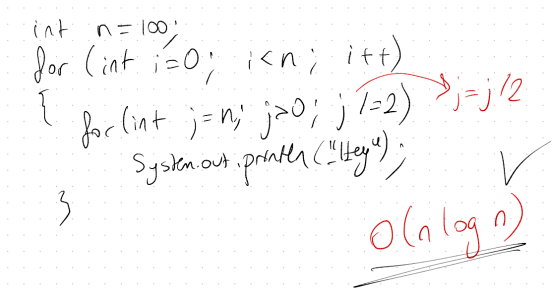
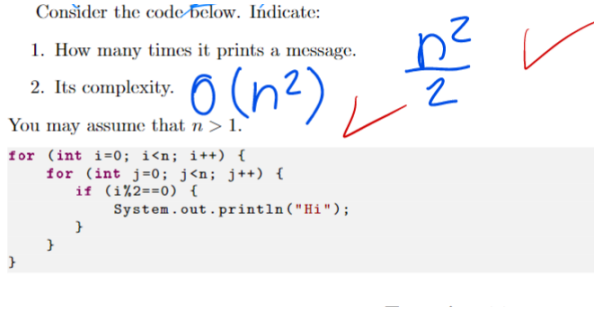
- Abstract classes and interfaces - both used for abstraction + cannot be instantiated
    - Abstract classes extend another class “extends” - child classes must have all functionality listed; you can instantiate a variable and an object in the class
    - Interfaces are implementations “implements” + can only have static and final attributes - child classes must have all functionality listed
- UML - top is class, middle is attributes (+ is public, - is private), bottom is methods
- Overloading is within same class; Overriding is subclass overriding a superclass (allowed through polymorphism)
- Non-primitive = strings, arrays, classes
- Runtime - big O finds upper bound (worst case scenario)
    - Specify base when writing how many times code will print
    - In logarithmic runtime - logn base does not matter for big O
    - When the inner for loop is dependent on the outer, it is usually (n(n+1))/2 times
- Accessing item in array = O(1), removal/insertion = O(n)
- Stacks: Push: Puts an element into the stack; Peek: Looks at the top of the stack; Pop: Looks and removes the element at the top of the stack
- Queues: Offer : Adds an element to the end of the Queue; Peek : Looks at the front of the Queue; Poll : Looks and removes the front element of the queue






```
for (int i=0; i<n; i++){
    for (int j=0; j<i; j++){
        System.out.println(x: "Quadratic run time...");
    }
}
```

```
for (int i = 0; i < n^2; i++){
   for(int j = n; j > 0; j/=3){
       if(i > 3){
          System.out.print("yippee");
       }
   }
}
```

LEFT: n*((n+1)/2)) = num of prints, O(n^2)

RIGHT: O(n^2 * log(n)) or O(n^2 * log3(n))

BOTTOM: runs 3*((n+1)/2)*log_2(n) ; O(n*log(n))

```
for(int i = 0; i < n; i++){
    if(i % 2 == 0){
        for(int j = 1; j < n; j*=2){
            System.out.println("count me!");
            System.out.println("count me!");
            System.out.println("count me!");
        }
    }
}
```

```
public SingleLL<E> append(SingleLL<E> l2) {
    if (this.head == null) {
        return l2;
    }
    if (l2.head == null) {
        return this;
    }
    Node<E> lastNode = this.head;
    while (lastNode.next != null) {
        lastNode = lastNode.next;
    }
    lastNode.next = l2.head;
    return this;
}
```

```
public boolean remove(double number) {
   // Your code here
   Node curr = head;
   while (curr != null) {
         if (curr.data == number) {
            if (curr == head) {
               if (curr.next != null) {
                  head = curr.next;
                  head.previous = null;
               }
               else {
                  head = null;
               }
               return true;
            }
            if (curr.next == null) {
               curr.previous.next = curr.next;
               tail = curr.previous;
               return true;
            }
            curr.previous.next = curr.next;
            curr.next.previous = curr.previous;
            return true;
         }
         curr = curr.next;
   }
   return false;
}
}
```

```
public void insert(double number) {
   Node main_node = new Node(number);
   if (head == null) {
      head = main_node;
      tail = main_node;
      return;
   }
   if (head.data > number) {
         linkNodes(main_node, head);
         head = main_node;
         return;
      }
   Node curr = head;
      while (curr != null) {
         if (curr.data > number) {
            linkNodes(curr.previous, main_node);
            linkNodes(main_node, curr);
            return;
         }
         curr = curr.next;
      } //end
      linkNodes(tail, main_node);
      tail = main_node;
   }
```

```
//create a new list for reverse()
void reverse(){
      SLL<E> newList = new SLL<E>();
      Node<E> current = head;
      int newListIndex = size - 1;
      while(current!=null)
      {
            newList.add(current.data, newListIndex);
            newListIndex --;
            current = current.next;
      }
      this = newList;
}
```

```
//Using SLL's getAt and set functions with index. Not creating a new list
void reverse(){
      for(int i= 0; i<size; i++){
            int rear = size - i - 1; //to trace the linked list from back
            if(i < rear){
                  E temp = getAt(i);
                  set(i, getAt(rear));
                  set(rear, temp);
            }else
                  break;
      }
}
```

Determine the time growth rate of the following code. You must provide details on how it was established. You may assume that $n > 1$.

```
for (int i=0; i<n; i++) {
  if (i%2==0) {
    for (int j=0; j<i; j++) {
       System.out.println("Hi");
    }
  }
}
```

$0+2+4+6+8+\ldots+n-1$

$2(0+1+2+3+\ldots+\frac{n-1}{2})$

$2(\frac{n-1}{2}*(\frac{n-1}{2}+1))$

$O(n^2)$

$\frac{(n-1)}{2}*\frac{(n+1)}{2}=\frac{n^2-1}{4}$

Exercise 10

Consider the code below. Indicate:

1. How many times it prints a message.

n^2

```
public void add(E data, int index){
    if(index > this.length() || index<0){throw new IllegalArgumentException("Index out of bounds");}
    if(index == 0){
        head = new Node<>(data,head);
    } else{
        Node<E> current = head;
        for(int i = 0; i < index-1;i++){
            current = current.next;
        }
        current.next = new Node<>(data, current.next);
    }
    size++;

}
```