WIA2005: Algorithm Design and Analysis

Semester 2, Session 2016/17

Lecture 7: Stacks, Queues and Linked List

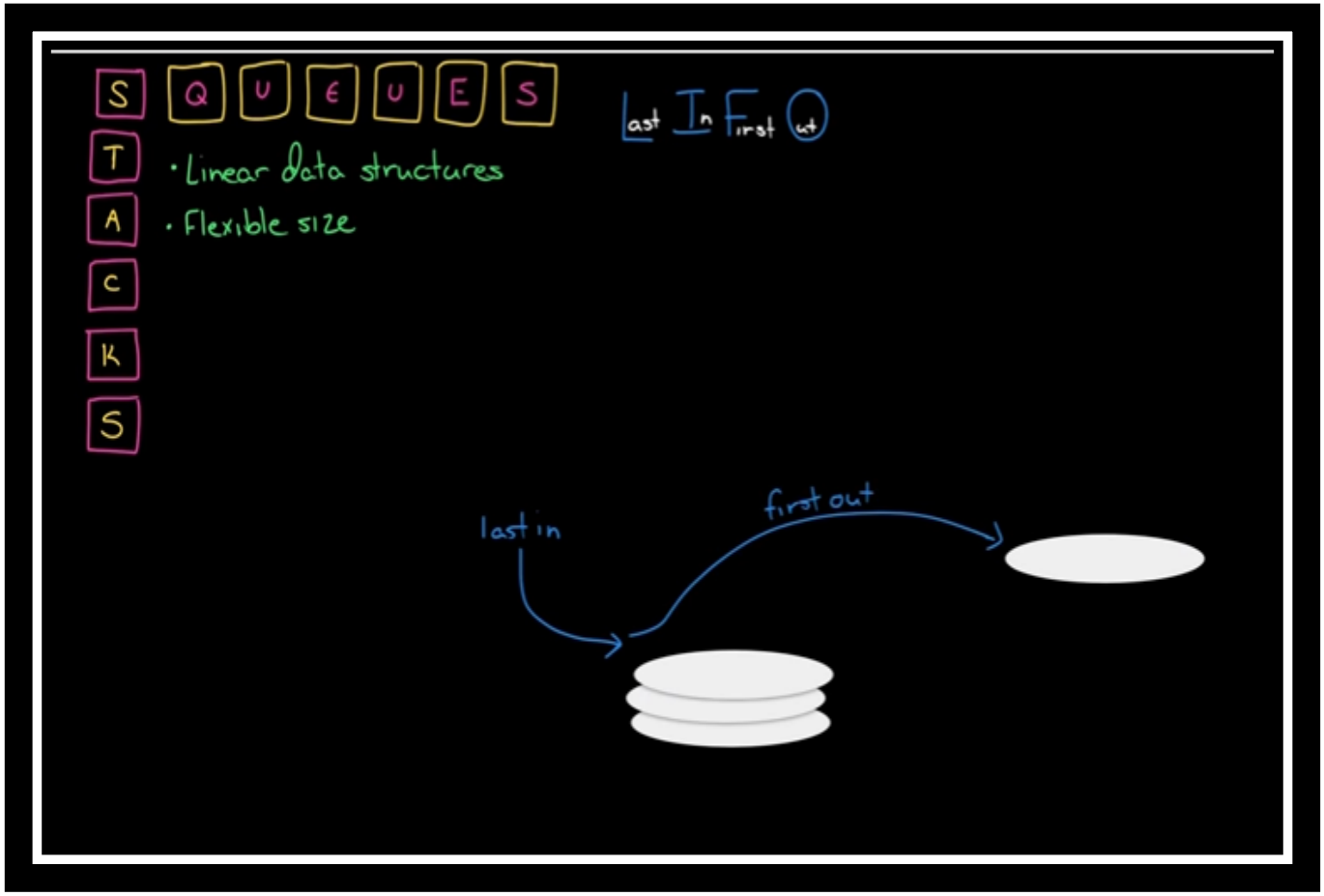
Content

- Stacks
- Queues
- Linked List

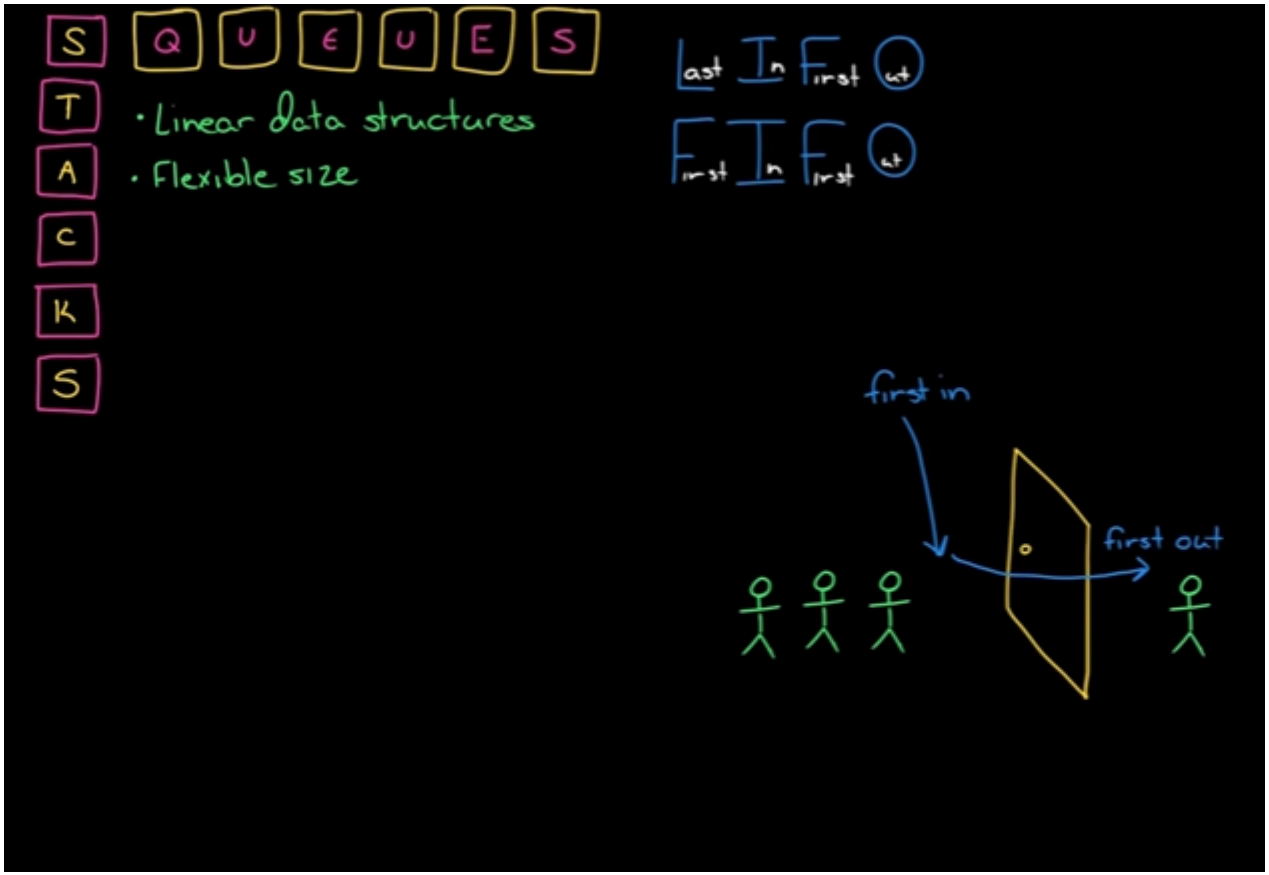
References / Source

- https://www.youtube.com/watch?v=njTh_OwMljA
- https://www.youtube.com/watch?v=wjI1WNcIntg&t=1s

Stacks


STACKS
QUEUES
Last In First Out
• Linear data structures
• Flexible size
last in
first out

Queues

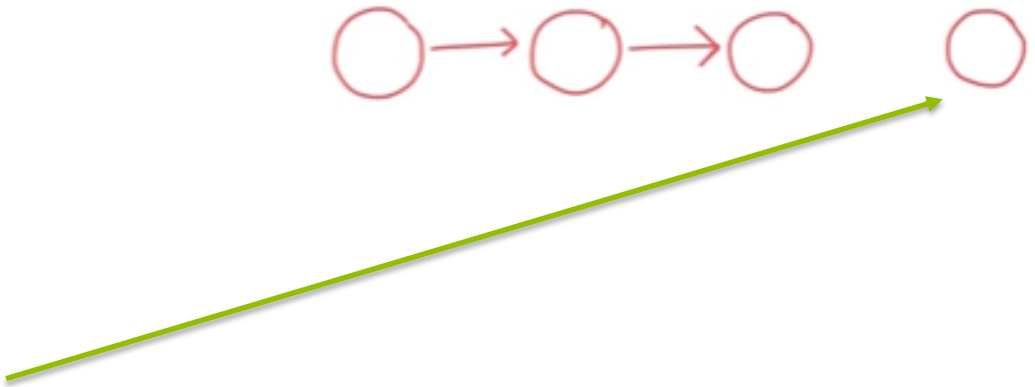

STACKS
QUEUES
• Linear data structures
• Flexible size
Last In First Out
First In First Out
first in
first out

Implementing Queue

```
1  public static class Queue {
2      private static class Node {
3          private int data;
4          private Node next;
5          private Node(int data) {
6              this.data = data;
7          }
8      }
9
10     private Node head;
11     private Node tail;
12
13     public boolean isEmpty() {}
14     public int peek() {}
15     public void add(int data) {}
16     public int remove() {}
17 }
```

