htm