Suppose we have the following definition of a Node:

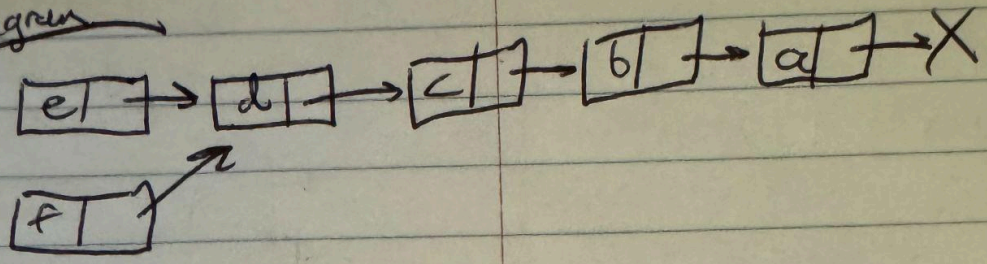
```
class Node<T> {
  public T data;
  public Node<T> next;

  public Node<T>(T data, Node<T> next) {
    this.data = data
    this.node = node
  }
}
```

Diagramming Question

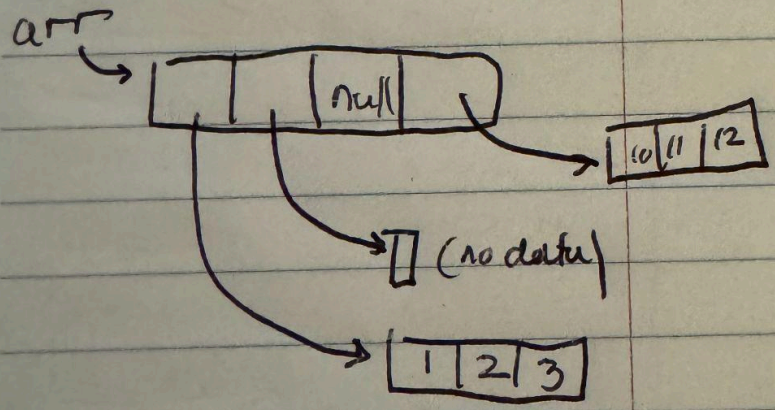
Suppose that I have the following code. Diagram the objects allocated and the links between them.

```
Node<String> a = new Node<String>(“a”, null);
Node<String> b = new Node<String>(“b”, a);
Node<String> c = new Node<String>(“c”, b);
Node<String> d = new Node<String>(“d”, c);
Node<String> e = new Node<String>(“e”, d);
Node<String> f = new Node<String>(“f”, d);  // Note the difference here!
```

Diagram



array of arrays



Implementation Question

Write a function called average which gets the average of a singly linked list of doubles (or more specifically Double, which is a pointer to a double and acts like a double for all purposes here). We are given the head node. It will likely return a divide by zero error if head is null:

```
public double average(Node<Double> head) {
  Sum = 0
  Count = 0
  While (head != null)
    Sum += head.value
    Count += 1
    Head = head.next

  Return sum / count
}
```

What is the big-Oh running time of your function?
O(n)

Diagramming Question

Suppose that I have an array of arrays (remember that's an array of pointers, all pointing to other arrays)

```
int[][] arr = { {1, 2, 3}, {}, null, {10, 11, 12} };
```

Diagram how this is stored in memory.

Running Time

What is the big-Oh running time of the following function in terms of n?

```
int countingOnAStack(int n) {
  Stack<Integer> stack = new ArrayStack(1000);
  stack.push(n);
  return countingOnAStack(new ArrayStack(1000);
}

int countingOnAStack(Stack<Integer> stack) {
  while (!stack.isEmpty())
    int item = stack.pop();

    if (item >= 0) {
      stack.push(item - 1);
      stack.push(item - 1)
    }
  }
}
```

O(2^n) [exponential time]

Running Time

What is the running time of the following code in terms of n?

```
// arr is an array of length n
int currentMax = 0;
for(i = 0; i < n; i++) {
  if (arr[i] > currentMax) {
    currentMax = arr[i];
  }

  // Increase the rest of the array
  for(j = i + 1; j < n; j++) {
    arr[j] += 1
  }
}
```

O(n^2)

Why: Loop within a loop, both linear counts, “triangular” in the sense that the inner loop is only from i + 1 to n, but we’ve established that this is still n^2

Running Time + Coding

What is the worst case running time of findAll in terms of n, which is the length of an array?

```
//  int[] arr is an array of length n

int findInArray(int element) {
  for(i = 0; i < n; i++) {
    if (element == i) return i;
  }
  return i;
}

int[] findAll() {
  int[] answer = new int[n]; // Allocate an array of length n

  for(int i = 0; i < n; i++) {
   answer[i] = findInArray(i)
  }

  return answer
}
```

BTW - this is a weird one - I wouldn't like it as an exam question.

BUT - findInArray runs in time i, and you're counting i = 0 to n - 1, so this is O(n^2)

Could you write a better version of findAll that accomplishes the same thing with a better big-Oh running time?

Coding

You are given a linked list. Construct another Linked List that is your input in reverse order.

What is the big-Oh running time of your algorithm?

