CS32: Introduction to Computer Science

Discussion Week 8

Junheng Hao, Ramya Satish

March 1, 2019

Announcement

UCLA Samueli Computer Science

- Homework 4 is due on next Tuesday, March 5.
- Homework 5 and Project 4 are both due on March 14.
- … and Project 3 is due 11PM yesterday...

Outline

UCLA Samueli Computer Science

- Sorting
- Tree

Big-O Notation

Formal definition

Definition: Let `T(n)` be the function that measures the runtime of the program given n size of input. And let `g(n)` be another function defined on the real number field.

$T(n) = O(g(n))$ iff $\exists M > 0$ and $\exists N > 0$ s.t. $\forall n>N : T(n) \leqq M*g(n)$

Well, you can simply understand as how many operations given input size of **n** regardless of the constant.

No need to memorize definitions.

Example: if your program takes,

- about `n` steps → `O(n)`
- about `2n` steps → `O(n)`
- about `n^2` steps → `O(n^2)`
- about `3n^2+10n` steps → `O(n^2)`
- about `2^n` steps → `O(2^n)`

Question: What is the speed of growth for typical function?

`f(n) = log(n) / n / n^2 / 2^n / n!`

Big-O Arithmetic

How to determine the entire program?

Generally,

- If things happen sequentially, we add Big-Os;
- If one thing happen within another, then we multiply Big-Os.
- Simple rule: Watch the LOOPS in your programs!

Rules:

$$O(f(n)) + O(g(n)) = O(\max(f(n), g(n)))$$
$$O(f(n)) \times O(g(n)) = O(f(n) \times g(n))$$

Sorting
Introduction

Most important algorithm ever!

Methods:

- Selection sort
- Bubble sort
- Insertion sort
- Merge sort
- Quick sort

Focus on:

1. Steps for each sorting algorithm
2. Runtime complexity for worst cases, best cases and average cases
3. Space complexity
4. How about additional assumptions, such as the array is “almost sorted” / “reversed” arrays

Sorting
Selection sort

Steps:

4	3	1	5	2
1	3	4	5	2
1	2	4	5	3
1	2	3	5	4
1	2	3	4	5

Idea: Find the smallest item in the unsorted portion and place it in the front.

Runtime complexity:

Average: $O(n^2)$

Worst: $O(n^2)$

Best: $O(n^2)$

Space complexity: $O(1)$

Sorting

Insertion sort

UCLA Samueli Computer Science

Steps:

4	3	1	5	2
3	4	1	5	2
1	3	4	5	2
1	3	4	5	2
1	2	3	4	5

Idea: Pick one from the unsorted part and place it in the right position.

Runtime complexity:

Average: $O(n^2)$

Worst: $O(n^2)$

Best: $O(n)$

Space complexity: $O(1)$

Sorting

Bubble sort

Steps:

4	3	1	5	2
3	4	1	5	2
3	1	4	5	2
3	1	4	2	5
1	3	2	4	5
1	2	3	4	5

Idea: Well, just “bubble” as its name

Runtime complexity:

Average: $O(n^2)$

Worst: $O(n^2)$

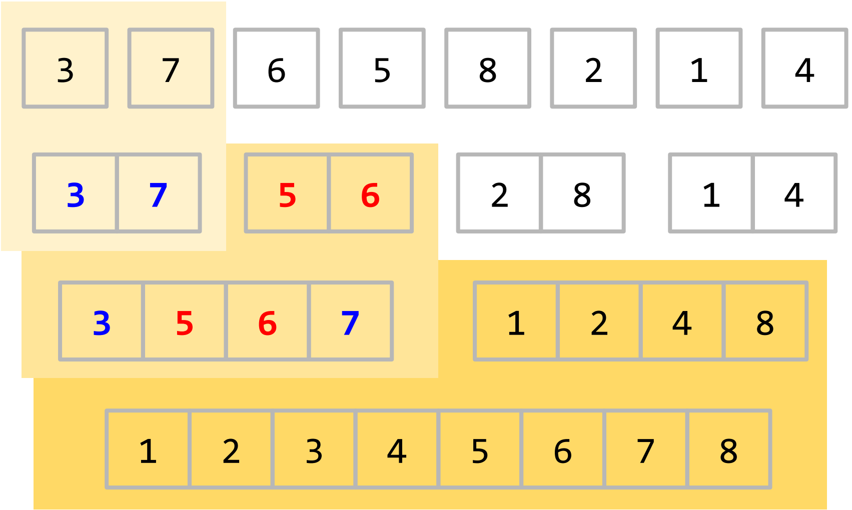
Best: $O(n)$

Space complexity: $O(1)$

Sorting

Merge sort

Steps:



