Binary Search and Proofs of Correctness

CS/MCS 401: Computer Algorithms I
Lecture 7
Professor: Jonathan Liu

Introduce yourself!
Icebreaker: What's your favorite restaurant of all time?

Announcements

Homework 1 has been graded (if you submitted on time). Solutions can be found on Canvas. Resubmissions will open tonight.

I was particularly harsh when grading Problem 4. I want you to know two things:

- I feel comfortable being harsh because there are resubmissions available.
- On exams, you would lose points for the same reasons, but I'd be more generous with partial credit.

Homework 2 will be released by the end of the day tomorrow.

Lesson Goals

Today we're going to discuss a whole new type of divide and conquer algorithm, and use it as an example to learn how to write proofs of correctness.

Then, we're going to do a lot of practice.

Binary Search

Binary Search

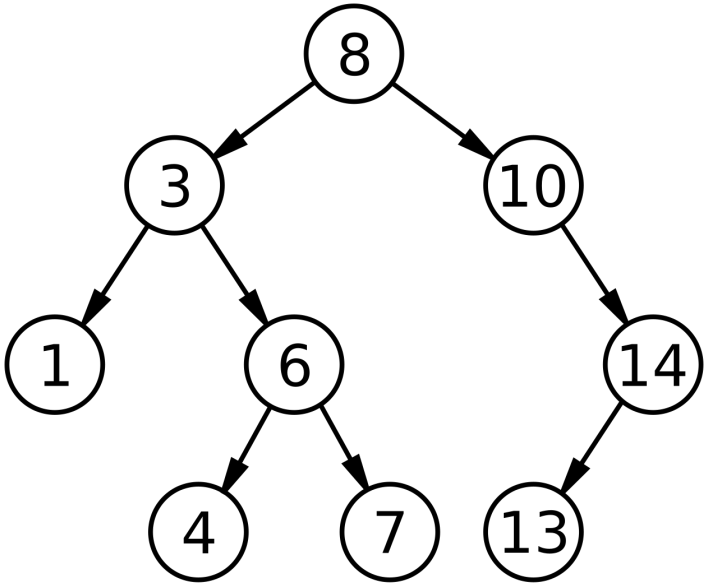
Binary Search is a style of Divide and Conquer algorithm that follows two characteristics:

1. You are *searching* for a specific input within a data structure.
2. That data structure has some underlying order that helps you navigate the structure faster than simply iterating through every item in the structure.

Recap: Binary Search Trees

Goal: Return node with value x.

```
def FIND(x, T):
    if T.value = x:
        return T
    if T.value < x:
        return FIND(T.left)
    else:
        return FIND(T.right)
```

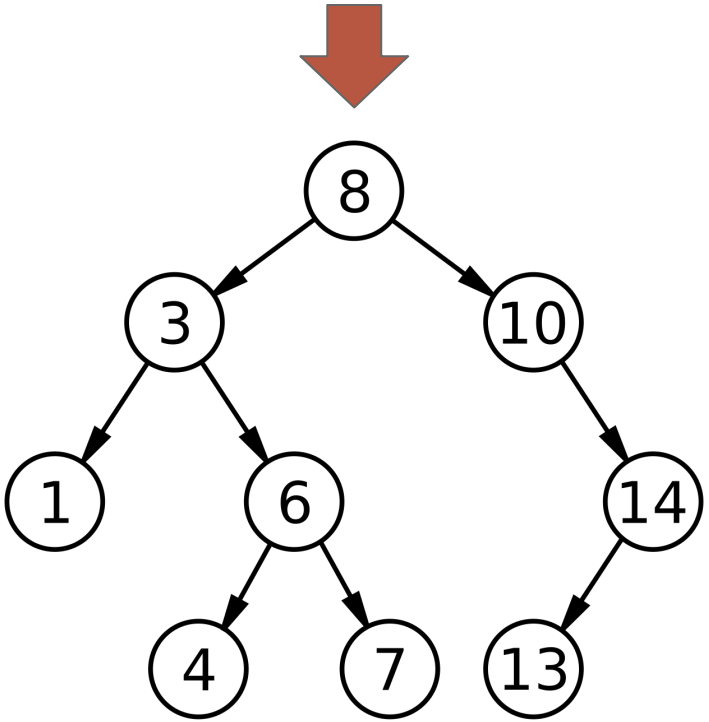


Recap: Binary Search Trees

Goal: Return node with value x.

```
def FIND(x, T):
    if T.value = x:
        return T
    if T.value > x:
        return FIND(T.left)
    else:
        return FIND(T.right)
```