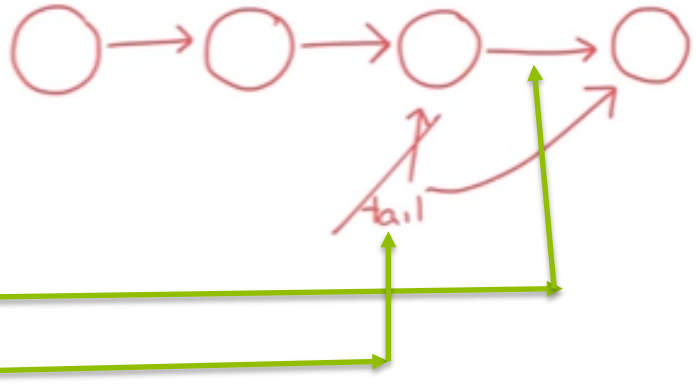
Add

```
9
10      private Node head; // remove from the head
11      private Node tail; // add things here
12
13      public boolean isEmpty() {
14          return head == null;
15      }
16
17      public int peek() {
18          return head.data;
19      }
20
21      public void add(int data) {
22          Node node = new Node(data);
23      }
24
```



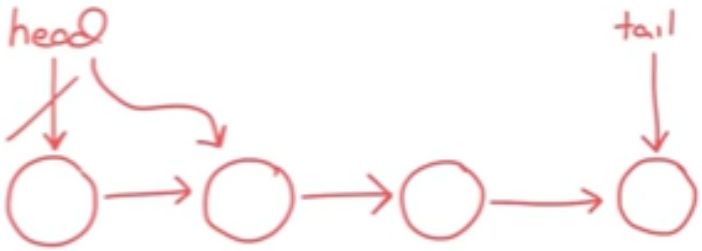
Add

```
10      private Node head; // remove from the head
11      private Node tail; // add things here
12
13      public boolean isEmpty() {
14          return head == null;
15      }
16
17      public int peek() {
18          return head.data;
19      }
20
21      public void add(int data) {
22          Node node = new Node(data);
23          if (tail != null) {
24              tail.next = node;
25          }
26          tail = node;
27          if (head == null) {
28              head = node;
29          }
```



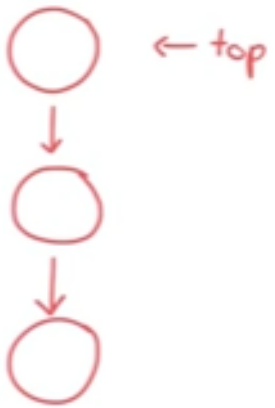
Remove

```
19      }
20
21      public void add(int data) {
22          Node node = new Node(data);
23          if (tail != null) {
24              tail.next = node;
25          }
26          tail = node;
27          if (head == null) {
28              head = node;
29          }
30      }
31
32      public int remove() {
33          int data = head.data;
34          head = head.next;
35          if (head == )
36      }
```



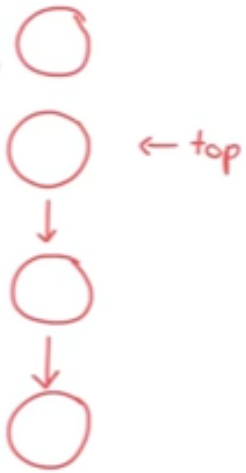
Stacks

```
3           private int data;
4           private Node next;
5           private Node(int data) {
6               this.data = data;
7           }
8       }
9
10      private Node top;
11
12      public boolean isEmpty() {
13          return top == null;
14      }
15
16      public int peek() {
17          return top.data;
18      }
19
20      public void push(int data) {}
21      public int pop() {}
22  }
```



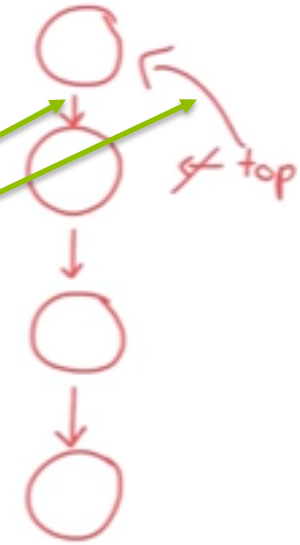
Push

```
4           private Node next;
5           private Node(int data) {
6               this.data = data;
7           }
8       }
9
10      private Node top;
11
12      public boolean isEmpty() {
13          return top == null;
14      }
15
16      public int peek() {
17          return top.data;
18      }
19
20      public void push(int data) {
21          Node node = new Node(data);
22          
23      }
```



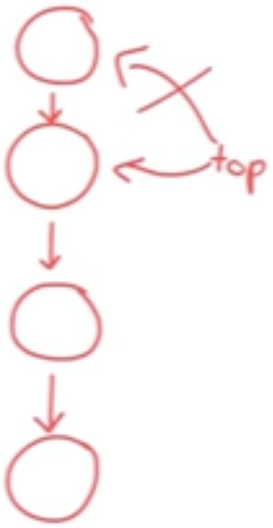
Push

```
8       }
9
10      private Node top;
11
12      public boolean isEmpty() {
13          return top == null;
14      }
15
16      public int peek() {
17          return top.data;
18      }
19
20      public void push(int data) {
21          Node node = new Node(data);
22          node.next = top;
23          top = node;
24      }
25      public int pop() {}
26  }
```

