Sorting Algorithms

N. Lutz
MIT Media Lab
Object Based Media

Algorithms Are For Everyone

Who am I?

Grad student here at OBM.

Have TAed and taken various courses in Computer Science, specifically in computational for not computer science and algorithms

MIT gave me a B.Sc in Computer Science.

Recap

An **algorithm** is a process of instructions for a computer and can be written in **pseudocode.**

They run in time and space. We denote this with big O notation.

O(1)
O(lgn)
O(n)
O(nlgn)
O(n^2)
O(2^n)
O(n!)

Sorting Algorithms

Used to rearrange elements in a data structure.

Most sorting algorithms rely on a **comparator** between elements. This can be numerical, logical, or alphabetical.

Naive Sorting Algorithm

6 4 2 8 3 1 7

Maybe you start at the first number and for each number you move it to the left or right of the pivot if it’s smaller or larger. That sounds like it would be O(n), right?

Naive Sorting Algorithm

6 4 2 8 3 1 7

4 **6** 2 8 3 1 7

4 2 **6** 8 3 1 7

4 2 **6** 8 3 1 7

4 2 3 **6** 8 1 7

4 2 3 1 **6** 7 8

4 2 3 1 **6** 8 7 <— not sorted in O(n) :(

(it’s a bit more complicated than first meets the eye)

Pivots and Comparators

Last example, 6 was our **pivot.** Which is a starting point of a **comparator.**

But rarely can we sort with O(n) comparisons on a stagnant pivot.

We say rarely because we can get lucky. But again, not often.

Considerations of Sorting Algorithms

Speed - How fast it is

Memory - How much space it takes up

Stability - sorting repeated elements in the same order as original structure

Comparator - Most comparisons are constant time but some aren’t

A Lower Bound

Blah blah blah

Common Sorting Algorithms

Insertion
Bubble
Binary Tree Sort
Merge Sort

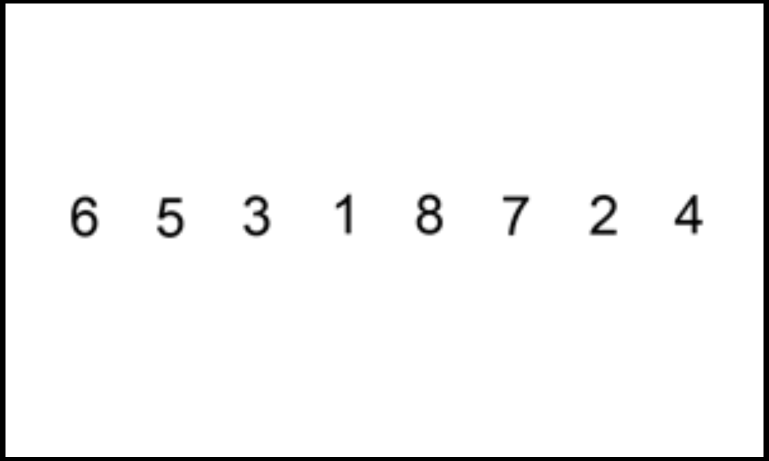
Insertion Sort

```
i ← 1
while i < length(A)
    j ← i
    while j > 0 and A[j-1] > A[j]
        swap A[j] and A[j-1]
        j ← j - 1
    end while
    i ← i + 1
end while
```

Insertion Sort

Iterates through the list comparing one by one and swapping if an element is less than the current moving pivot.

O(n^2), best case O(n)



Bubble Sort

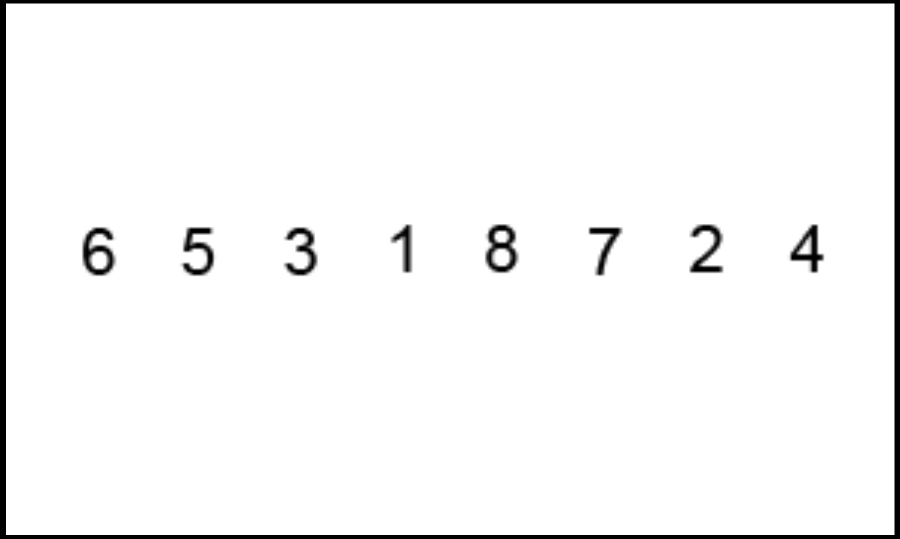
```
procedure bubbleSort( A : list of sortable items )
    n = length(A)
    repeat
        swapped = false
        for i = 1 to n-1 inclusive do
            if A[i-1] > A[i] then
                swap(A[i-1], A[i])
                swapped = true
            end if
        end for
        n = n - 1
    until not swapped
end procedure
```