Idea: Divide and conquer

Runtime complexity:

Average: $O(n \log n)$

Worst: $O(n \log n)$

Best: $O(n \log n)$

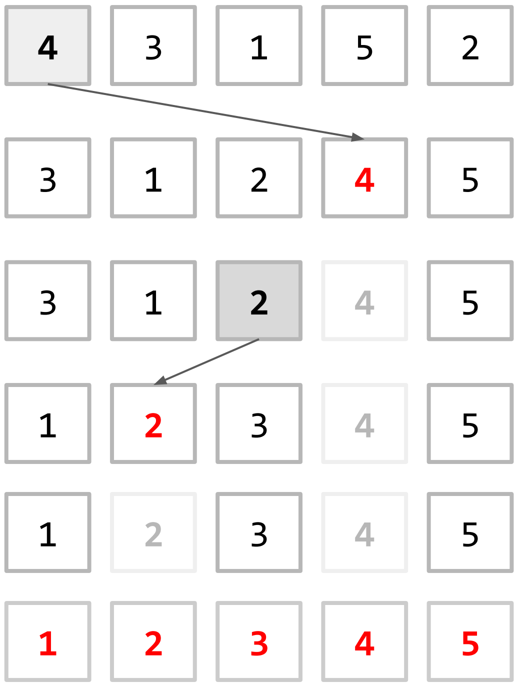
Space complexity: $O(n)$

Sorting

Quicksort

UCLA Samueli Computer Science

Steps:



Idea: Set a pivot. Numbers less then pivot are placed to front while other to end.

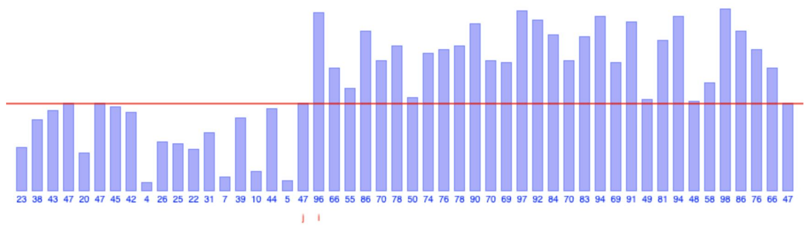
Runtime complexity:

Average: $O(n \log n)$

Worst: $O(n^2)$

Best: $O(n \log n)$

Space complexity: $O(\log n)$

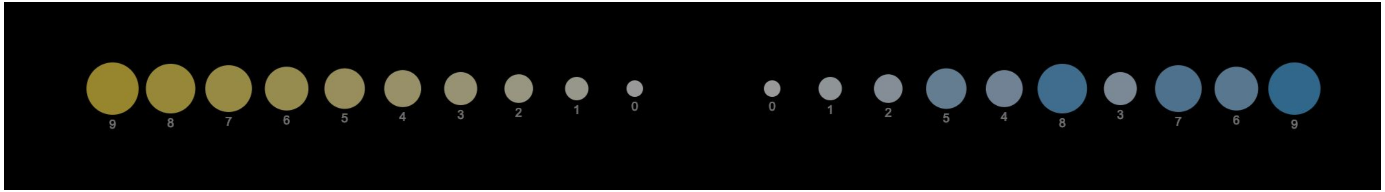


Sorting

Other methods and complexity?

- O(n log n) is faster than O(n^2) → Merge sort is more efficient than selection, insertion and bubble sort in runtime.
- O(n log n) is best average complexity that a general sorting algorithm can achieve.
- With more information about the data provided, you can sometimes sort things almost linearly.

Question: What is the complexity of these sorting algorithms if you know the array is **reversed**? What if the array is **almost already sorted**?

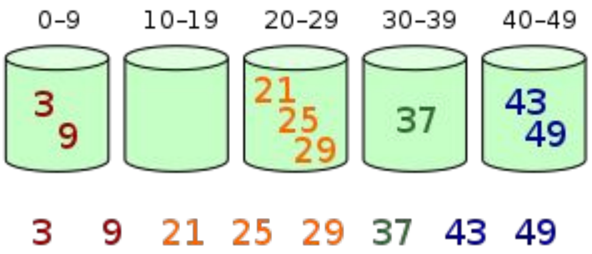


Sorting

Other methods and complexity?