Say I Call: FIND(6, T)

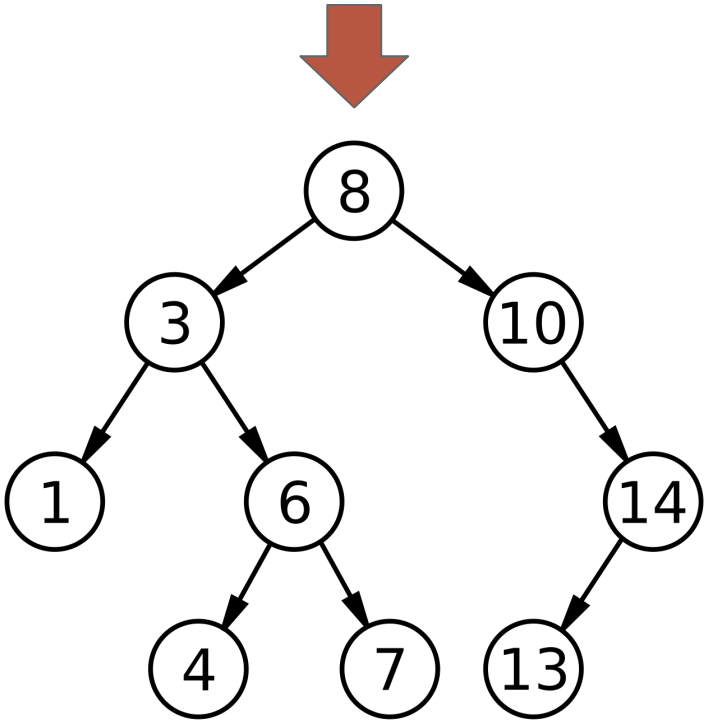


Recap: Binary Search Trees

Goal: Return node with value x.

```
def FIND(x, T):
    if T.value = x:
        return T
    if T.value > x:
        return FIND(T.left)
    else:
        return FIND(T.right)
```

Say I Call: FIND(6, T)

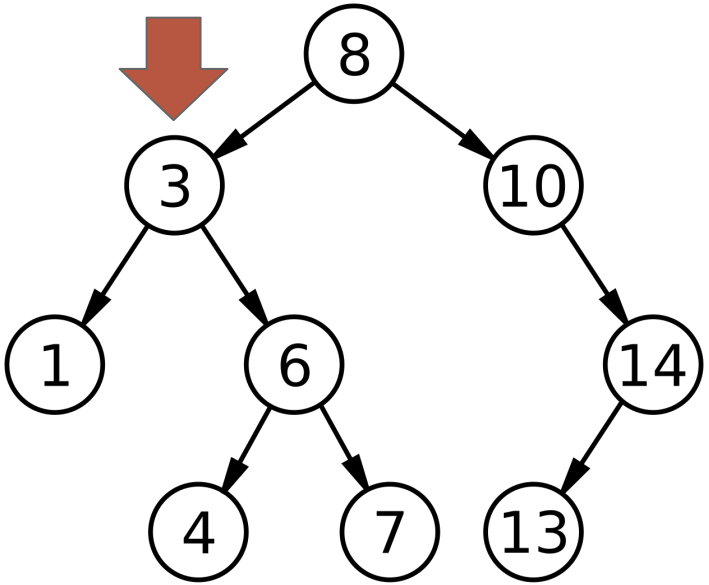


Recap: Binary Search Trees

Goal: Return node with value x.

```
def FIND(x, T):
    if T.value = x:
        return T
    if T.value > x:
        return FIND(T.left)
    else:
        return FIND(T.right)
```

Say I Call: FIND(6, T)

