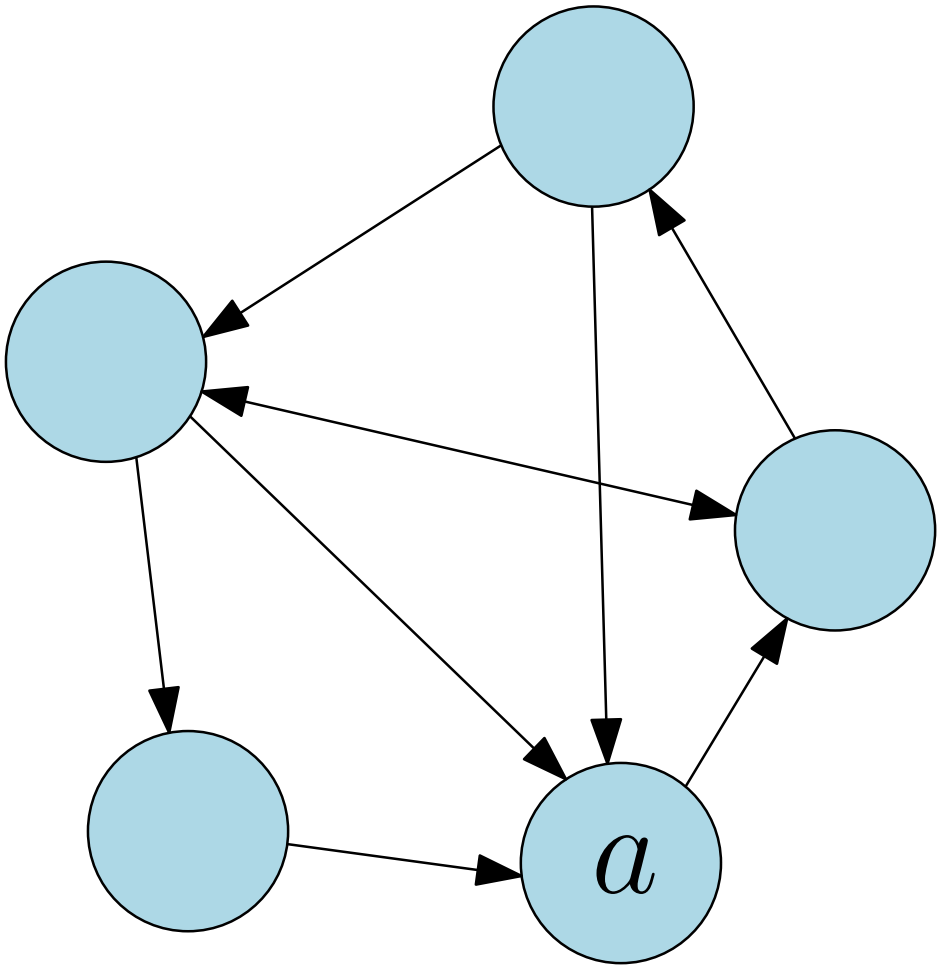
Complexity and Data Structures

Daniel Archambault

Previously in CS-115



Graphs connect the world!

Previously in CS-115

- What differentiates a graph from a tree?

Previously in CS-115

- What differentiates a graph from a tree?
- What is a directed graph?

Previously in CS-115

- What differentiates a graph from a tree?
- What is a directed graph?
- What is an undirected graph?

Previously in CS-115

- What differentiates a graph from a tree?
- What is a directed graph?
- What is an undirected graph?
- What is degree?

Previously in CS-115

- What differentiates a graph from a tree?
- What is a directed graph?
- What is an undirected graph?
- What is degree?
- What are two graph data structures that implement the ADT?

Previously in CS-115

- What differentiates a graph from a tree?
- What is a directed graph?
- What is an undirected graph?
- What is degree?
- What are two graph data structures that implement the ADT?
- What are advantages/disadvantages?

Previously in CS-115

- Complexity of algorithms? What about data structures?

Complexity and Data Structures

Complexity, Data Structures, and Sorting

- Complexity tries to quantify how fast or big something is
  - given an input size of n how long could this take?
  - given an input size of n how much memory/space do we need?
- We use n as the input size because it's variable
- With big n, sometimes a lot of time and space is required!
- In this lecture, we look at worst case (how bad it can get) and average case complexity.

Rules of Complexity

- In complexity, we worry only about the largest exponent
- We ignore all constant values and lower exponents
  - $2n^3 + 32n^2 + 4$ is $O(n^3)$
- Why can we do this? As n becomes really large, other terms don't matter so much.

Space Complexity

- You have already done time complexity? Think sorting...
- Well, you can use the same tool to talk about how much space something takes in memory.
  - Given an input size of n, it's possible for it to take $O(n^2)$ space
- Important for data structures
  - time complexity for the operations on the data
  - space complexity for the data itself

Warning!

- In this lecture we *estimate* complexity
- We are providing estimates not proofs
- Complexity theory is hard and involves proofs
  - particular algorithm has performance...
  - any algorithm for a problem will have performance...
- Our arguments are not proofs, but estimates
- In second year, you will do rigourous proofs.

Array Complexity

```
int[] a = new int [n];
```

- Insertion complexity?

Array Complexity

```
int[] a = new int [n];
```

- Insertion complexity? $O(n)$
- Access complexity?

Array Complexity

```
int[] a = new int [n];
```

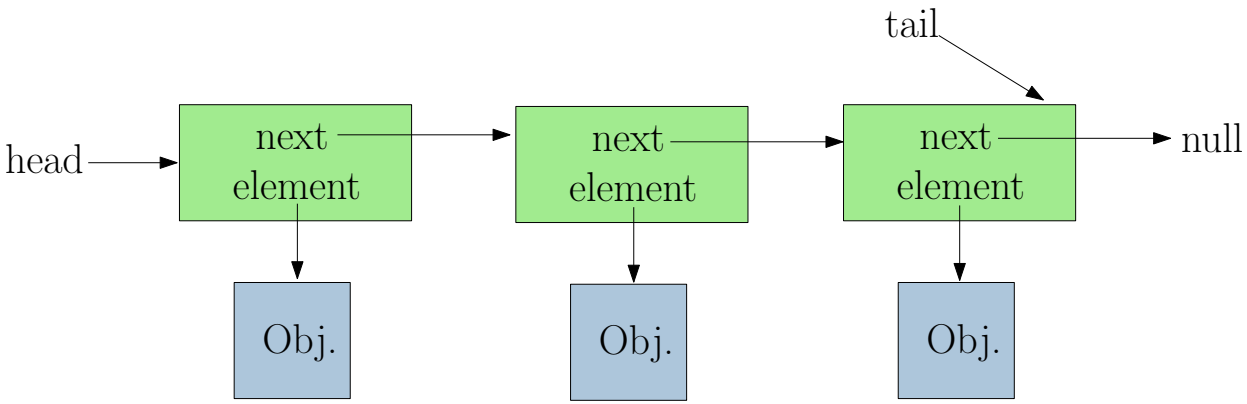
- Insertion complexity? $O(n)$
- Access complexity? $O(1)$
- Space complexity?