There are many other sorting methods:

- Shell sort (shell 1959, Knuth 1973, Ciura 2001)
- Quicksort 3-way
- Heap sort
- Bucket sort



Sorting

Why sorting is important?

Sorting is the most important and basic algorithm. Many other real-world problems are somewhat based on sorting, including:

Sorting Algorithms Animations: https://www.toptal.com/developers/sorting-algorithms
Other good demos:
https://www.cs.usfca.edu/~galles/visualization/ComparisonSort.html
http://sorting.at/

Sorting - Problem 1
Find the K-th largest

UCLA Samueli Computer Science

Question: How about get the *K-th* largest numbers in one array?
Leetcode question #215

Hint:
1. How to find the k-th largest numbers by merge sort and quicksort (or other sort methods)? What are the average and worst complexity?
2. What data structures is good to use?
3. What about ***K* largest numbers (can be unordered)** instead of **the *K*-th largest number**?

Sorting - Problem 2

Find the median of streaming data

How to find the median of the streaming data?

That is, implementing the following program:

```
addNum(1)
addNum(2)
findMedian() -> 1.5
addNum(3)
findMedian() -> 2
```

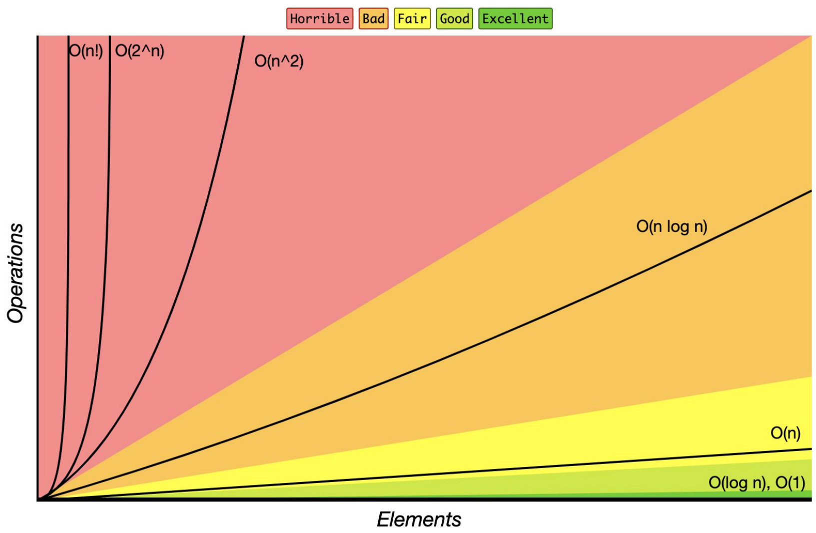
```
class MedianFinder {
public:
  MedianFinder() {
    // construction
  }

  void addNum(int num) {
    // add new integers from stream
  }

  double findMedian() {
    // return the median
  }
private:
    // define your private data member(s)
};
```

Big-O Notation

Big-O Complexity Chart



Array Sorting Algorithms

Algorithm	Time Complexity			Space Complexity
	Best	Average	Worst	Worst
Quicksort	Ω(n log(n))	Θ(n log(n))	O(n^2)	O(log(n))
Mergesort	Ω(n log(n))	Θ(n log(n))	O(n log(n))	O(n)
Timsort	Ω(n)	Θ(n log(n))	O(n log(n))	O(n)
Heapsort	Ω(n log(n))	Θ(n log(n))	O(n log(n))	O(1)
Bubble Sort	Ω(n)	Θ(n^2)	O(n^2)	O(1)
Insertion Sort	Ω(n)	Θ(n^2)	O(n^2)	O(1)
Selection Sort	Ω(n^2)	Θ(n^2)	O(n^2)	O(1)
Tree Sort	Ω(n log(n))	Θ(n log(n))	O(n^2)	O(n)
Shell Sort	Ω(n log(n))	Θ(n(log(n))^2)	O(n(log(n))^2)	O(1)
Bucket Sort	Ω(n+k)	Θ(n+k)	O(n^2)	O(n)
Radix Sort	Ω(nk)	Θ(nk)	O(nk)	O(n+k)
Counting Sort	Ω(n+k)	Θ(n+k)	O(n+k)	O(k)
Cubesort	Ω(n)	Θ(n log(n))	O(n log(n))	O(n)

Tree
Definition

- Terms: Node/edge, root node, leaf node, parent and child node, subtree, levels (height/depth).
- Features: No loop, no shared children
- Question: How many edges should there be in a tree with ***n*** nodes?
- Binary tree: no node has more than two children.
- Question: How many nodes can a binary tree of height ***h*** have? → Full binary tree