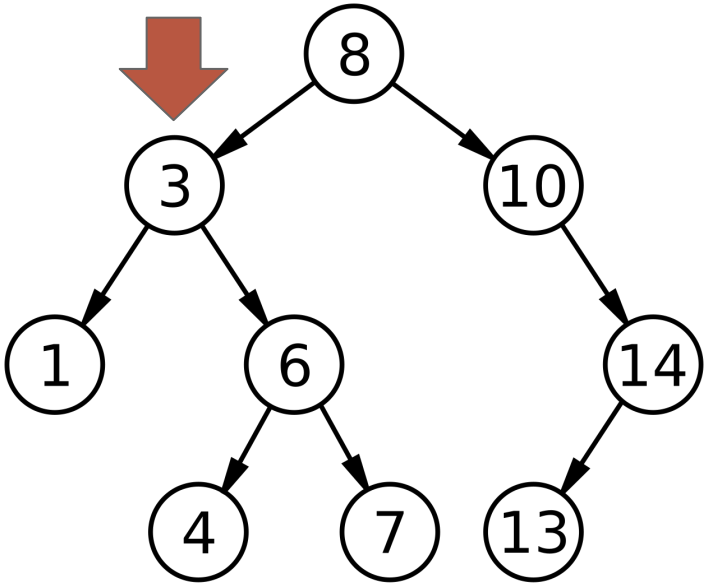


Recap: Binary Search Trees

Goal: Return node with value x.

```
def FIND(x, T):
    if T.value = x:
        return T
    if T.value > x:
        return FIND(T.left)
    else:
        return FIND(T.right)
```

Say I Call: FIND(6, T)

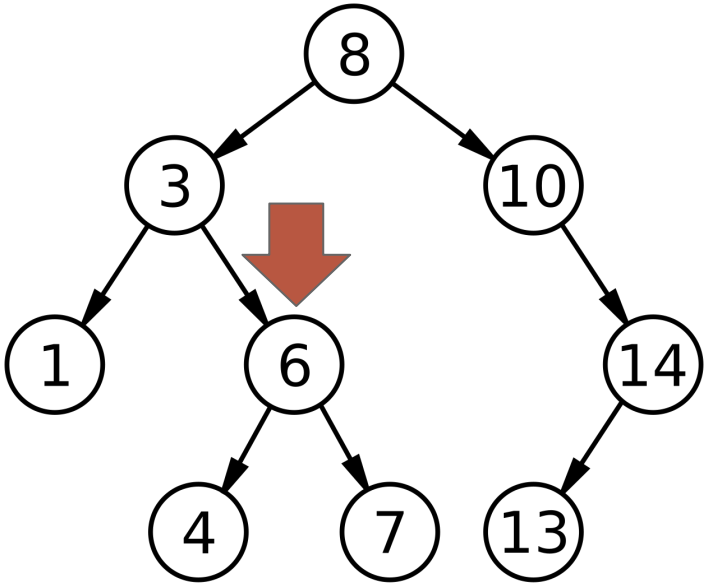


Recap: Binary Search Trees

Goal: Return node with value x.

```
def FIND(x, T):
    if T.value = x:
        return T
    if T.value > x:
        return FIND(T.left)
    else:
        return FIND(T.right)
```

Say I Call: FIND(6, T)

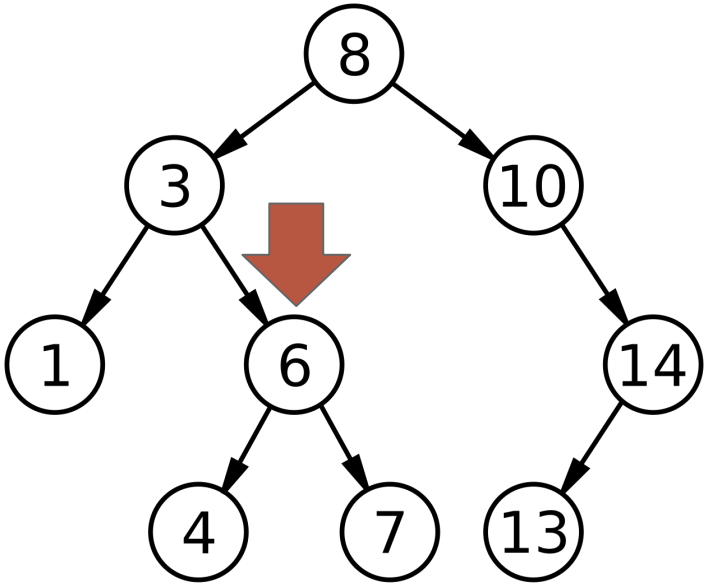


Recap: Binary Search Trees

Goal: Return node with value x.

```
def FIND(x, T):
    if T.value = x:
        return T
    if T.value > x:
        return FIND(T.left)
    else:
        return FIND(T.right)
```

Say I Call: FIND(6, T)



Recap: Binary Search on a Sorted Array

Goal: Find the index of *value* in sorted array.

2	8	12	29	59	74	79	91

```
def FIND(value, array):
    midpoint = len(array)//2
    if array[midpoint] = value:
        return midpoint
    else if array[midpoint] > value:
        return FIND(value, array[0:midpoint])
    else:
        return FIND(value, array[midpoint:n]) + (midpoint)
```

Recap: Binary Search on a Sorted Array

Goal: Find the index of *value* in sorted array.