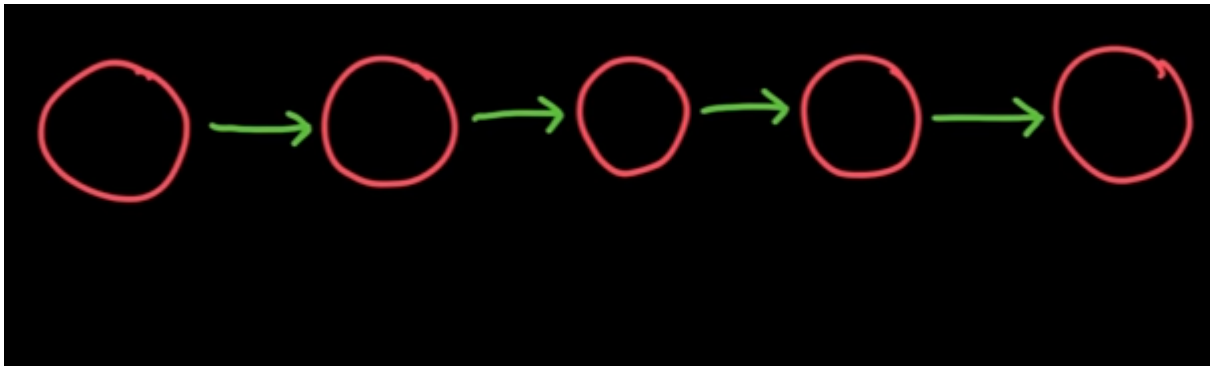


Pop

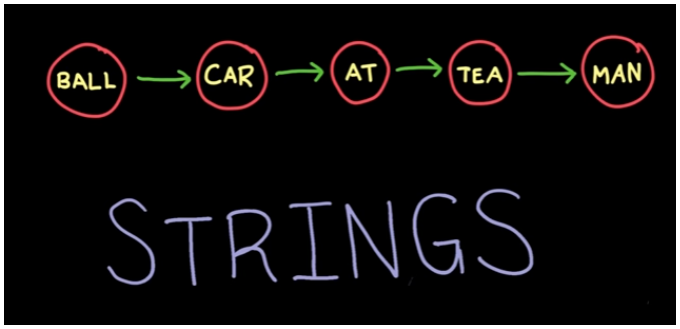
```
12      public boolean isEmpty() {
13          return top == null;
14      }
15
16      public int peek() {
17          return top.data;
18      }
19
20      public void push(int data) {
21          Node node = new Node(data);
22          node.next = top;
23          top = node;
24      }
25
26      public int pop() {
27          int data = top.data;
28          top = top.next;
29          return data;
30      }
```

