Final Exam Review

Array

1.

```
Random random = new Random();
int[] array = new int[10];
for (int i=0; i<array.length; i++) {
	array[i] = random.nextInt(100);
}

public void remove(int[] array, int indexToRemove) {
	for (int i=0; i<array.length; i++) {
		if (i > indexToRemove) {
			array[i-1] = array[i];
		}
	}
}
```

2.

```
public void insert_2d(int[][] array, int i, int j, int value) {
	array[i][j] = value;
}
```

Linked List

```
public class Node {

	Node next;
	int value;

	public Node() {
		this.next = null;
	}

	public Node( int data ) {
		this.next = null;
		this.value = value;
	}
}

public class LinkedList {

	Node head;
	Node tail;

	public LinkedList() {
		this.head = null;
		this.tail = null;
	}
```

```
	public void append(Node node) {
		if (this.head == null) {
			this.head = node;
			this.head.next == this.tail;
		} else {
			this.tail.next = node;
			this.tail = this.tail.next;
		}
	}
}
```

1.

```
Node node1 = new Node(1);
Node node2 = new Node(2);
Node node3 = new Node(3);
Node node4 = new Node(4);

node1.next = node2;
node2.next = node3;
node3.next = node4;
```

2.

```
Node tmpNode = node1;
while (tmpNode != null) {
	System.out.println(tmpNode.value);
	tmpNode = tmpNode.next;
}
```

3.

```
public static void storeNewValue(Node head, int newValue) {
	if (head == null) {
		head.value = newValue;
	} else {
		Node newNode = new Node(newValue);
		Node tmpNode = head;
		while (tmpNode.next != null) {
			tmpNode = tmpNode.next;
		}
		tmpNode.next = newNode;
	}
}
```

Stack

1. & 2.

```
public class Stack {
	int[] array;
	int indexLastValue;
	int capacity;

	public Stack() {
		capacity = 10;
		array = new int[capacity];
		indexLastValue = 0;
	}

	public void insert(int value) {
		if (array.length <= indexLastValue) {
			doubleArrayCapacity();
		}

		array[indexLastValue] = value;
		indexLastValue ++;
	}

	public int remove() {
		if (indexLastValue > 0) {
			indexLastValue  - -;
			return array[indexLastValue];
		}
		return 0;
	}

	public void doubleArrayCapacity() {
		int[] tmpArray = new int[array.length * 2];
		for (int i=0; i<array.length; i++) {
			tmpArray[i] = array[i];
		}
		array = tmpArray;
	}
}
```

Queue

1. & 2.

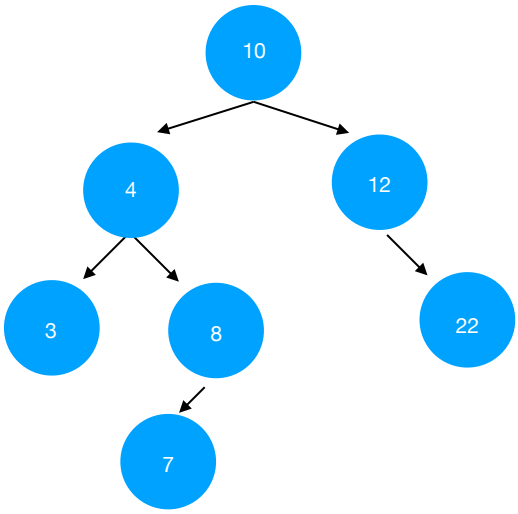
```
public class Queue {
	int[] array;
	int indexLastValue;
	int capacity;

	public Queue() {
		capacity = 10;
		array = new int[capacity];
		indexLastValue = 0;
	}

	public void insert(int value) {
		if (array.length <= indexLastValue) {
			doubleArrayCapacity();
		}

		if (indexLastValue == 0) {
			array[indexLastValue] = value;
		} else {
			for (int i=indexLastValue; i>0; i—) {
				array[i] = array[i-1];
			}
			array[0] = value;
		}
		indexLastValue ++;
	}

	public int remove() {
		if (indexLastValue > 0) {
			indexLastValue  - -;
			return array[indexLastValue];
		}
		return 0;
	}

	public void doubleArrayCapacity() {
		int[] tmpArray = new int[array.length * 2];
		for (int i=0; i<array.length; i++) {
			tmpArray[i] = array[i];
		}
		array = tmpArray;
	}
}
```

3.

Add four random numbers (A, B, C, D) in turn to the Queue, then remove 3 numbers.
The order of the values removed is D, C, B.
The value remained is A.

Binary Search Tree

1.



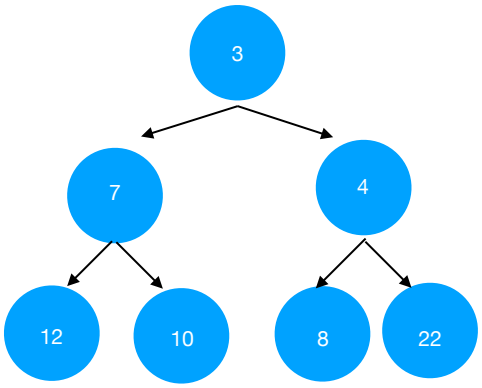
2. & 3.

```
public class Node {
	Node left;
	Node right;
	Node parent;
	int value;

	public Node() {
		left = null;
		right = null;
		parent = null;
	}

	public Node(int value) {
		this.value = value;
		left = null;
		right = null;
		parent = null;
	}
}

public class Tree {
	Node root;
```

```
	public Tree() {
		root = null;
	}

	public void insert(int value) {
		root = insert(value, root);
	}

	public Node insert(int value, Node root) {
		if (root == null) {
			root = new Node(value);
			return root;
		}

		if (value < root.value) {
			root.left = insert(value, root.left);
			root.left.parent = root;
		} else if (value > root.value) {
			root.right = insert(value, root.right);
			root.right.parent = root;
		}
		return root;
	}

	public void preorder() {
		preorder(root);
		System.out.println();
	}

	public void preorder(Node root) {
		if (root != null) {
			System.out.print(root.value + “, ”);
			preorder(root.left);
			preorder(root.right);
		}
	}
}
```

Heap

1.



2. & 3.

```
public class Heap {
	int[] heap;
	int size;
	int capacity;

	public Heap () {
		capacity = 10;
		heap = new int[capacity];
		size = 0;
	}

	public void insert (int value) {
		if (size == (heap.length - 1)) {
			doubleHeapCapacity();
		}

		size += 1;
		int pos = size;

		while (pos > 1 && value < heap[pos/2]) {
			heap[pos] = heap[pos/2];
			pos = pos/2;
		}

		heap[pos] = value;
	}

	public void doubleHeapCapacity() {
		int[] tmpHeap = new int[heap.length * 2];
		for (int i=0; i<heap.length; i++) {
			tmpHeap[i] = heap[i];
		}
		heap = tmpHeap;
	}

	public void print () {
		System.out.println();
```

```
                for (int i=1; i<=size; i++) {
                        System.out.print(heap[i] + " ");
                }
                System.out.println();
        }
}
```