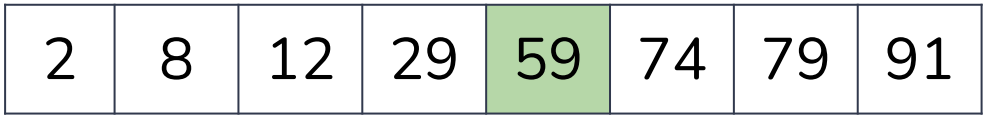
Say I Call: FIND(29, A)


```
def FIND(value, array):
    midpoint = len(array)//2
    if array[midpoint] = value:
        return midpoint
    else if array[midpoint] > value:
        return FIND(value, array[0:midpoint])
    else:
        return FIND(value, array[midpoint:n]) + (midpoint)
```

Recap: Binary Search on a Sorted Array

Goal: Find the index of *value* in sorted array.

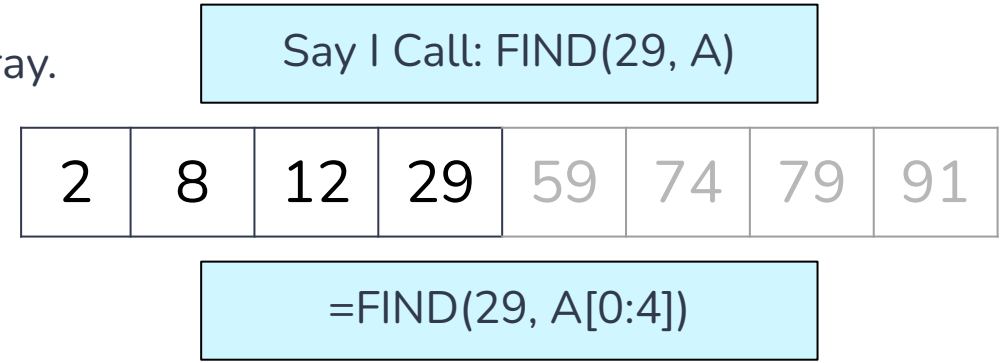
Say I Call: FIND(29, A)




```
def FIND(value, array):
    midpoint = len(array)//2
    if array[midpoint] = value:
        return midpoint
    else if array[midpoint] > value:
        return FIND(value, array[0:midpoint])
    else:
        return FIND(value, array[midpoint:n]) + (midpoint)
```

Recap: Binary Search on a Sorted Array

Goal: Find the index of *value* in sorted array.




```
def FIND(value, array):
    midpoint = len(array)//2
    if array[midpoint] = value:
        return midpoint
    else if array[midpoint] > value:
        return FIND(value, array[0:midpoint])
    else:
        return FIND(value, array[midpoint:n]) + (midpoint)
```

Recap: Binary Search on a Sorted Array

Goal: Find the index of *value* in sorted array.

Say I Call: FIND(29, A)

=FIND(29, A[0:4])

```
def FIND(value, array):
    midpoint = len(array)//2
    if array[midpoint] = value:
        return midpoint
    else if array[midpoint] > value:
        return FIND(value, array[0:midpoint])
    else:
        return FIND(value, array[midpoint:n]) + (midpoint)
```

Recap: Binary Search on a Sorted Array

Goal: Find the index of *value* in sorted array.

Say I Call: FIND(29, A)

=FIND(29, A[0:4])

=FIND(29, A[2:4]) + 2

```
def FIND(value, array):
    midpoint = len(array)//2
    if array[midpoint] = value:
        return midpoint
    else if array[midpoint] > value:
        return FIND(value, array[0:midpoint])
    else:
        return FIND(value, array[midpoint:n]) + (midpoint)
```

Recap: Binary Search on a Sorted Array

Goal: Find the index of *value* in sorted array.