Bubble Sort

Swaps by each pair.

Then goes through the new pairs until the list is sorted.

O(n^2), best case O(n)

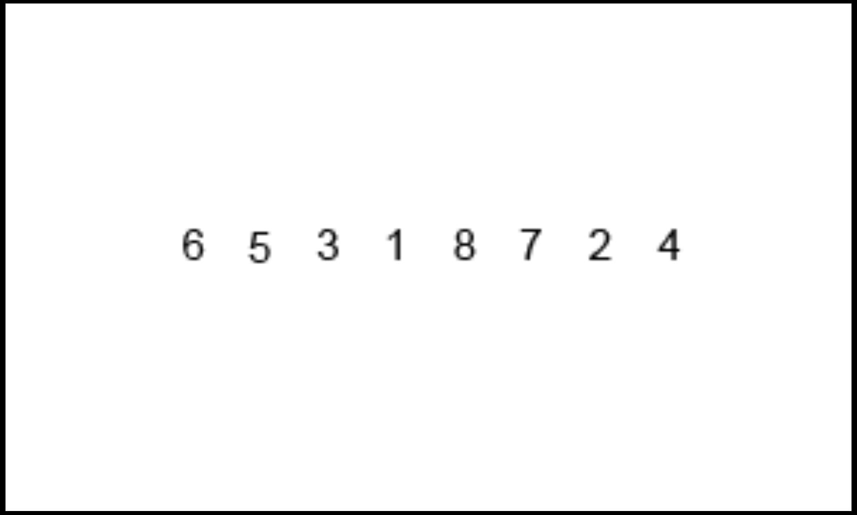


Merge Sort

Divide and conquer approach to sort.

Divides array into sub arrays and sorts from smallest size to largest.

O(nlgn) worst case, but in many cases can be O(n).



Merge Sort

```
MergeSort(arr[], l,  r)
If r > l
    1. Find the middle point to divide the array into two halves:
            middle m = (l+r)/2
    2. Call mergeSort for first half:
            Call mergeSort(arr, l, m)
    3. Call mergeSort for second half:
            Call mergeSort(arr, m+1, r)
    4. Merge the two halves sorted in step 2 and 3:
            Call merge(arr, l, m, r)
```

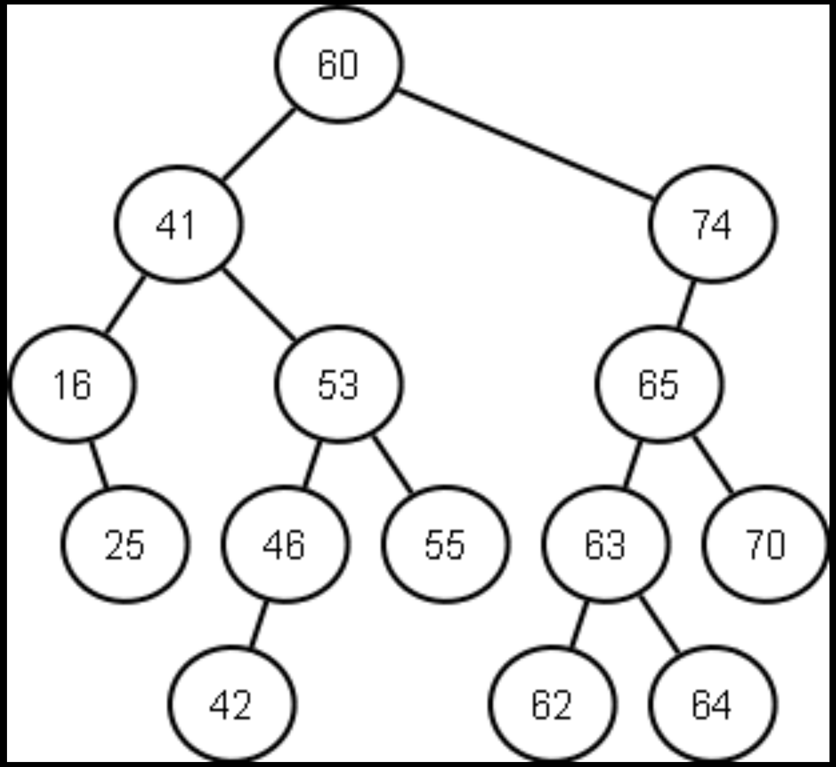
Binary Tree Search

This exploits a **data structure** for its run time. This is a common method in algorithms.