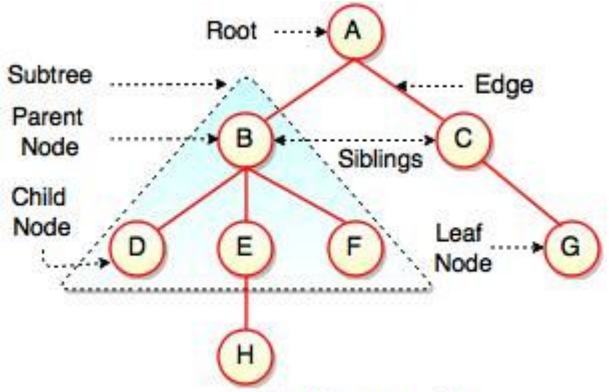
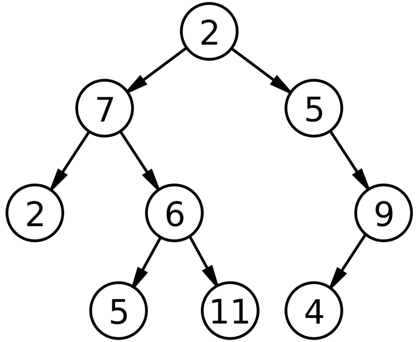


Fig. Structure of Tree



Tree

As a data structure

- Tree is a useful data structure!
- Basic functions: insert, remove, search, **traverse**
- How to traverse a tree?

```
struct Node{
  ItemType val;
  Node* leftChild;
  Node* rightChild;
} // a simple node
```

```
Class Tree{
public:

private:

}
```

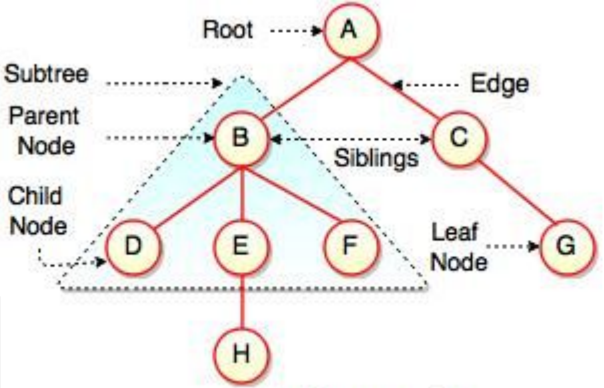
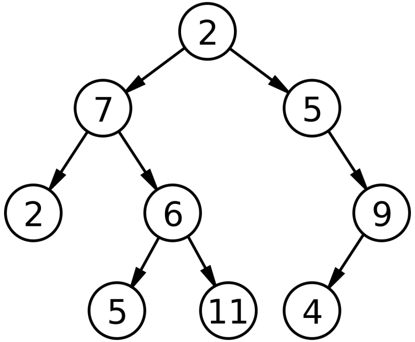


Fig. Structure of Tree



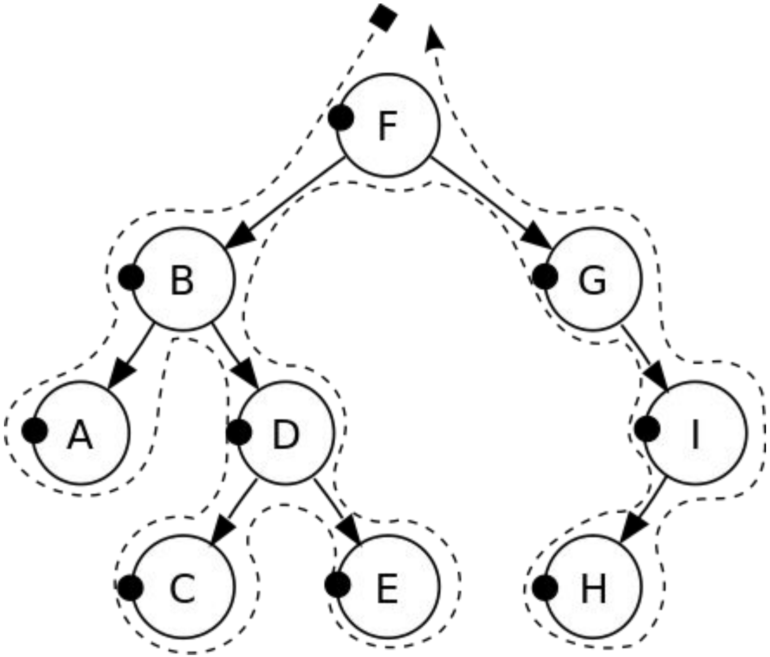
Tree Traversal: Pre-order

Three methods of tree traversal

```
void preorder(const Node* node)
{
  if (node == nullptr) return;
  cout << node->val << “,”;
  preorder(node->left);
  preorder(node->right);
}
```