Say I Call: FIND(29, A)

=FIND(29, A[0:4])

=FIND(29, A[2:4]) + 2

```
def FIND(value, array):
    midpoint = len(array)//2
    if array[midpoint] = value:
        return midpoint
    else if array[midpoint] > value:
        return FIND(value, array[0:midpoint])
    else:
        return FIND(value, array[midpoint:n]) + (midpoint)
```

Recap: Binary Search on a Sorted Array

Goal: Find the index of *value* in sorted array.

```
def FIND(value, array):
    midpoint = len(array)//2
    if array[midpoint] = value:
        return midpoint
    else if array[midpoint] > value:
        return FIND(value, array[0:midpoint])
    else:
        return FIND(value, array[midpoint:n]) + (midpoint)
```

Say I Call: FIND(29, A)

=FIND(29, A[0:4])

=FIND(29, A[2:4]) + 2

= 1 + 2 = **3**

Recap: Binary Search on a Sorted Array

Goal: Find the index of *value* in sorted array.

```
def FIND(value, array):
    midpoint = len(array)//2
    if array[midpoint] = value:
        return midpoint
    else if array[midpoint] > value:
        return FIND(value, array[0:midpoint])
    else:
        return FIND(value, array[midpoint:n]) + (midpoint)
```

In general, the divide and conquer paradigm involves algorithms that look as follows:

- **Base Case:** Solve the trivial input.
- **Divide:** Split the input into smaller chunks.
- **Conquer:** Recursively solve.
- **Combine:** Do something efficient to create the final answer.

Recap: Binary Search on a Sorted Array

Goal: Find the index of *value* in sorted array.

```
def FIND(value, array):
    midpoint = len(array)//2
    if array[midpoint] = value:
        return midpoint
    else if array[midpoint] > value:
        return FIND(value, array[0:midpoint])
    else:
        return FIND(value, array[midpoint:n]) + (midpoint)
```

In general, the divide and conquer paradigm involves algorithms that look as follows:

- **Base Case:** Solve the trivial input.
- **Divide:** Split the input into smaller chunks.
- **Conquer:** Recursively solve.
- **Combine:** Do something efficient to create the final answer.

Recap: Binary Search on a Sorted Array

Goal: Find the index of *value* in sorted array.

```
def FIND(value, array):
    midpoint = len(array)//2
    if array[midpoint] = value:
        return midpoint
    else if array[midpoint] > value:
        return FIND(value, array[0:midpoint])
    else:
        return FIND(value, array[midpoint:n]) + (midpoint)
```

In general, the divide and conquer paradigm involves algorithms that look as follows:

- **Base Case:** Solve the trivial input.
- **Divide:** Split the input into smaller chunks.
- **Conquer:** Recursively solve.
- **Combine:** Do something efficient to create the final answer.

Recap: Binary Search on a Sorted Array

Goal: Find the index of *value* in sorted array.

```
def FIND(value, array):
    midpoint = len(array)//2
    if array[midpoint] = value:
        return midpoint
    else if array[midpoint] > value:
        return FIND(value, array[0:midpoint])
    else:
        return FIND(value, array[midpoint:n]) + (midpoint)
```

In general, the divide and conquer paradigm involves algorithms that look as follows:

- **Base Case:** Solve the trivial input.
- **Divide:** Split the input into smaller chunks.
- **Conquer:** Recursively solve.
- **Combine:** Do something efficient to create the final answer.

Proofs of Correctness for Divide and Conquer Algorithms

Proof of Correctness on Binary Search

Proof of Correctness (Induction)
Claim: If *v* is in *A*, FIND(v, A) returns its index.

Base Case: n=1. Then *A*[0]=*v*, so the algorithm should return 0, and it does.
IH: For any array of length ≤ n, if *v* is in *A* then FIND(v, A) returns the index of value in the array.

IS: We must show that FIND(v, A) returns the right index when A has length n+1. There are three possibilities: A[midpoint] = v, A[midpoint] > v, or A[midpoint] < v.

- **If A[midpoint] = v,** the algorithm returns *midpoint*, which is correct.
- **If A[midpoint] > v,** the algorithm returns FIND(v, A[0:midpoint]). Because A is sorted, we know v is in A[0:midpoint]. Furthermore, A[0:midpoint] has length less than n. Thus, by the IH, FIND(v, A[0:midpoint]) returns the correct index.
- **If A[midpoint] < v,** the algorithm returns FIND(v, A[midpoint:n]) + midpoint. Because A is sorted, we know v is in A[midpoint:n]. Furthermore, A[midpoint:n] has length less than n. Thus, by the IH, FIND(v, A[midpoint:n]) returns the index of v in A[midpoint:n]. Looking back at the whole array, the index of that point is FIND(v, A[midpoint:n]) + midpoint, so it is correct.

Rest of Today

Work, either alone or with other people, on the packet.

I'd recommend doing all of the non-bonus problems first before trying the bonus problems.

For your attendance point today: you must ask me a question about something you're unsure about in the packet. That can be asking for a hint, or asking to check your work, or asking for clarification, or anything else. You can come up to me, or you can raise your hand and I'll come to you.