Array Complexity

```
int[] a = new int [n];
```

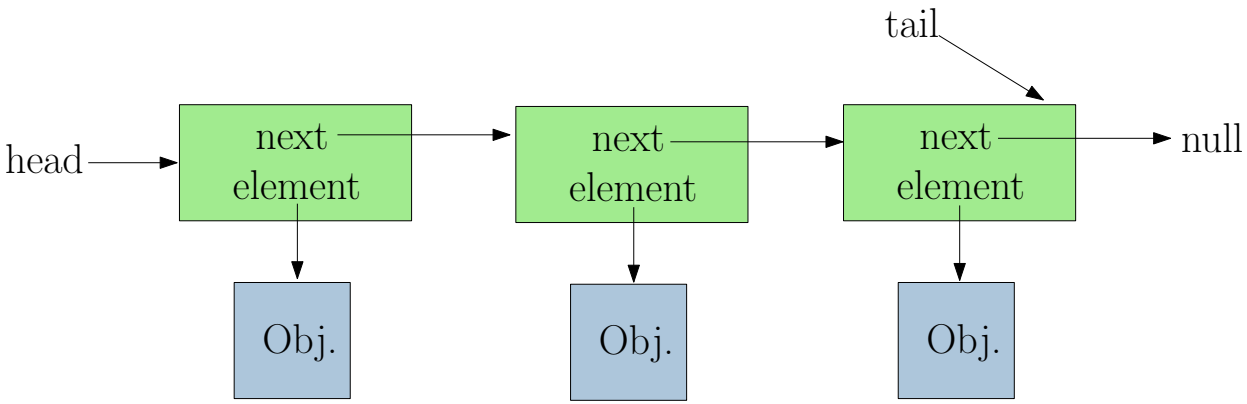
- Insertion complexity? $O(n)$
- Access complexity? $O(1)$
- Space complexity? $O(n)$... usually

Linked List Complexity



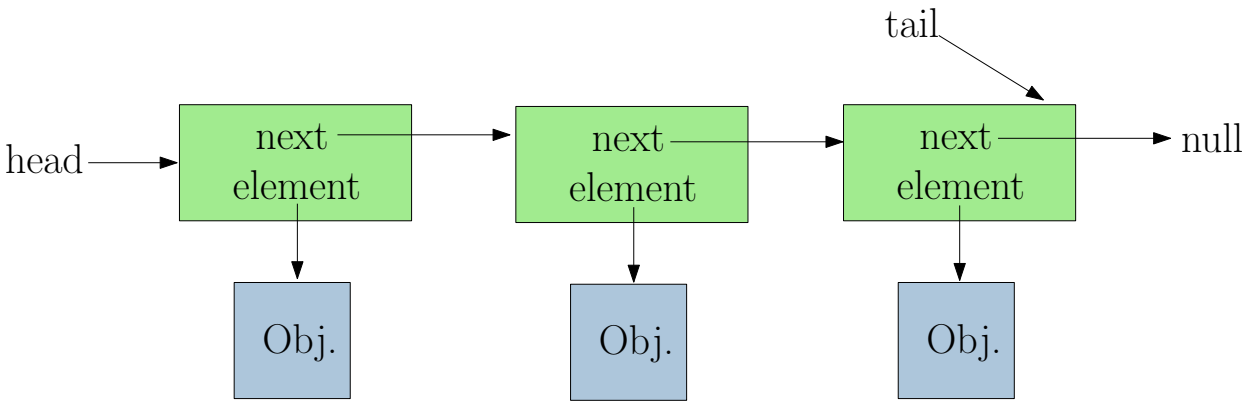
- Insertion complexity (if location known)?

Linked List Complexity



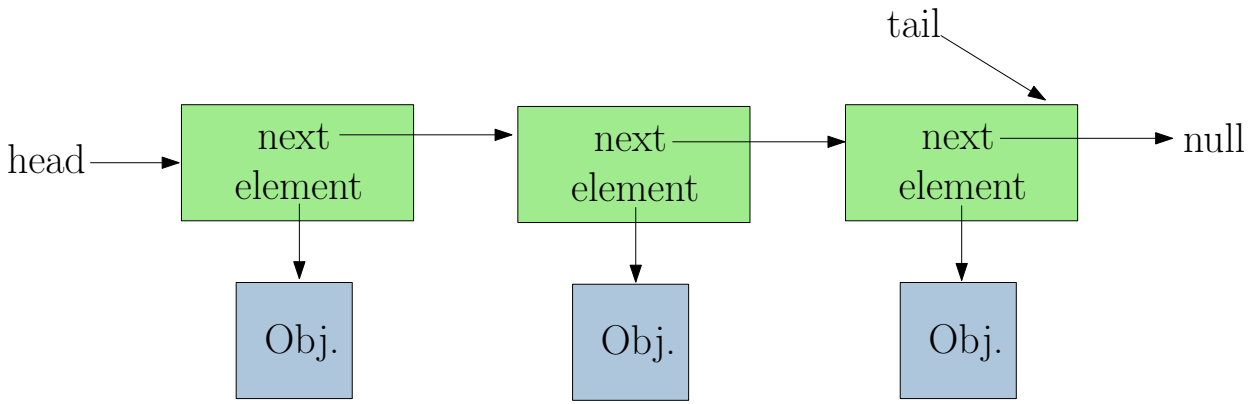
- Insertion complexity (if location known)? $O(1)$
- Access complexity?

Linked List Complexity



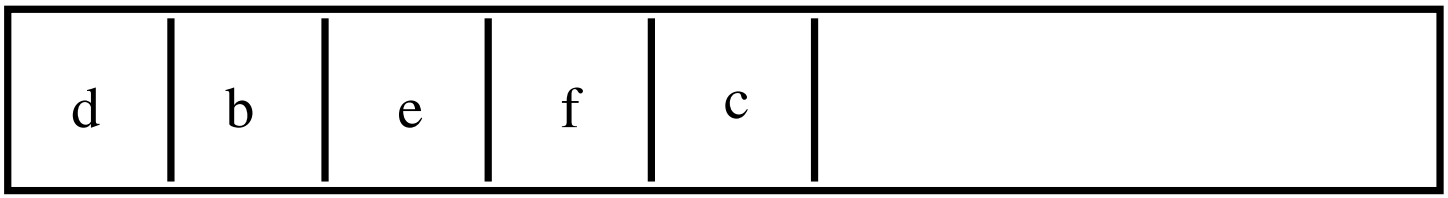
- Insertion complexity (if location known)? $O(1)$
- Access complexity? $O(n)$
- Space complexity?