ANSWER:
You'll have the SinglyLinkedList.java code from the book available, and the easiest way to do it would be this. Note I've removed generics; that's ok for this type of code (pseudocode is ok too).

```
SinglyLinkedList constructReverse(SinglyLinkedList list) {
        SinglyLinkedList newList = new SinglyLinkedList();
        while(!list.isEmpty()) {
            newList.addFirst(list.removeFirst())
        }
}
```

O(n)

Stacks (Basic)

What gets printed out by the following code?

```
Stack<Integer> stack = new ListStack<Integer>()
stack.push(2)
stack.push(8)
int x = stack.pop()
stack.push(10)
stack.push(x)
stack.push(x)
stack.push(12)
x = stack.pop();
stack.push(x + 1)

while(!stack.isEmpty()) {
  System.out.println(stack.pop())
}
```

Answer
13
8
8
10
2

In more detail here's what the stack looks like after each step (this isn't what the question asked, but you would have to think it out to answer the question):
2
2 8
2 (and x = 8)
2 10 (and x = 8)
2 10 8 (and x = 8)
2 10 8 8 (and x = 8)
2 10 8 8 12 (and x = 8)
2 10 8 8 (and x = 12)
2 10 8 8 13
(and then the stack gets popped and printed starting from 13 and working back to 2)

Queue (Basic)

You have a queue of Integers stored in a circular array of length 5. Draw the values of the Queue after the following operation. Shade in the space in the array that is unused.

Queue<Integer> q = new ArrrayQueue(5);

q.enqueue(4)
q.enqueue(3)
q.enqueue(2)

4 (start)	3	2 (end)	XXXX	XXXX

q.dequeue()
q.enqueue(8)
q.enqueue(10)

XXXX	3 (start)	2	8	10 (end)

int x = q.dequeue()

q.dequeue()
q.enqueue(x)
q.enqueue(x * 2)

(also draw the starting and ending elements of the queue)

3	6 (end)	XXXX	8 (start)	10

Coding (Stack)

Given a stack with at least two elements, implement addAndRestack, which pops 2 elements, adds them together, and pushes the sum back onto the stack.

Write a sumUntil(int stop) function that call addAndRestack until a stopping value is found. In that case, leave the stopping value on the stack, and push the sum on top of it.

- Let's clarify this - we will call addAndRestack until stack.peek() == stop

Pseudocode:

```
addAndRestack(stack):
  X = stack.pop()
  Y = stack.pop()
  stack.push(X + Y)

sumUntil(int stop, stack):
 while(stack.peek() != stop) {
   addAndRestack(stack)
 }
```

Code (Stack)

You are given a headed Singly-linked list, and nodes a and b.

Describe a function called exchange(Node<> a, Node<> b) which exchanges the place of the nodes a.next and b.next. You algorithm may only modify links, and may not allocate new nodes (you must work with the nodes you have, only changing the next values)

```
exchange(a, b):
  If (a == b): return // There's nothing to exchange
  If (a.next == b): // Then we are exchanging 2 adjacent nodes
    last = b.next.next
    b.next.next = b // Careful what order you call these in!
    a.next = b.next
    b.next = last // can't use b.next.next because that's been changed
  If (b.next == a):
    last = a.next.next
    a.next.next = a
    b.next = a.next
    a.next = last
  else:
    temp = a.next.next
    a.next.next = b.next.next
    b.next.next = temp

    temp = a.next
    a.next = b.next
    b.next = temp // Store where it used to point!
```

What is the Big-Oh running time of your algorithm?

O(1)

Array Scheme

You have an array of arrays.

int[][] arr.

The array is of length n, and arr[i] is always of length 2^i

Describe a scheme to store this entire data structure in a single array, and how you would access arr[i][j] in this single array.

The answer to this is a bit messier than you can expect on the real exam, but if you understand this answer, you should be able to solve the array scheme problem on the exam

This means that the lengths of each subarray are going to be:
1, 2, 4, 8, 16, etc all the way to 2^(n-1)

So the total number of entries is going to be 1 + 2 + 4 + 8 + …
That's (2^n) - 1 entries

So, we create a new array with length (2^n) - 1

Now we are given (i, j) in our old array. What index k does that correspond with in our new array? It helps to draw a picture:

0
1–2
3–4–5–6
7–8–9–10–11–12–13–14
15–16–17–18–19–20–21–22–23–24–25–26–27–28–29

Note that the first number in every row is always 1 less than a power of 2:

First element of row i = (2^i) - 1

=> Therefore, if we're given an index k, in order to find what row it is in, we need to find the largest value of row i that points to an index < k

```
findRow(k):
  i = 0
  while((1 << i) - 1 < k): // Note that (1 << i) is a bit shift, so it's a clean way to write 2^i
    i = i + 1
  Return i - 1 // Careful not to cause off-by-1-errors here. Go through some examples!
```

Once you find the row, finding the column is easy

```
findColumn(k, i):
  // Just subtract off the first element of the row
  return (k - (1 << i) - 1);
```

So, to translate between k and (i, j), we get:

```
i = findRow(k)
j = findColumn(k, i)
```

And to go from (i, j) to k, you can do:

```
(1 << i) - 1 + j
```