# CSCE146 – Practice Exam (Midterm 2)

CSCE146 F2017 SI | Midterm #2 | JJ Shepphard's class

## Asymptotics

1. Sort the Big O times in Bounding order.

$O(n)$ $O(n^2)$ $O(n^2lg(n))$ $O(n^3)$ $O(1)$ $O(n!)$ $O(n^n)$ $O(lg(n))$ $O(2^n)$

**$O(1)$ $O(lg\ n)$ $O(n)$ $O(n^2)$ $O(n^2lg\ n)$ $O(n^3)$ $O(2^n)$ $O(n!)$ $O(n^n)$**

2. List the Big O times (Worst-case) of the following algorithms

Binary search, Merge Sort, Quick Sort, Insertion Sort, Bubble Sort, Selection Sort, Binary Search Tree Insertion, Tower of Hanoi, Travelling Sales Person

**Binary Search - O(lg n)**

**Merge sort - O(n lg n)**

**Quick sort - O(n^2)**

**Insertion Sort - O(n^2)**

**Bubble Sort - O(n^2)**

**Selection Sort - O(n^2)**

**BST Insertion - O(n) if tree is balanced, then O(lg n)**

**Tower of Hanoi - O(2^n)**

**Travelling Salesman - O(n!)**

**HeapSort - O(n lg n)**

## Java Code

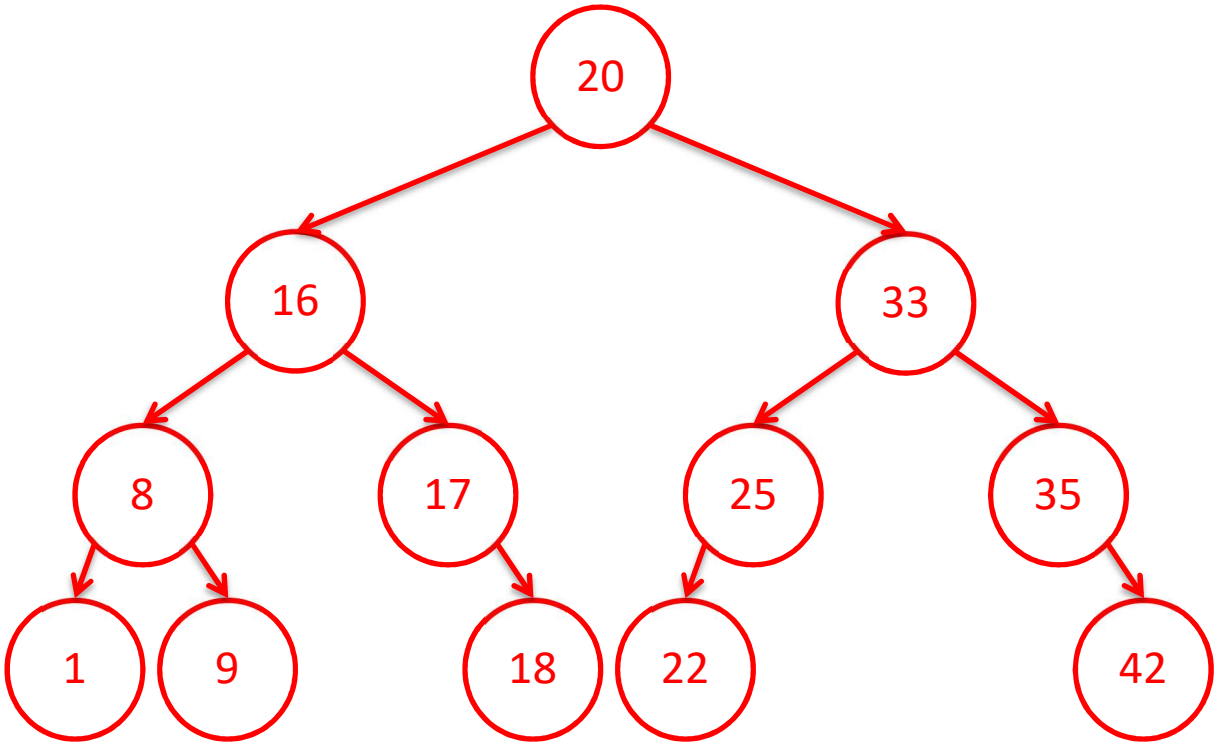
3. Write a Method for Binary Search

```
Public static boolean binarySearch(int[] a, int value, int minIndex, int maxIndex) {
        Int midIndex = (maxIndex + minIndex) / 2;
        If (minIndex > maxIndex) {//Recursion Halt Condition
            Return false;
        }
```

```
            If (a[midIndex] == value) return true; //Found case
            If (value > a[midIndex]) return binarySearch(a, value, midIndex +
1, maxIndex);
            Else Return binarySearch(a, val, minIndex, midIndex - 1);
      }
```