Linked List Complexity



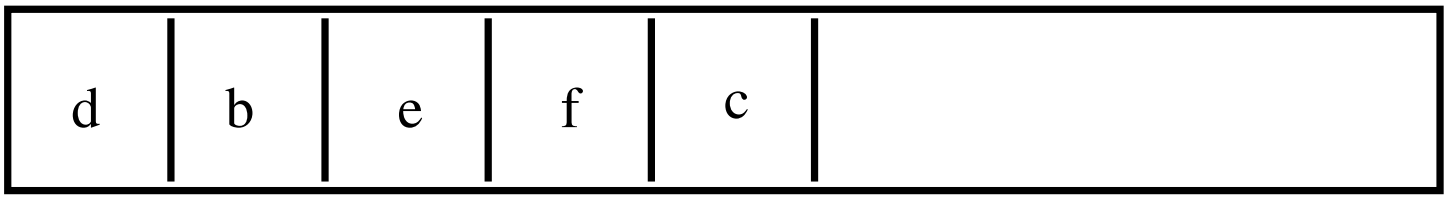
- Insertion complexity (if location known)? $O(1)$
- Access complexity? $O(n)$
- Space complexity? $O(n)$

Queue



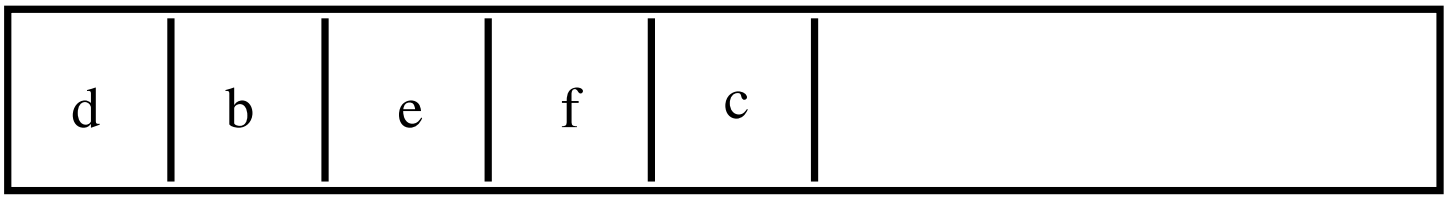
- isEmpty complexity?

Queue



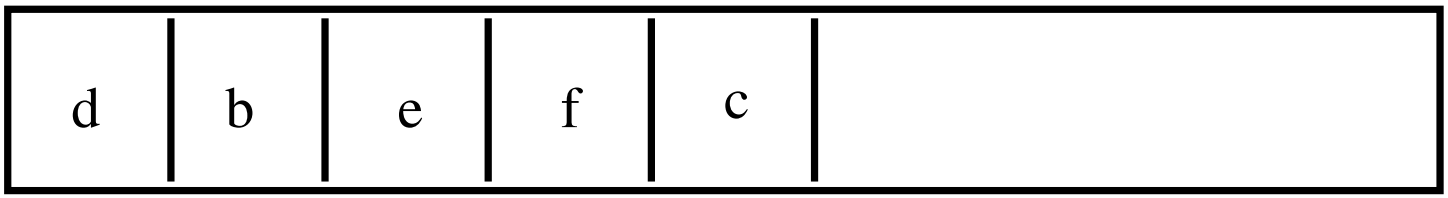
- isEmpty complexity? $O(1)$
- enqueue/dequeue complexity?

Queue



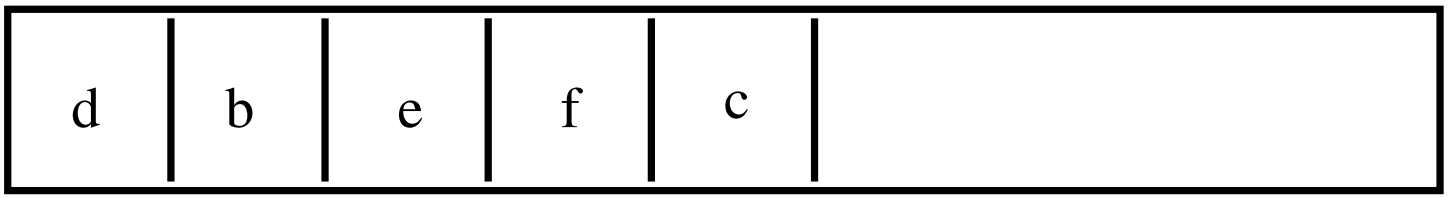
- isEmpty complexity? $O(1)$
- enqueue/dequeue complexity? $O(1)$
- peek complexity?

Queue



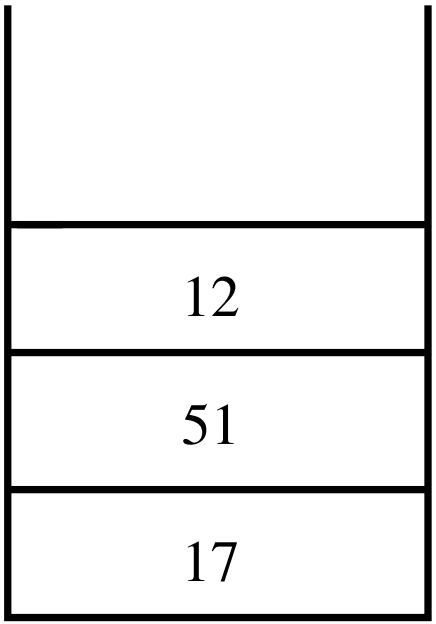
- isEmpty complexity? $O(1)$
- enqueue/dequeue complexity? $O(1)$
- peek complexity? $O(1)$
- Space complexity?