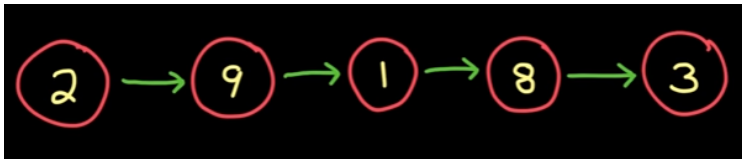


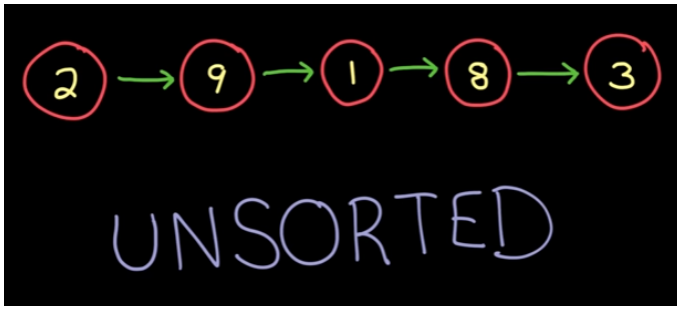
Linked List

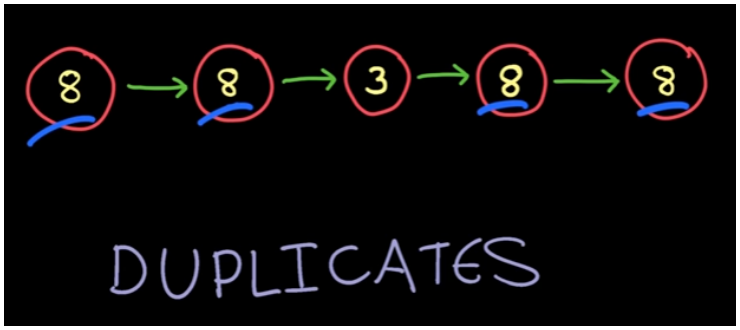


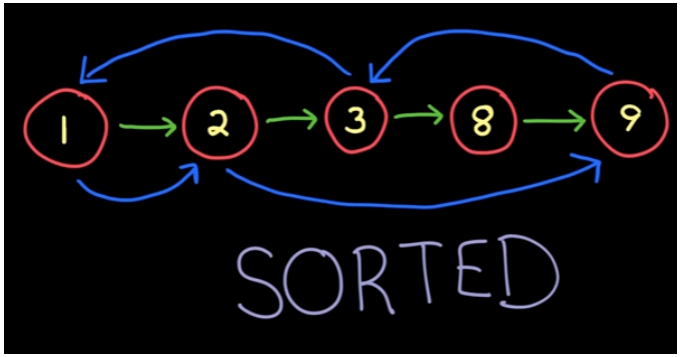
Types of Linked list

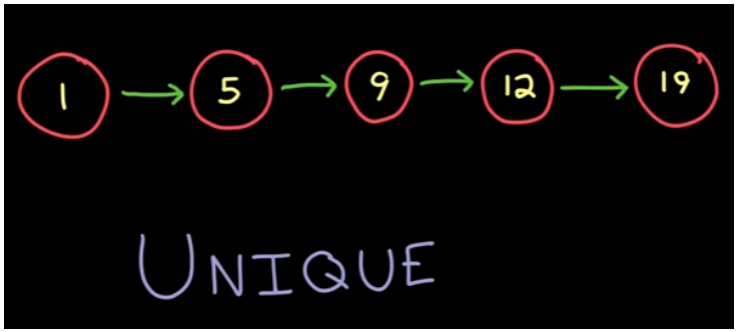




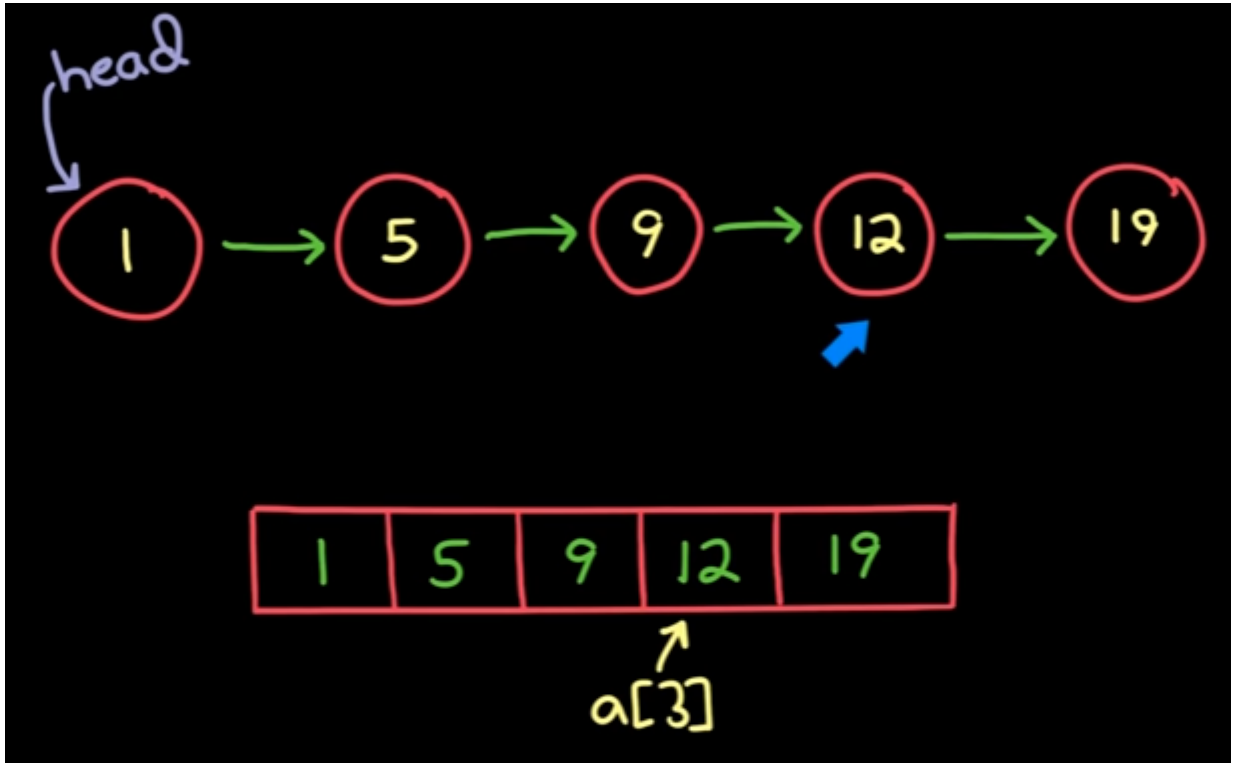




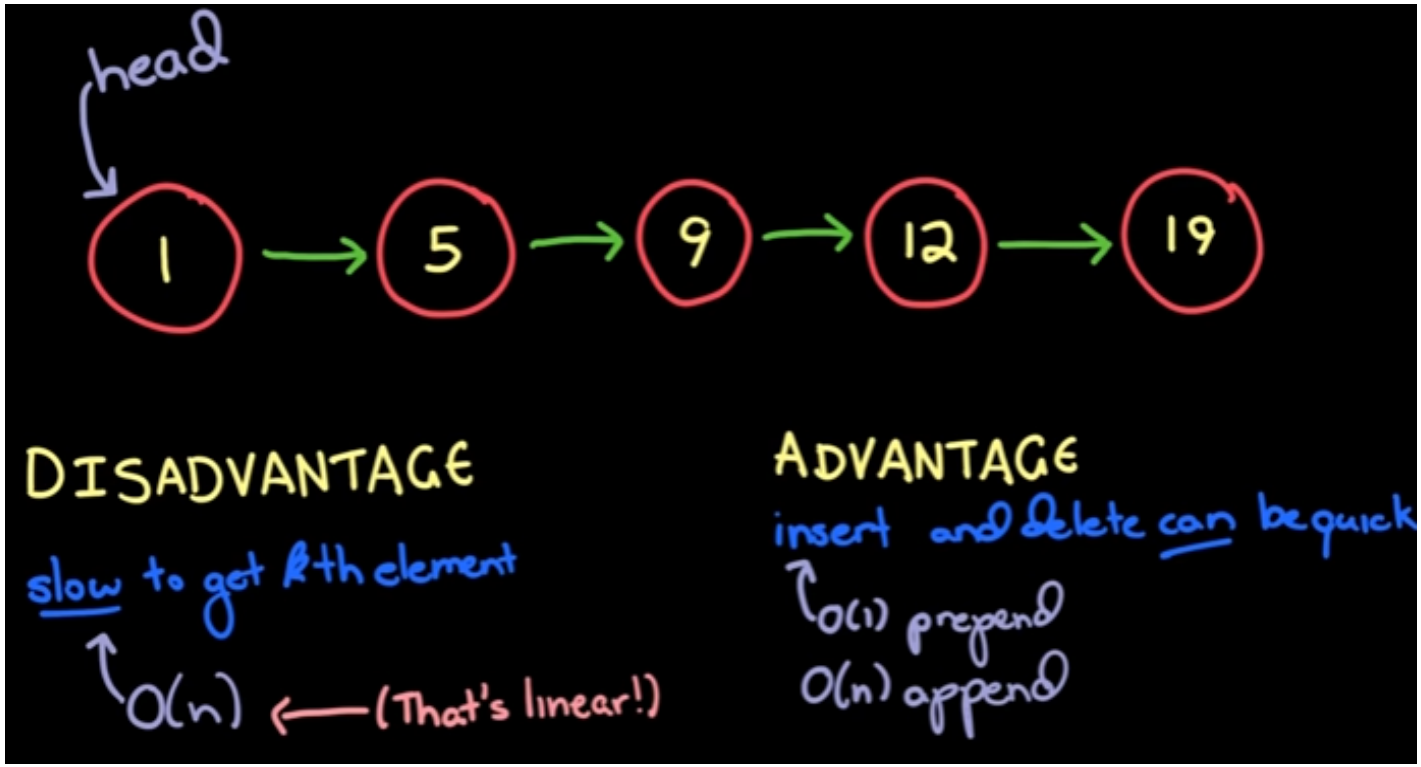




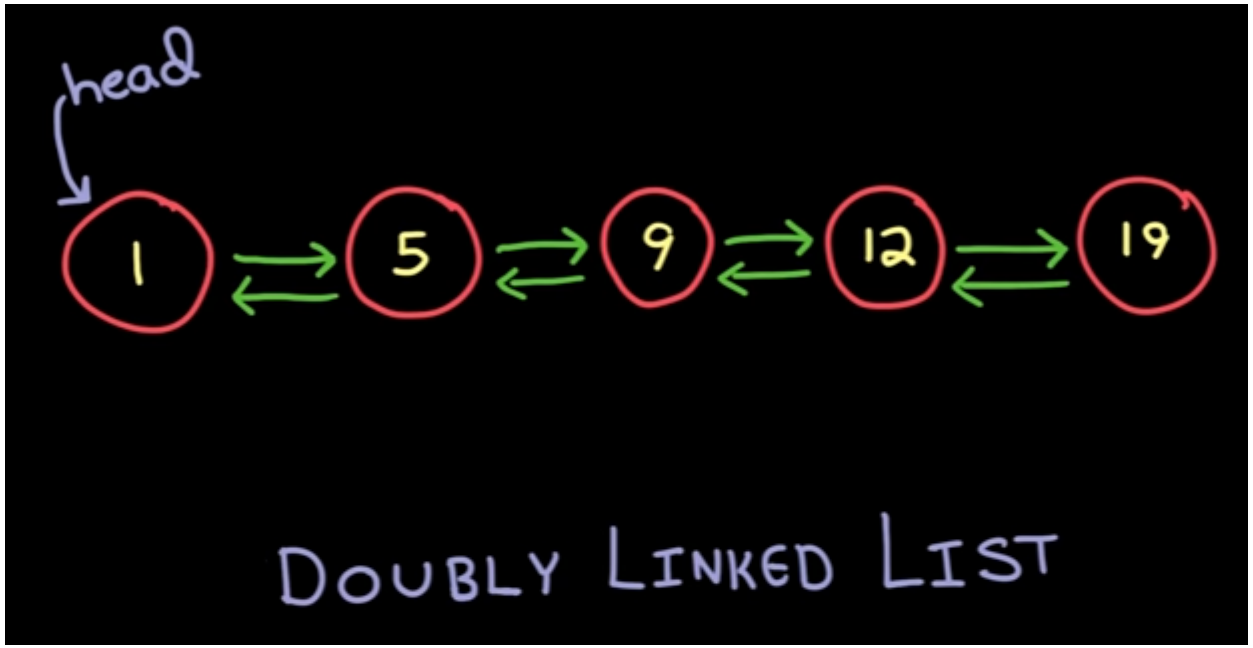
Array vs Linked List



Advantage and disadvantage



Doubly Linked List: previous, next



Implementing Linked List

```
1  public class Node {
2      Node next;
3      int data;
4      public Node(int data) {
5          this.data = data;
6      }
7  }
```

Append: finding null

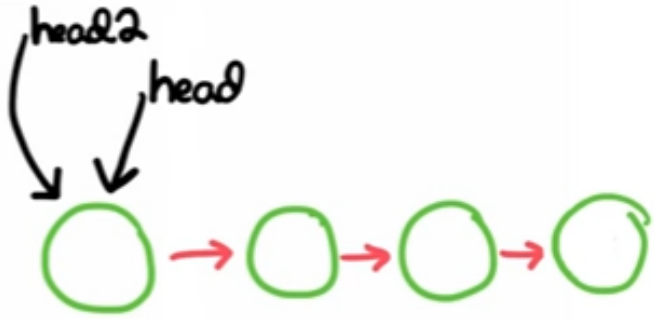
```
1  public class Node {
2      Node next;
3      int data;
4      public Node(int data) {
5          this.data = data;
6      }
7
8      public void append(int data) {
9          Node current = this;
10         while (current.next != null) {
11             current = current.next;
12         }
13
14     }
15 }
```

Append: put new node

```
1  public class Node {