## Binary Search Trees

4. Remove 28 from this BST. Show end result.




5. Show Pre-order, In-order, post-order and breadth-order traversals of this tree

   **Pre: 20 16 8 1 9 17 18 28 25 22 35 33 42**

   **In: 1 8 9 16 17 18 20 22 25 28 33 35 42**

   **Post: 1 9 8 18 17 16 22 25 33 42 35 28 20**

   **Breadth: 20 16 28 8 17 25 35 1 9 18 22 33 42**

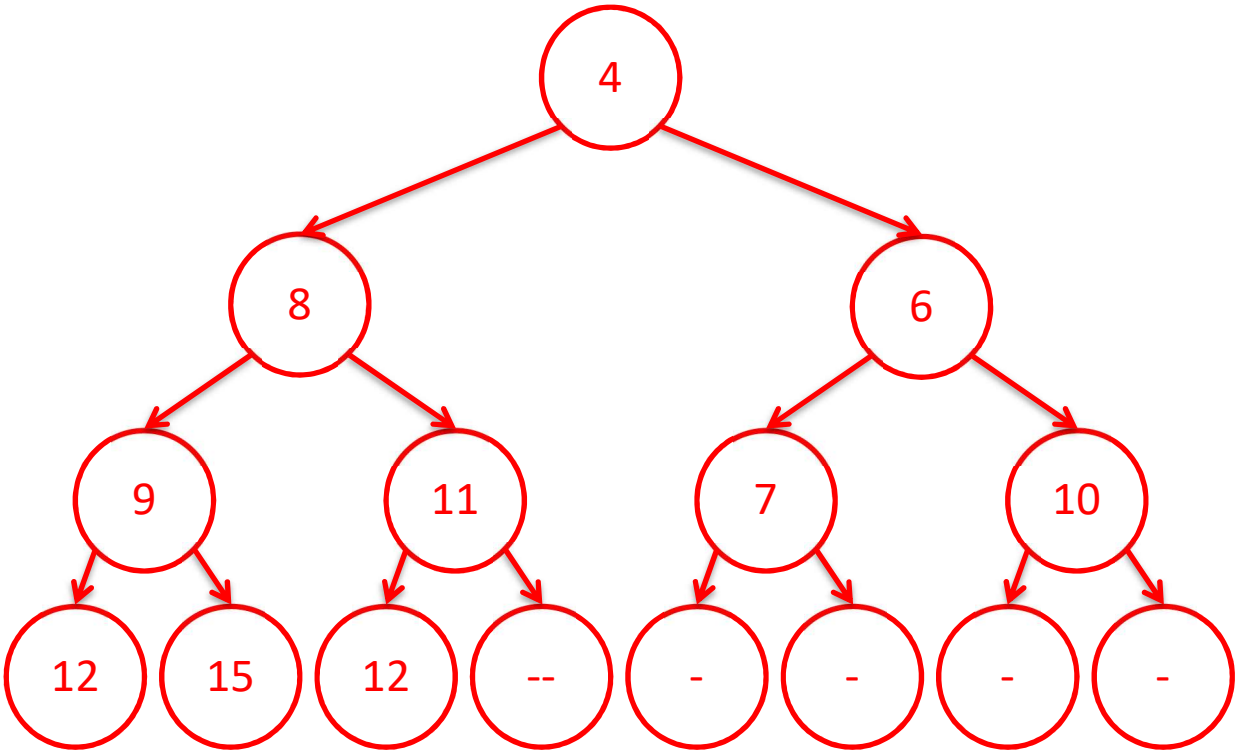
## Heaps

6. Write insert method for a heap

```
public void insert(T value) {
```

```
    if (lastIndex >= heap.length) return; //Heap is full
    heap[lastIndex] = value;
    bubbleUp();
    lastIndex++;
  }
  public void bubbleUp() {
    int index = lastIndex;
