APS 105 Lecture 36 Notes

Yesterday: Quicksort

Today: Searching algorithms and binary search trees
↳ efficient clever data structure for storing sorted lists.

Searching Algorithms

Search for an element in an array

```
int sequentialSearch ( int list[], int length, int data){
        int index = -1;
        for (int i=0; i< length && index==-1; i++)
            if (list[i] == data)
                index = i;
        return i;
}
```

At most we do length comparisons

Best case: 1 comp.

Average: n/2 comp.

Is there a better way?

If my array was sorted, e.g. 1 3 5 10 13 (indices 0 1 2 3 4)
and I want to look for 10?

1) look at $arr[n/2] = arr[5/2] = arr[2] = 5$

if $10 > 5 \rightarrow$ look right subarray

$10 < 5 \rightarrow$ look left subarray

2) Repeat (1)

$$arr\left[\frac{n/2 + n}{2}\right] = arr[3] = 10$$

if $10 == 10 \rightarrow$ found

We eliminate half of the array everytime.

We do $\log_2(\text{length})$ comparisons.

We call the method "binary search"

Iterative Solution

```
int binarySearch (int list[], int length, int data){
    int low = 0, high = length - 1;
    while ( low <= high ) {
        int middle = (low + high) / 2;
        if ( list[middle] == data)
            return middle;
        else if ( list[middle] > data)
            high = middle - 1;
        else
            low = middle + 1;
    }
```

0 1 3 5 8 13 look for 1

① low = 0 middle = 2 high = 5

② low = 0 middle = 0 high = 1

③ low = 1 middle = 1 high = 1 found, so we need to enter loop when low == high

Rewrsive


```
int binarySearchHelper(int list[], int length,
                       int data, int low, int high){

    if (low > high)
        return -1;

    int middle = (low + high)/2;
    if (list[middle] == data)
        return middle;
    if (list[middle] > data)        go left
        return binarySearchHelper(list, length, data,
                                  low, middle-1);
    else                            go right
        return binarySearchHelper(list, length, data,
                                  middle+1, high);
}

int binarySearch(int list[], int length, int data){

    return binarySearchHelper(list, length, data, 0,
                              length-1);
```

Recall different data structures:

1. Arrays
2. Linked lists —

```
typedef struct node {
    int data;
    struct node * next;
} Node;
```

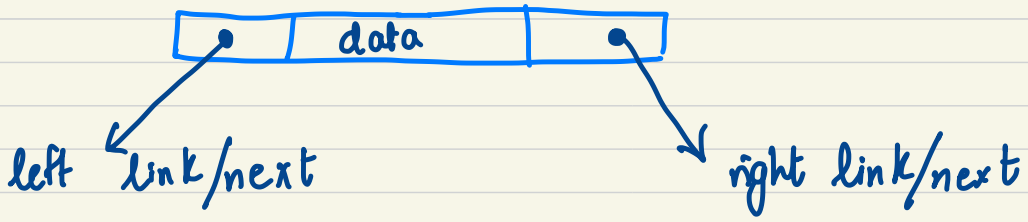
Problem with arrays and linked list:

To look for an item, it takes on average $n/2$ comparisons; n: size of data structure

New data structure:

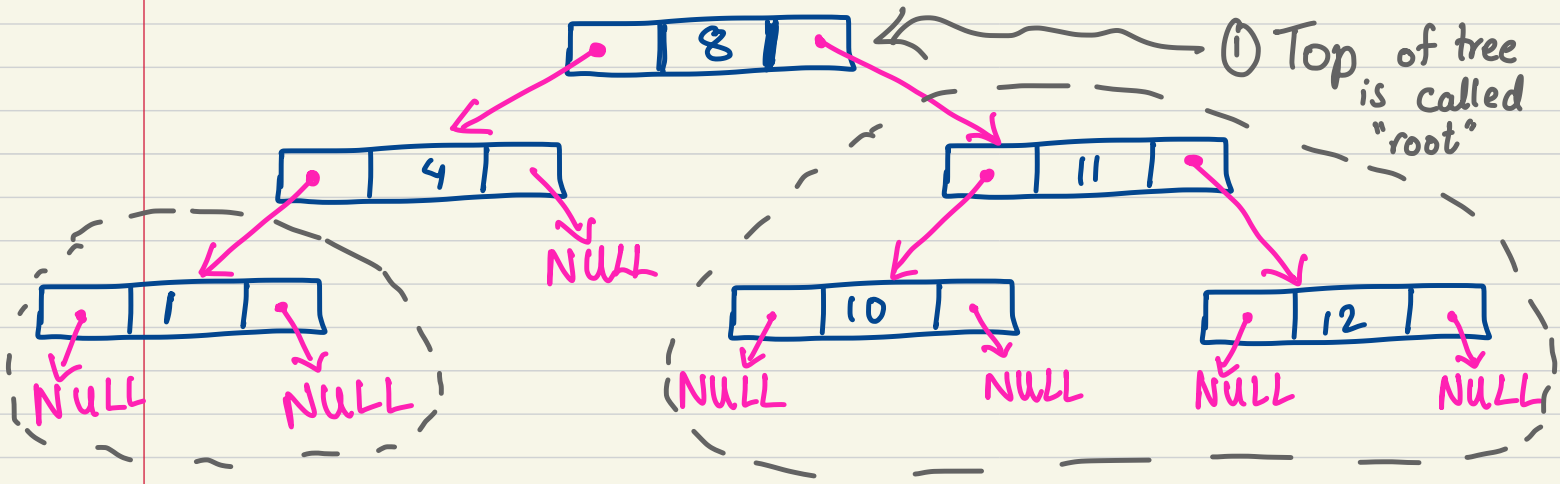
Binary Trees - similar to linked list, instead it points to 2 items.

Looks like an upside-down tree




```
typedef struct node{
    int data;
    struct node * left, *right;
} Node;
```

Here is a binary tree storing numbers 1, 4, 8, 10, 11, 12 in order



A binary tree: has 1 parent only for each node
each node has at most 2 children
hence binary

A binary search tree: is an ordered binary tree

How is the tree ordered?

all values of nodes on left subtree are smaller than parent node.
all values of nodes on right subtree are greater than parent node.

Create a Node

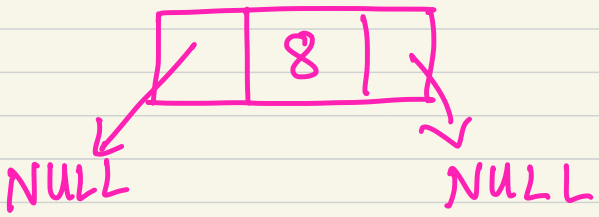
```
typedef struct bstree{
    Node *root;
}BSTree;

typedef struct node{
    int data;
    struct node* left, right;
}Node;

int main(){

    BSTree tree;
    tree->root = (Node*) malloc (sizeof(Node) * 1);
    tree->root->data = 8;
    tree->root->left = NULL;
    tree->root->right = NULL;
```

All of this can be in createNode funct.




```
}

Node* createNode(int value){

    Node* newNode = (Node*) malloc(sizeof(Node));
    if(p != NULL){
        p->data = value;
        p->left = p->right = NULL;
    return p;
}
```