2      Node next;
3      int data;
4      public Node(int data) {
5          this.data = data;
6      }
7
8      public void append(int data) {
9          Node current = this;
10         while (current.next != null) {
11             current = current.next;
12         }
13         current.next = new Node(data);
14     }
15 }
```

How do we know the change of the head value?

```
1  public class Node {
2      Node next;
3      int data;
4      public Node(int data) {
5          this.data = data;
6      }
7
8      public void append(int data) {
9          Node current = this;
10         while (current.next != null) {
11             current = current.next;
12         }
13         current.next = new Node(data);
14     }
15 }
16
17 public class LinkedList {
18     Node head;
19 }
```

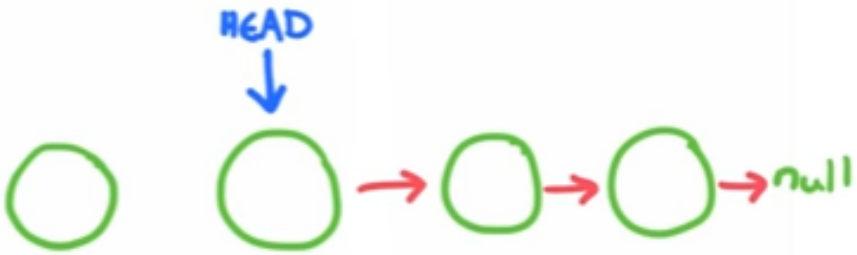


```java
public class Node {
    Node next;
    int data;
    public Node(int data) {
        this.data = data;
    }


}

public class LinkedList {
    Node head;

    public void append(int data) {
        Node current = this;
        while (current.next != null) {
            current = current.next;
        }
        current.next = new Node(data);
    }
}
```

```java
public class Node {
    Node next;
    int data;
    public Node(int data) {
        this.data = data;
    }


}

public class LinkedList {
    Node head;

    public void append(int data) {
        if (head == null) {
            head = new Node(data);
            return;
        }
        Node current = head;
        while (current.next != null) {
            current = current.next;
        }
        current.next = new Node(data);
```

Prepend

```
10
11 public class LinkedList {
12     Node head;
13
14     public void append(int data) {
15         if (head == null) {
16             head = new Node(data);
17             return;
18         }
19         Node current = head;
20         while (current.next != null) {
21             current = current.next;
22         }
23         current.next = new Node(data);
24     }
25
26     public void prepend(int data) {
27         Node newHead = new Node(data);
28
29     }
30 }
```