    while (index > 0 && heap[(index - 1) / 2].compareTo(heap[index]) <
0) {
      //Child was greater than parent, so swap
      T temp = heap[(index - 1) / 2];
      heap[(index - 1) / 2] = heap[index];
      heap[index] = temp;
      index = (index - 1) / 2;
    }
  }
```

7. Remove from the Min Heap and show end result.



Using the array implementation of a min heap, show the array after inserting 7

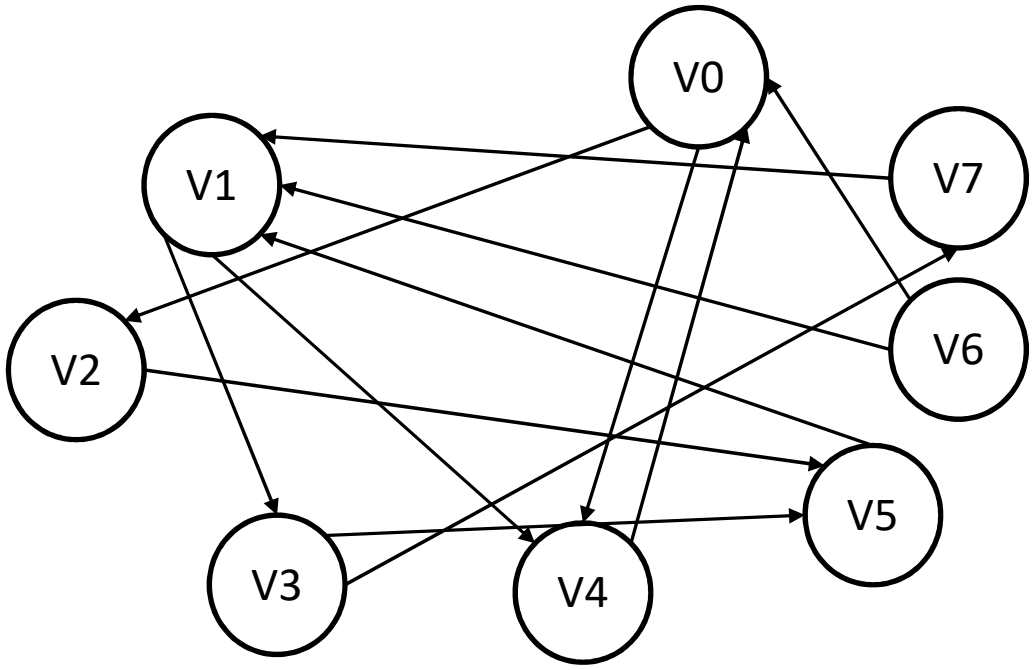
| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| Value | 4 | 5 | 11 | 8 | 6 | 16 | 20 |

| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| Value | **4** | **5** | **11** | **7** | **6** | **16** | **20** | **8** |