Queue



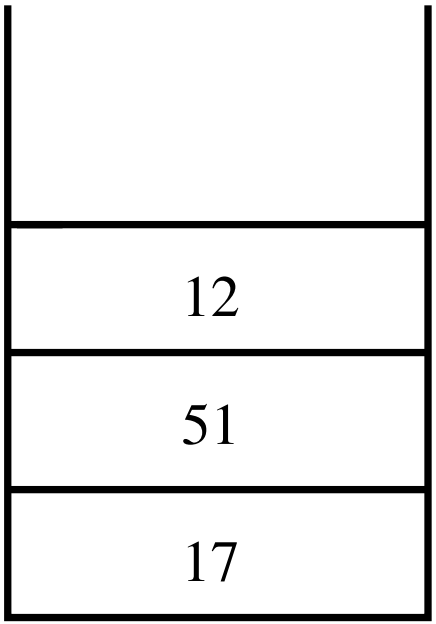
- isEmpty complexity? $O(1)$
- enqueue/dequeue complexity? $O(1)$
- peek complexity? $O(1)$
- Space complexity? $O(n)$

Stack



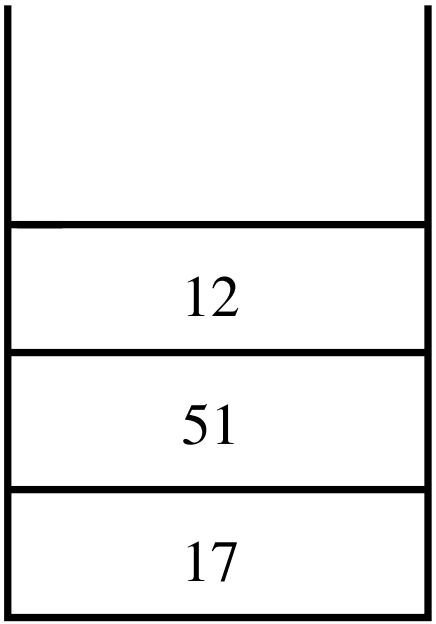
- isEmpty complexity?

Stack



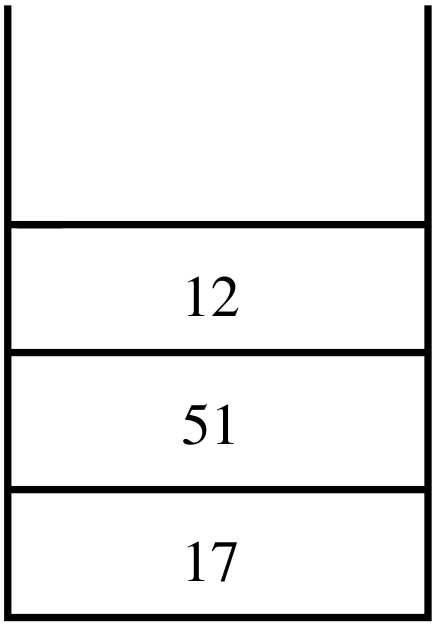
- isEmpty complexity? $O(1)$
- push/pop complexity?

Stack



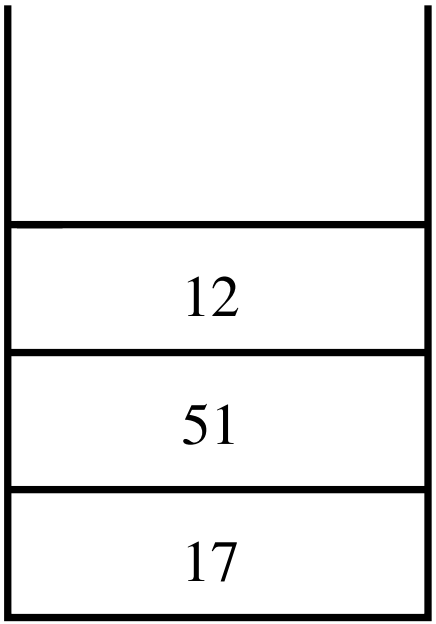
- isEmpty complexity? $O(1)$
- push/pop complexity? $O(1)$
- peek complexity?

Stack



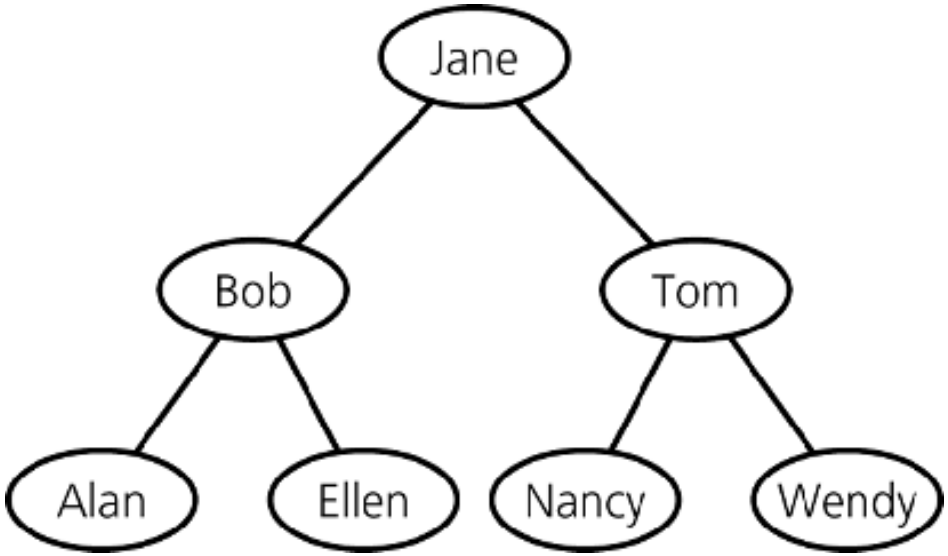
- isEmpty complexity? $O(1)$
- push/pop complexity? $O(1)$
- peek complexity? $O(1)$
- Space complexity?

Stack



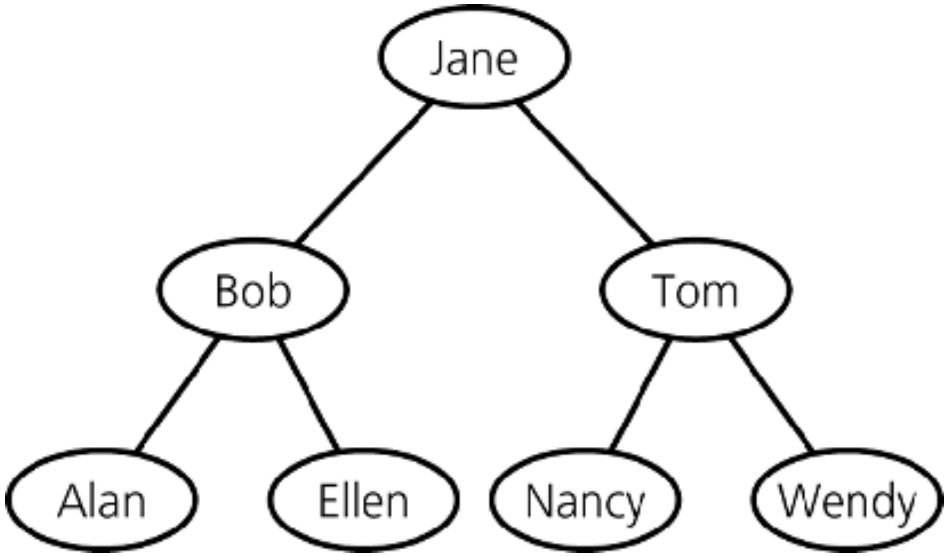
- isEmpty complexity? $O(1)$
- push/pop complexity? $O(1)$
- peek complexity? $O(1)$
- Space complexity? $O(n)$