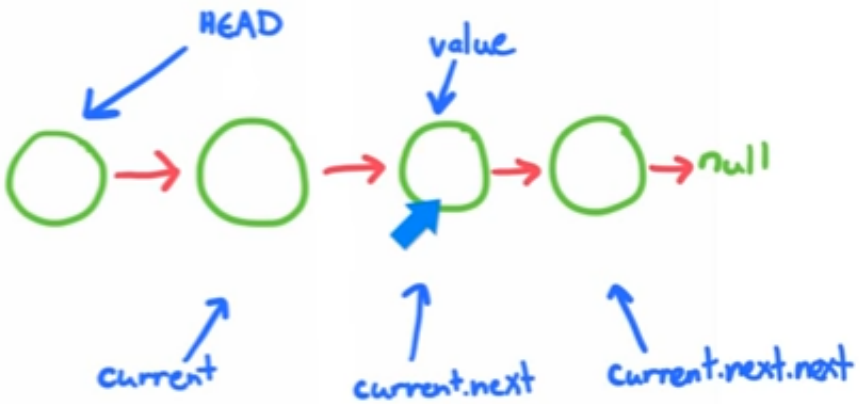


Prepend


```
10
11  public class LinkedList {
12      Node head;
13
14      public void append(int data) {
15          if (head == null) {
16              head = new Node(data);
17              return;
18          }
19          Node current = head;
20          while (current.next != null) {
21              current = current.next;
22          }
23          current.next = new Node(data);
24      }
25
26      public void prepend(int data) {
27          Node newHead = new Node(data);
28          newHead.next = head;
29          head = newHead;
30      }
31  }
```

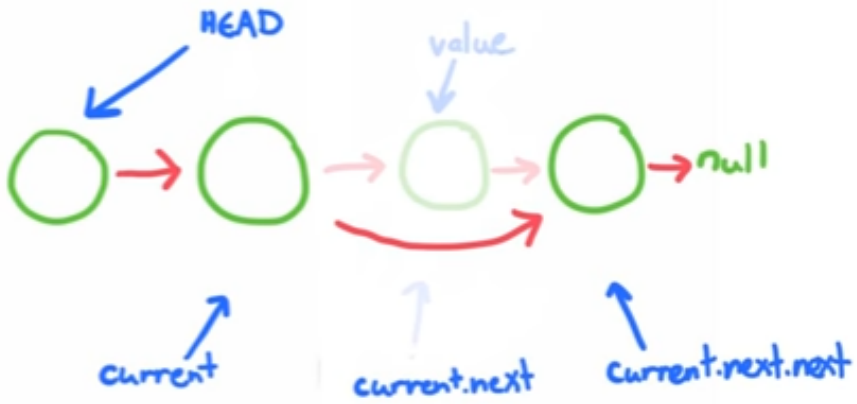
Delete

```
19          Node current = head;
20          while (current.next != null) {
21              current = current.next;
22          }
23          current.next = new Node(data);
24      }
25
26      public void prepend(int data) {
27          Node newHead = new Node(data);
28          newHead.next = head;
29          head = newHead;
30      }
31
32      public void deleteWithValue(int data) {
33          if (head == null) return;
34
35          Node current = head;
36          while (current.next != null) {
37              if (current.next.data == data) {
38
39              }
40          }
41      }
42  }
```



Delete

```
20          while (current.next != null) {
21              current = current.next;
22          }
23          current.next = new Node(data);
24      }
25
26      public void prepend(int data) {
27          Node newHead = new Node(data);
28          newHead.next = head;
29          head = newHead;
30      }
31
32      public void deleteWithValue(int data) {
33          if (head == null) return;
34
35          Node current = head;
36          while (current.next != null) {
37              if (current.next.data == data) {
38                  current.next = current.next.next;
39                  return;
40              }
41              current = current.next;
42          }
43      }
44  }
```



Deleting the Head node

```
23          current.next = new Node(data);
24      }
25
26      public void prepend(int data) {
27          Node newHead = new Node(data);
28          newHead.next = head;
29          head = newHead;
30      }
31
32      public void deleteWithValue(int data) {
33          if (head == null) return;
34          if (head.data == data) {
35              head = head.next;
36              return;
37          }
38
39          Node current = head;
40          while (current.next != null) {
41              if (current.next.data == data) {
42                  current.next = current.next.next;
43                  return;
44              }
45              current = current.next;
```

```
26    public void prepend(int data) {
27        Node newHead = new Node(data);
28        newHead.next = head;
29        head = newHead;
30    }
31
32    public void deleteWithValue(int data) {
33        if (head == null) return;
34        if (head.data == data) {
35            head = head.next;
36            return;
37        }
38
39        Node current = head;
40        while (current.next != null) {
41            if (current.next.data == data) {
42                current.next = current.next.next;
43                return;
44            }
45            current = current.next;
46        }
47    }
```