The algorithm is O(nlgn) and is fairly efficient and organized.

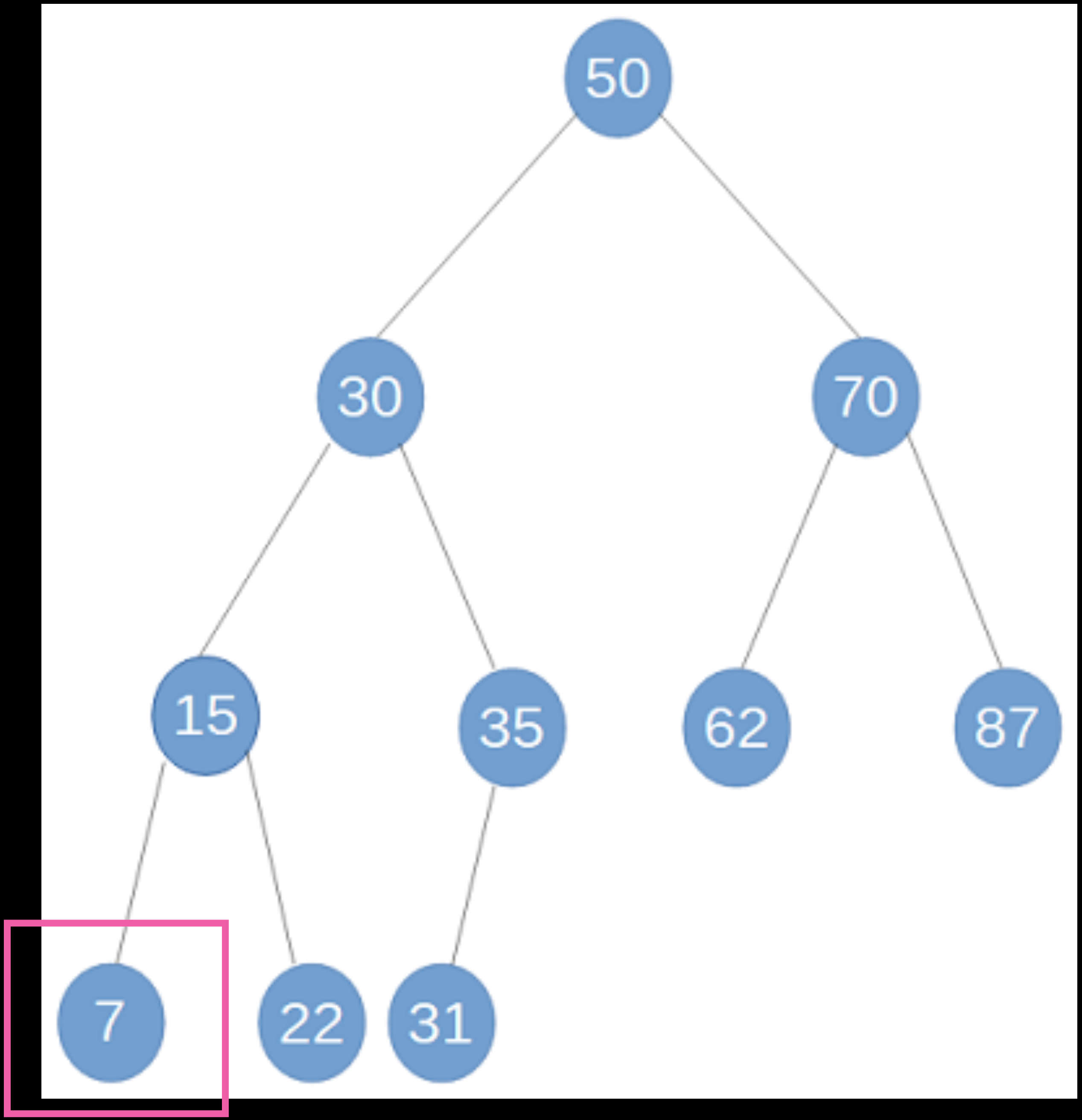
Binary Search Tree

- A **BST** is a data structure where each node has a parent and often a child.
- The left node MUST be smaller than the right AND parent nodes.
- The right node MUST be bigger than the left AND parent node.
- A **balanced BST** happens if all nodes have 2 children
- **Depth** is important to runtime.