Binary Search Tree



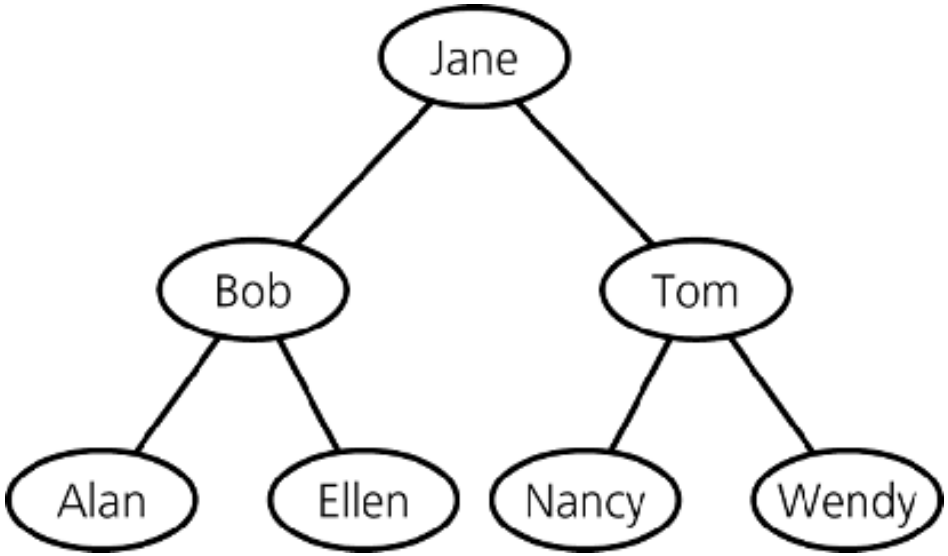
- Find a element?

Binary Search Tree



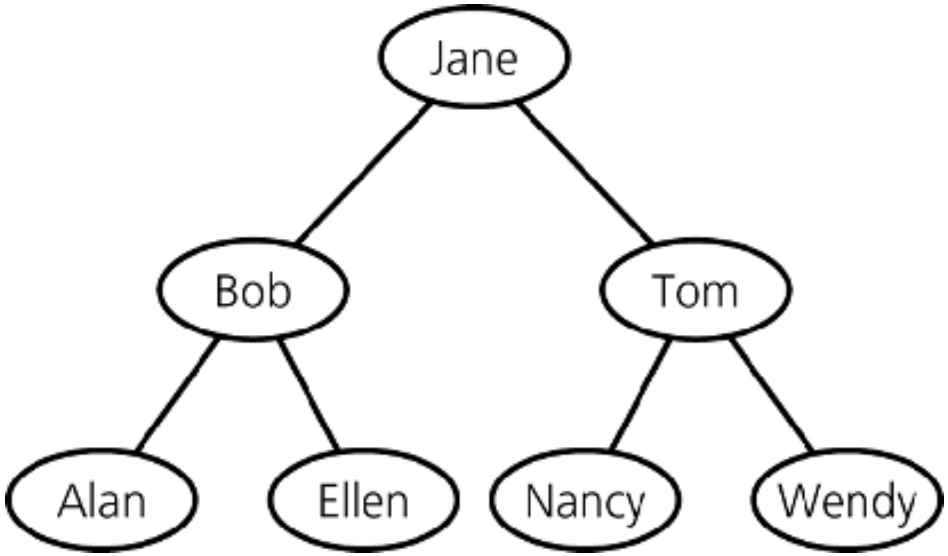
- Find a element? can be $O(n)$ but usually $O(\lg(n))$
  - a balanced binary tree has maximum height of $O(\lg(n))$
  - $O(\lg(n))$ - log base 2 of n
- Print out in alphabetical order?

Binary Search Tree



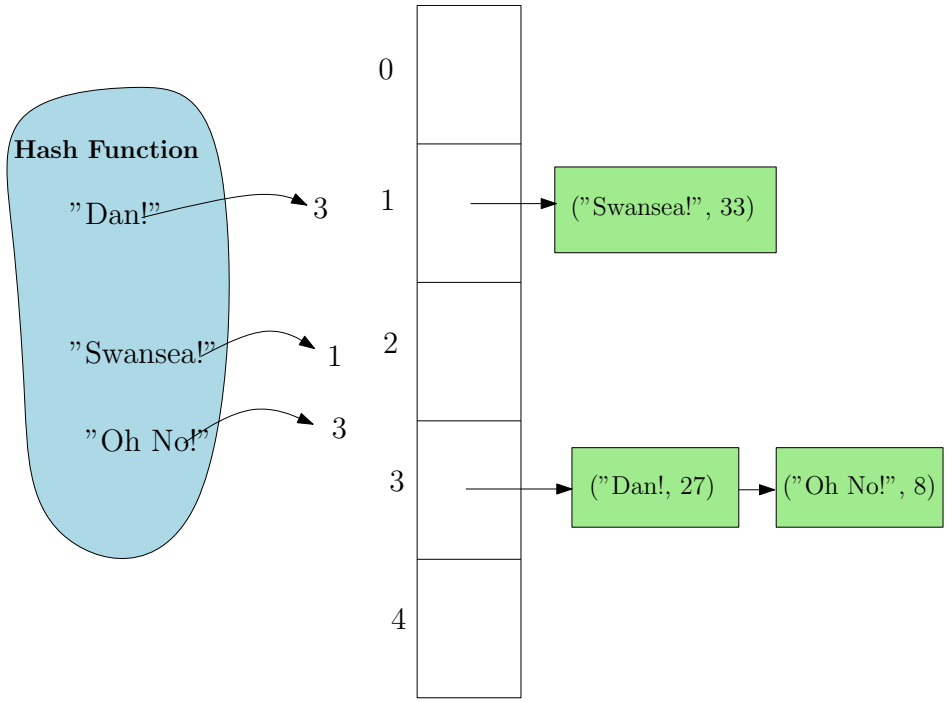
- Find a element? can be $O(n)$ but usually $O(\lg(n))$
  - a balanced binary tree has maximum height of $O(\lg(n))$
  - $O(\lg(n))$ - log base 2 of n
- Print out in alphabetical order? $O(n)$
- Space complexity?