Pre-order output:

`F, B, A, D, C, E, G, I, H`



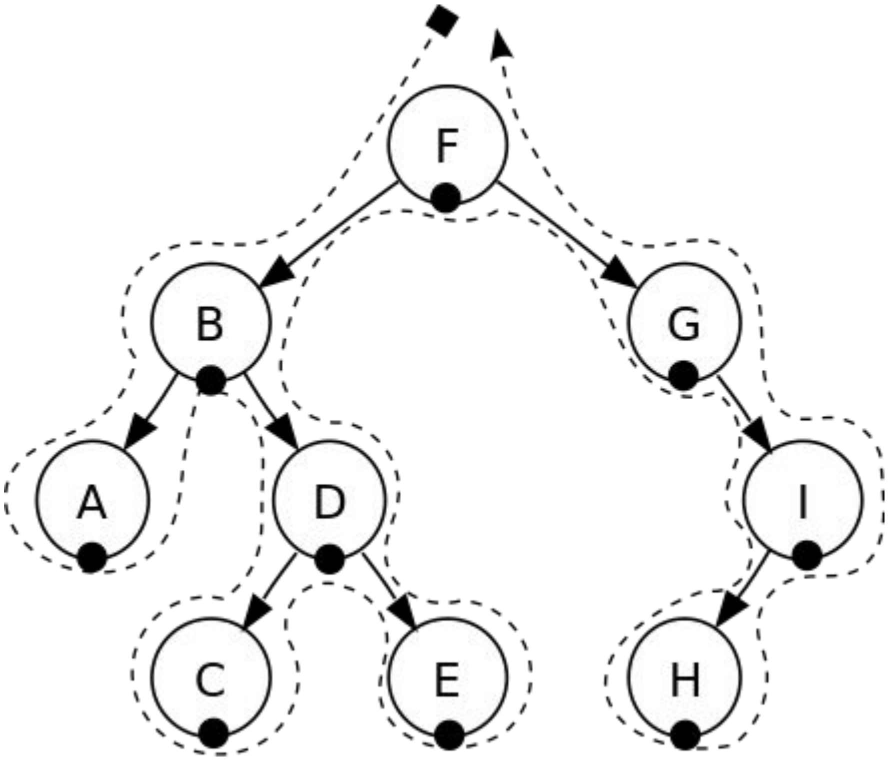
Tree Traversal: In-order

Three methods of tree traversal

```
void inorder(const Node* node)
{
  if (node == nullptr) return;
  inorder(node->left);
  cout << node->val << “,”;
  inorder(node->right);
}
```

In-order output:

