Binary Trees

University of Massachusetts Amherst
ECE 242 – Data Structures and Algorithms
Lecture 18

Quicksort

- Sorting process with quicksort
  - Partition array based on pivot element
  - Recursively sort left part of partition
  - Recursively sort right part of partition

Quicksort

```
private static void quickSort(int[] values, int left, int right) {
    if (right-left <= 0) {
        return;
    } else {
        int pivotIndex = partition(values, left, right);
        quickSort(values, left, pivotIndex-1);
        quickSort(values, pivotIndex+1, right);
    }
}

public void sort(int[] values) {
    quickSort(values,0,values.length-1);
}
```


Partitioning in quicksort

```
private static int partition(int[] values, int leftBound, int rightBound) {
    int pivot = values[rightBound];
    int left = leftBound-1;
    int right = rightBound; // rightmost is pivot

    while (true) {
        while (values[++left]<pivot) { ; } // search from left
        while (right>leftBound && values[--right]>pivot) { ; } // search from right
        if (left >= right) { // cross-over indicates end of partitioning process
            break;
        } else { // found misplaced items; swap
            swap(values, left, right);
        }
    }
    swap(values, left, rightBound); // move pivot from right position to "middle"
    return left; // return index of pivot
}
```

Quicksort example run

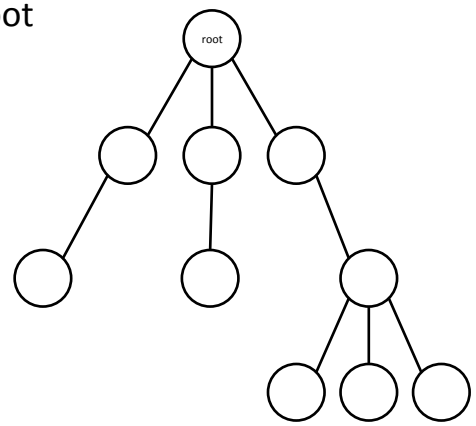
```
call to quickSort: left=0 right=9 array:47 38 86 13 56 52 41 71 33 16
partitioned around 16
call to quickSort: left=0 right=0 array:13 16 86 47 56 52 41 71 33 38
call to quickSort: left=2 right=9 array:13 16 86 47 56 52 41 71 33 38
partitioned around 38
call to quickSort: left=2 right=2 array:13 16 33 38 56 52 41 71 86 47
call to quickSort: left=4 right=9 array:13 16 33 38 56 52 41 71 86 47
partitioned around 47
call to quickSort: left=4 right=4 array:13 16 33 38 41 47 56 71 86 52
call to quickSort: left=6 right=9 array:13 16 33 38 41 47 56 71 86 52
partitioned around 52
call to quickSort: left=6 right=5 array:13 16 33 38 41 47 52 71 86 56
call to quickSort: left=7 right=9 array:13 16 33 38 41 47 52 71 86 56
partitioned around 56
call to quickSort: left=7 right=6 array:13 16 33 38 41 47 52 56 86 71
call to quickSort: left=8 right=9 array:13 16 33 38 41 47 52 56 86 71
partitioned around 71
call to quickSort: left=8 right=7 array:13 16 33 38 41 47 52 56 71 86
call to quickSort: left=9 right=9 array:13 16 33 38 41 47 52 56 71 86
```


Tree data structures

- New data structure
  - "2-dimensional" structure
  - Easy access (similar to binary search in array)
  - Easy insert and removal (similar to linked list)
- Trees
  - Consist of nodes and links
  - Are graphs without loops
  - Typically have a root node

Structure of trees

- Nodes
  - Special node at top: root
- Links
  - Connect nodes
  - Zero or more nodes connected to a node
- Nodes can store information



Tree terminology

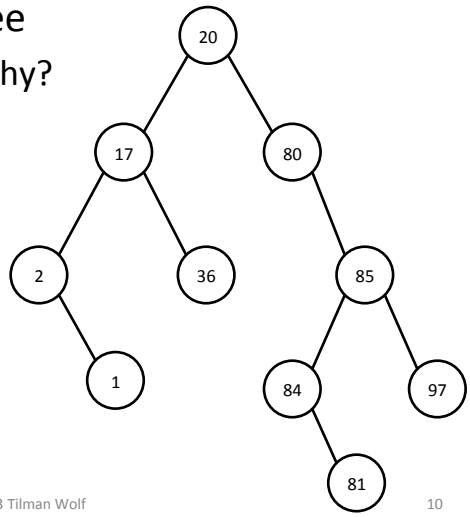
- Root: top node
- Parent: node "above"
  - Every node (except root) has exactly one parent
- Child: node "below"
  - Nodes may have zero or more children
  - Binary trees have at most two children
- Leaf: node without children
- Subtree: tree below a given node
  - That node becomes root of the subtree
- Level: distance from root

Binary trees

- Binary tree
  - Every node has at most two children
    - Left child
    - Right child
- Binary search tree
  - Nodes are arranged in a particular fashion
    - Left subtree has values smaller than node
    - Right subtree has values larger than node

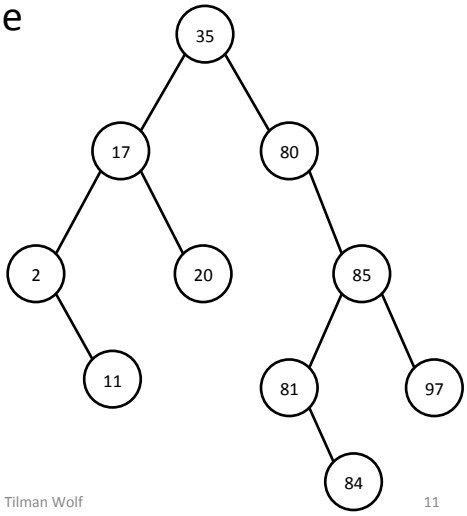
Binary tree example

- Example of binary tree
  - Not a search tree – why?



Binary tree example

- Example of binary tree
  - Search tree – why?



Binary tree implementation

- Node data structure

```
public class Node {
    int value; // data stored at node
    Node left; // pointer to left child
    Node right; // pointer to right child

    public Node(int i) {
        value = i;
        left = null;
        right = null;
    }
}
```


Binary tree implementation

- Tree data structure

```
public class Tree {
    Node root;

    public Tree() {
        root = null;
    }
}
```


Finding a node

- How to find a node in binary search tree?

```
public Node find(int i) {









}
```

Finding a node

- How to find a node in binary search tree?

```
public Node find(int i) {
    Node current = root;
    while (current!=null && current.value!=i) {
        if (i<current.value) {
            current = current.left;
        } else {
            current = current.right;
        }
    }
    return current;
}
```


Next Steps

- Class on Monday
  - Taught by Prof. Tessier