Binary Search Tree



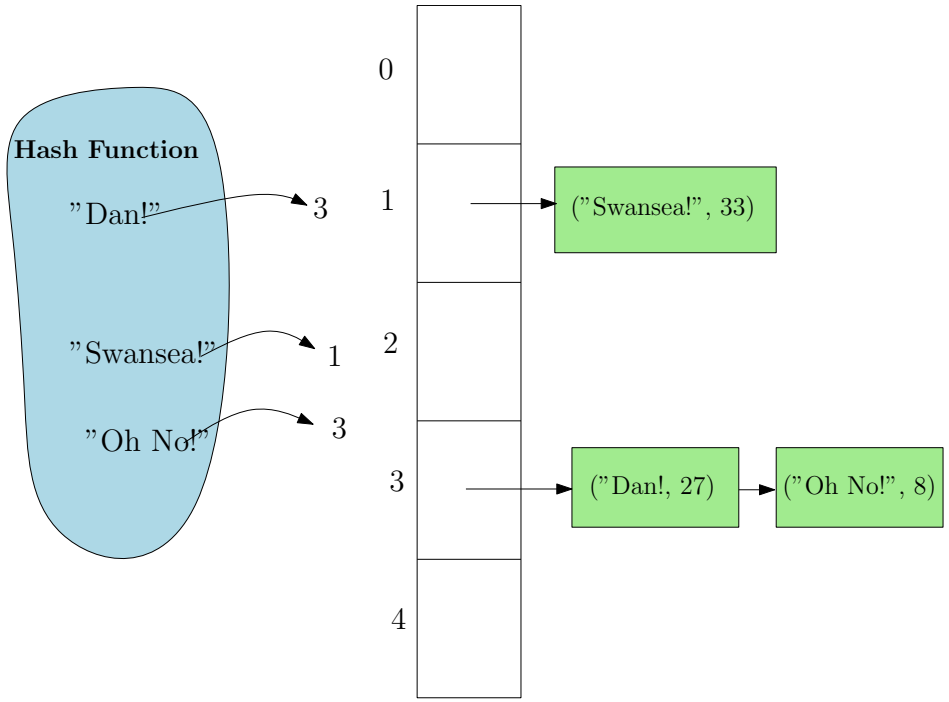
- Find a element? can be $O(n)$ but usually $O(\lg(n))$
  - a balanced binary tree has maximum height of $O(\lg(n))$
  - $O(\lg(n))$ - log base 2 of n
- Print out in alphabetical order? $O(n)$
- Space complexity? $O(n)$

Hash Maps



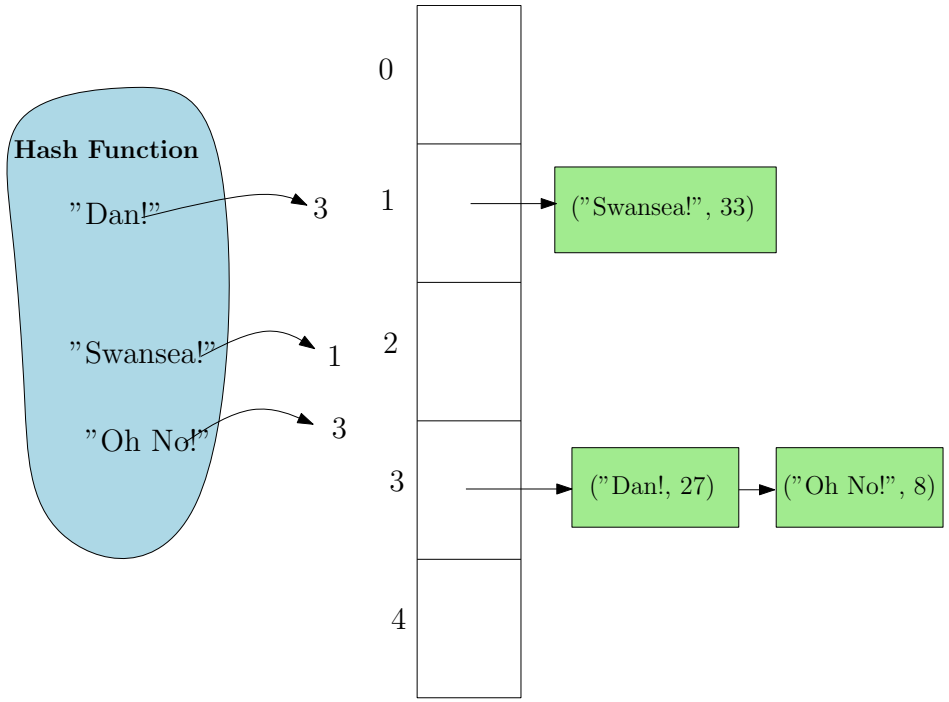
- Find a element?

Hash Maps



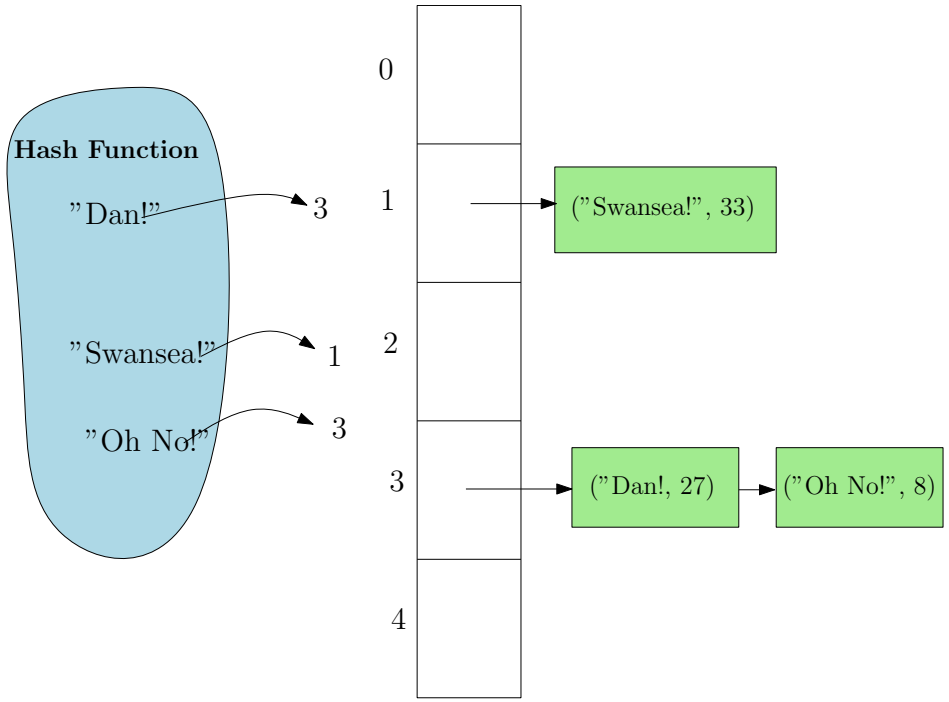
- Find a element? assume good hash function $O(1)$
- Insert an element?