8. Mention Heapsort

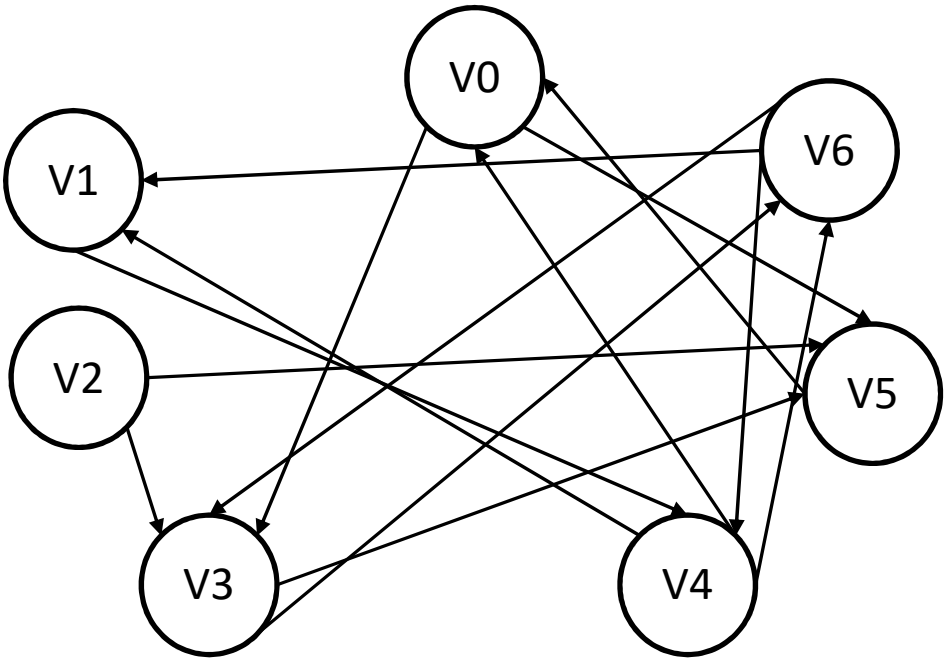
## Graphs

9. Talk about if Graphs are trees
10. For the Following Graphs:
    - Show an Adjacency Matrix (Row is From, Column is To) **-> See graphs.txt for Adjacency Matrices**
    - Show the DFS and BFS Traversals



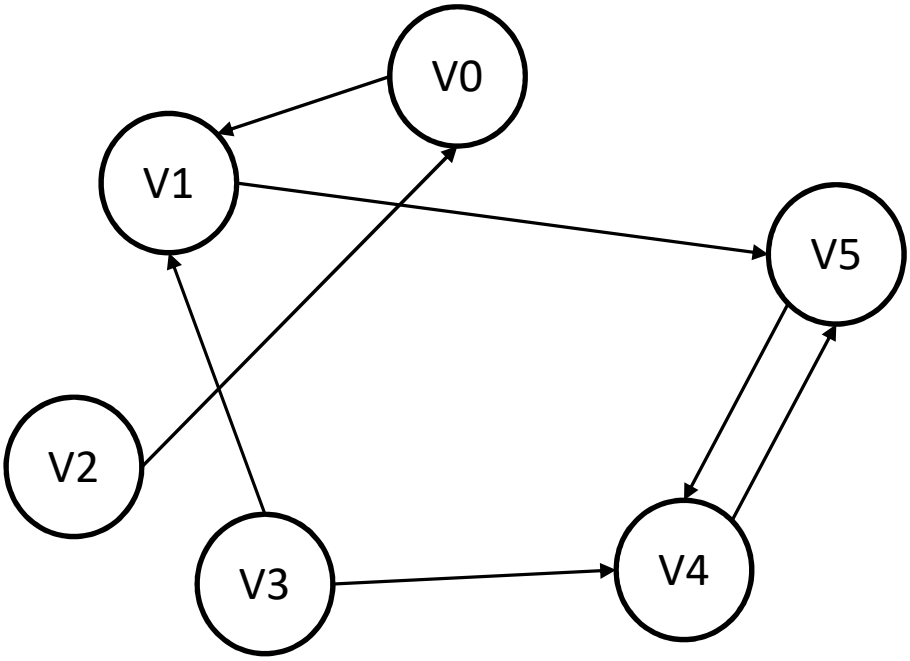

**DFS(v0):v0, v2, v5, v1, v3, v7, v4**

**BFS(v0): v0, v2, v4, v5, v1, v3, v7**




**DFS(v0): v0, v3, v6, v1, v4, v5**

**BFS(v0): v0, v3, v5, v6, v1, v4**




**DFS(v0): v0, v1, v5, v4**

**BFS(v0): v0, v1, v5, v4**