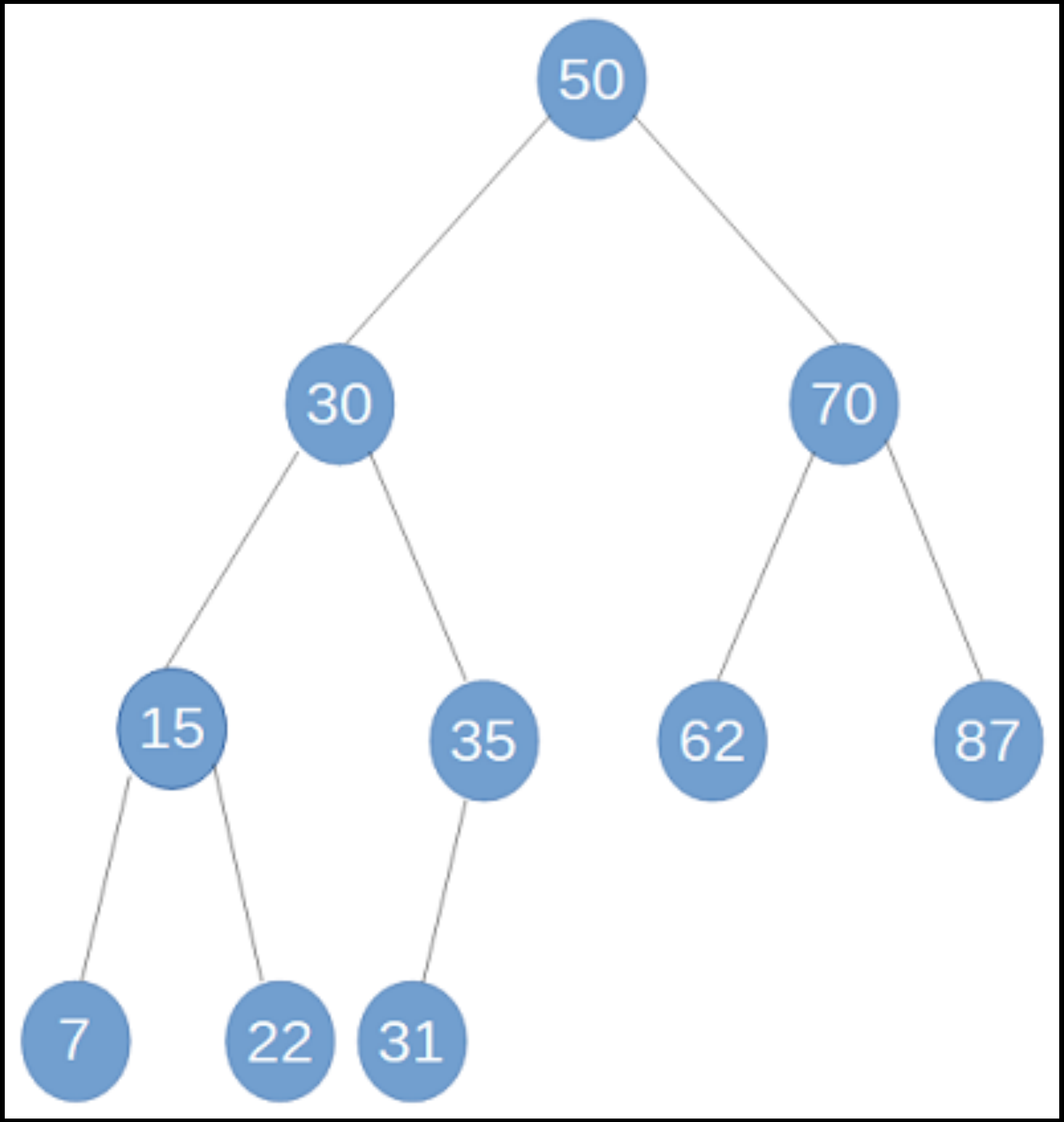
Binary Tree Sort

The sort simply goes through the tree and builds a sorted array. Think about this example and start at the bottom left node.



Walkthrough

We walk from the bottom left node, go up, go right, then go to the next layer.



Other Considerations

- https://en.wikipedia.org/wiki/Sorting_algorithm

In place merge sort

- A chosen **implementation** of merge sort. This isn't a separate algorithm itself, but a programmer uses pointers to rearrange their structure instead of an auxiliary array.
- Same time complexity, but saves space in memory.
- Franceschini, G. (June 2007). "Sorting Stably, in Place, with O(n log n) Comparisons and O(n) Moves". *Theory of Computing Systems*. **40** (4): 327–353. doi:10.1007/s00224-006-1311-1.

So why is it so important?

- Easier and faster to locate data
- Can help with other algorithmic processes like search or matching or countless others

Good resources

- Time complexity lessons
- 6.006 OCW
- Geeks for Geeks is great
- Next week I talk about trees
- https://github.com/ninalutz/AlgorithmsWorkshops