`A, B, C, D, E, F, G, H, I`



Tree Traversal: Post-order

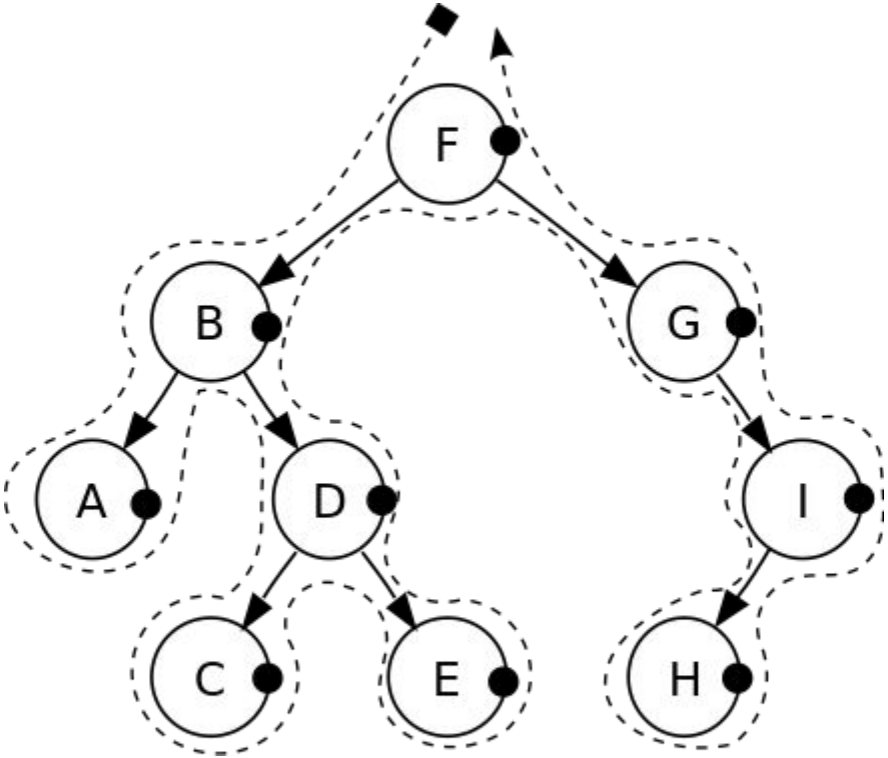
Three methods of tree traversal

UCLA Samueli Computer Science

```
void postorder(const Node* node)
{
  if (node == nullptr) return;
  postorder(node->left);
  postorder(node->right);
  cout << node->val << “,”;
}
```

Post-order output:

`A, C, E, D, B, H, I, G, F`



Tree Traversal: Compare

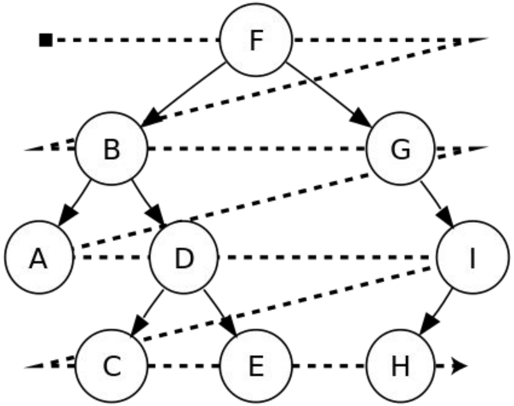
UCLA Samueli Computer Science

```
void preorder(const Node* node)
{
  if (node == nullptr) return;
  cout << node->val << “,”;
  preorder(node->left);
  preorder(node->right);
}
```

```
void inorder(const Node* node)
{
  if (node == nullptr) return;
  inorder(node->left);
  cout << node->val << “,”;
  inorder(node->right);
}
```

```
void postorder(const Node* node)
{
  if (node == nullptr) return;
  postorder(node->left);
  postorder(node->right);
  cout << node->val << “,”;
}
```

```
//Other ways?
Level-order or say
breadth-first search!
```

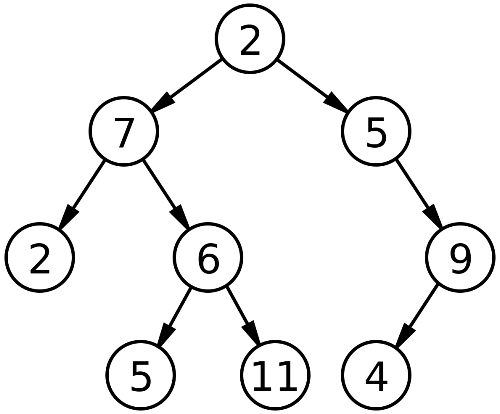


Tree

Recursion in trees

- It is easy and natural to apply recursion on trees!
- Pre-order / in-order / post-order are all recursive methods to traverse a tree.
- Question: How to calculate a height of a given tree?

```
int getTreeHeight(const Node* node)
{
  if (node == nullptr) return 0;
  int leftHeight = getTreeHeight(node->left);
  int rightHeight = getTreeHeight(node->right);
  if (leftHeight > rightHeight)
    return leftHeight + 1;
  else
    return rightHeight + 1;
}
```



Tree

Conclusions and applications

- Non-linear data structures unlike arrays or lists.
- Present hierarchy
- Optimized (like balanced) tree and variants can improve efficiency of search, sort and other typical problems.
- Other tree variants:
  - B tree / B+ tree
  - Red-black tree
  - K-D Tree
  - Suffix Tree

```
file system
-----------
     /    <-- root
  /     \
...     home
       /    \
   ugrad     course
    /       /   |   \
  ...    cs31 cs32 cs35L
```

Next week about tree

- Binary Search Tree
- Heap
- More interesting questions …

6 5 3 1 8 7 2 4

UCLA Samueli Computer Science

Thank you!

Q & A