Hash Maps



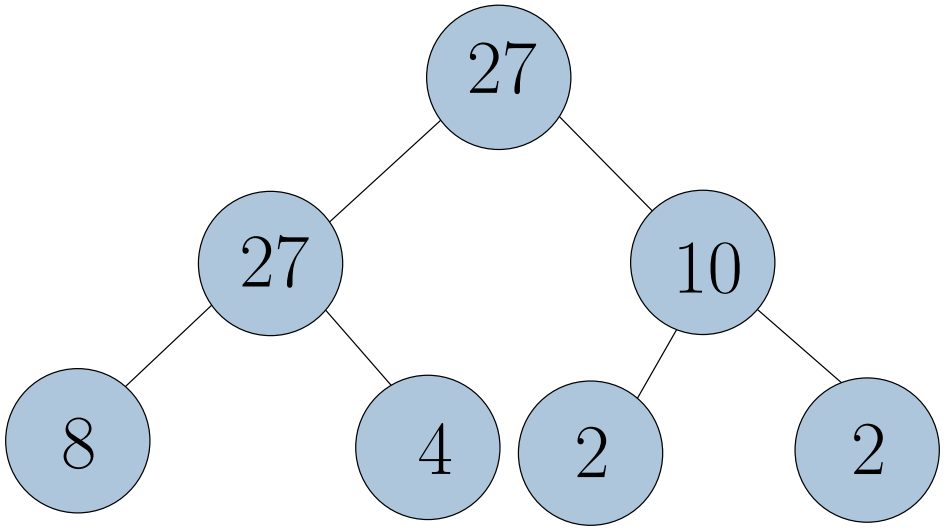
- Find a element? assume good hash function $O(1)$
- Insert an element? assume good hash function $O(1)$
- Space complexity?

Hash Maps



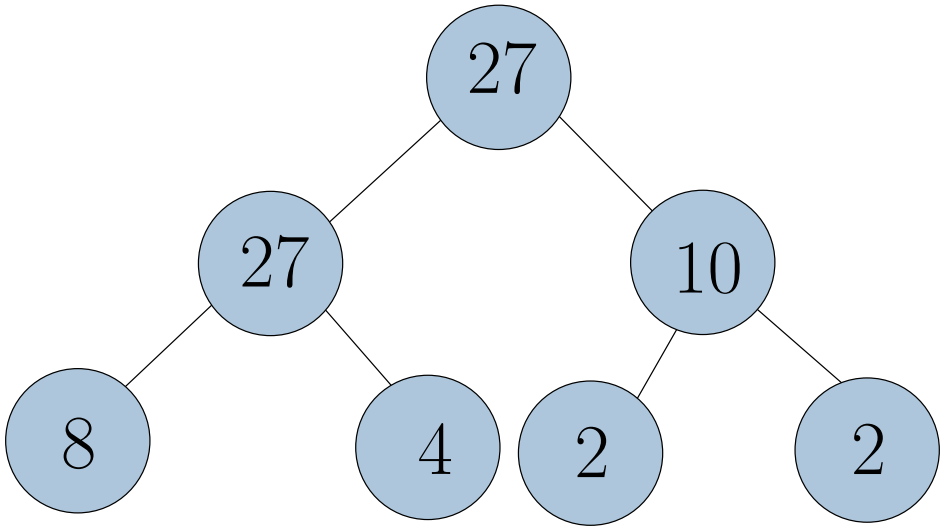
- Find a element? assume good hash function $O(1)$
- Insert an element? assume good hash function $O(1)$
- Space complexity? usually $O(n)$

Heaps and Priority Queues



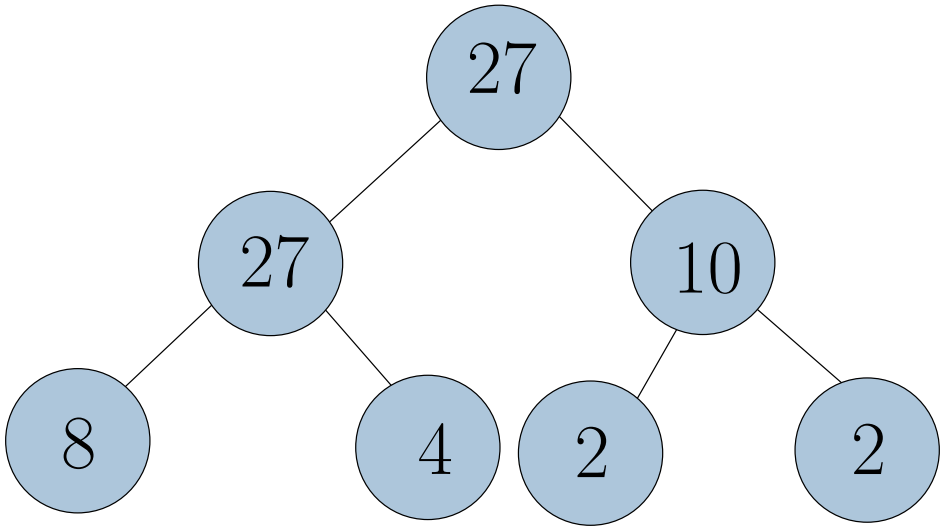
- Look at the top?

Heaps and Priority Queues



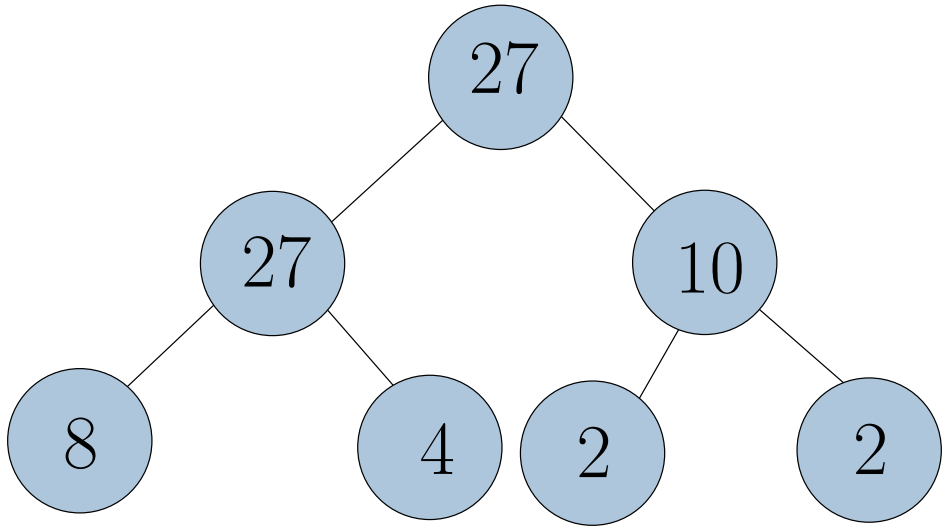
- Look at the top? $O(1)$
- Insert an element?

Heaps and Priority Queues



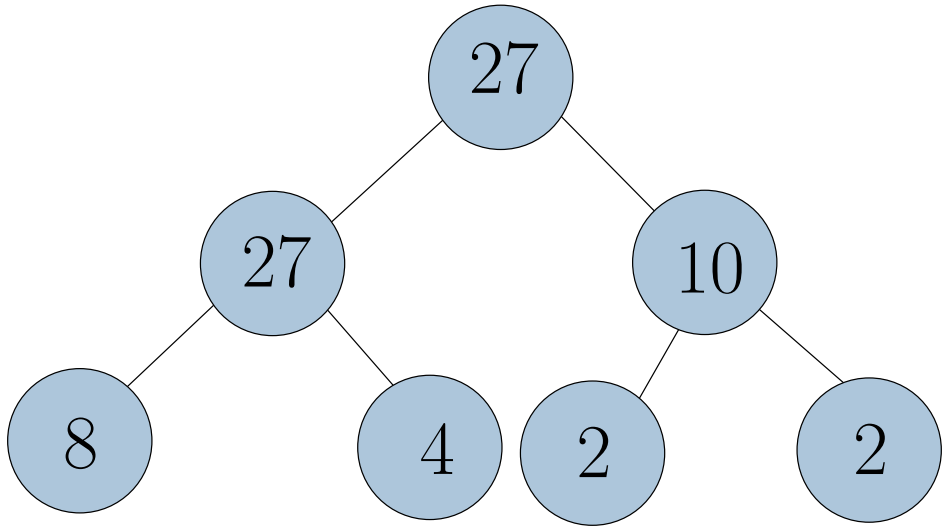
- Look at the top? $O(1)$
- Insert an element? $O(\lg n)$
- Remove the front/root?

Heaps and Priority Queues



- Look at the top? $O(1)$
- Insert an element? $O(\lg n)$
- Remove the front/root? $O(\lg n)$
- Space complexity?

Heaps and Priority Queues



- Look at the top? $O(1)$
- Insert an element? $O(\lg n)$
- Remove the front/root? $O(\lg n)$
- Space complexity? usually $O(n)$

Graphs: Matrix

```
Node[] nodes; //list of n nodes (made by
constructor)
Edge[][] matrix; //nxn matrix specified by
constructor
```

- Look up or remove an edge (given its 2 nodes)?

Graphs: Matrix

```
Node[] nodes; //list of n nodes (made by
constructor)
Edge[][] matrix; //nxn matrix specified by
constructor
```

- Look up or remove an edge (given its 2 nodes)? $O(1)$
- Look up or remove a node?

Graphs: Matrix

```
Node[] nodes; //list of n nodes (made by
constructor)
Edge[][] matrix; //nxn matrix specified by
constructor
```

- Look up or remove an edge (given its 2 nodes)? $O(1)$
- Look up or remove a node? $O(n)$
- Space complexity?

Graphs: Matrix

```
Node[] nodes; //list of n nodes (made by
constructor)
Edge[][] matrix; //nxn matrix specified by
constructor
```

- Look up or remove an edge (given its 2 nodes)? $O(1)$
- Look up or remove a node? $O(n)$
- Space complexity? $O(n^2)$

Graphs: Linked

```
public class Graph {
Node[] nodes;
....   }

public class Node {
Edge[] edgeList; //list of degree nodes
}
```

- Look up or remove an edge?

Graphs: Linked

```
public class Graph {
Node[] nodes;
....   }

public class Node {
Edge[] edgeList; //list of degree nodes
}
```

- Look up or remove an edge? $O(n)$
- Look up or remove an node?

Graphs: Linked

```
public class Graph {
Node[] nodes;
....   }

public class Node {
Edge[] edgeList; //list of degree nodes
}
```

- Look up or remove an edge? $O(n)$
- Look up or remove an node? $O(n)$
- Space complexity?

Graphs: Linked

```
public class Graph {
Node[] nodes;
....   }

public class Node {
Edge[] edgeList; //list of degree nodes
}
```

- Look up or remove an edge? $O(n)$
- Look up or remove an node? $O(n)$
- Space complexity? $O(n+e)$

Complexity of Comparison-Based Sorting

- All sorting algorithms currently are comparison based
  - general sorting algorithms that involves knowing nothing about data
- We know that there are algorithms that take $O(n \lg n)$ comparisons
- But, surely, there are faster algorithms, Dan?

Complexity of Comparison-Based Sorting

- All sorting algorithms currently are comparison based
  - general sorting algorithms that involves knowing nothing about data
- We know that there are algorithms that take $O(n \lg n)$ comparisons
- But, surely, there are faster algorithms, Dan?
- For comparison-based sorts, no.
  - mathematical proof says you need at least $\Omega(n \lg n)$ comparisons
  - proof is hard, but we can do it for fun sometime...
- However, if you know something about the data, you can do better!

Bucket Sort: $O(n)$

- If you know that your data can be easily divided into b buckets, you can use bucket sort which is $O(n)$.
- The algorithm:
  - Create an array of b elements. Each stores an array or linked list.
  - Traverse all n unsorted elements, throwing each into their correct bucket.
  - Assuming each bucket has a small number of elements, sort by any means.
- If any particular bucket ends up with n elements it is $O(n^2)$
- Bucket sort is really useful, especially with hashmaps
  - I've personally used this strategy many times.

Radix Sort: $O(n)$

- If you know that each element of your data has a key of b digits, you can use radix sort which is $O(bn)$.
- Algorithm:
  - start with the least significant digit and apply bucket sort
  - loop through all digits up to the greatest digit
- Assumption is good bucket sorts and small number lengths.
- What to do with non-integer keys...
- Still much debate about its complexity