CSCE146 – Practice Final Exam

CSCE146 F2017 SI | Final Exam | JJ Sheppphard's class

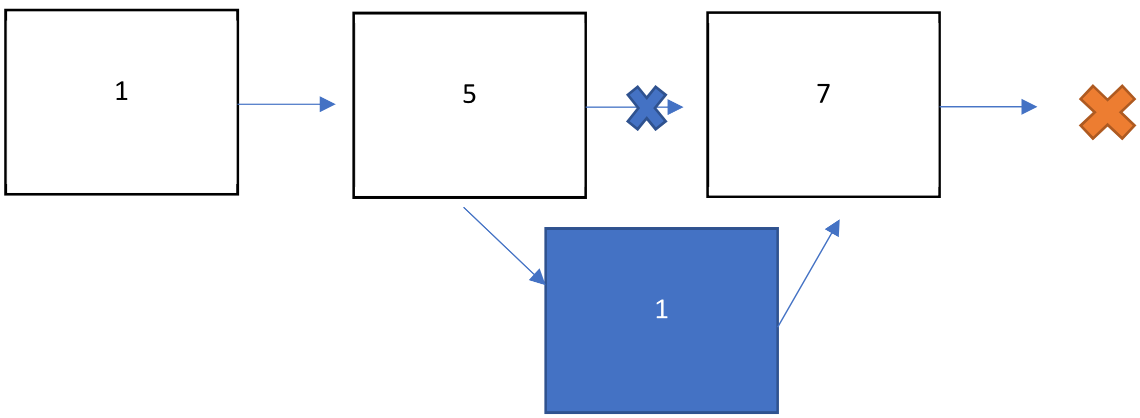
Linked Lists

Know how to write code to find, delete, and insert Nodes

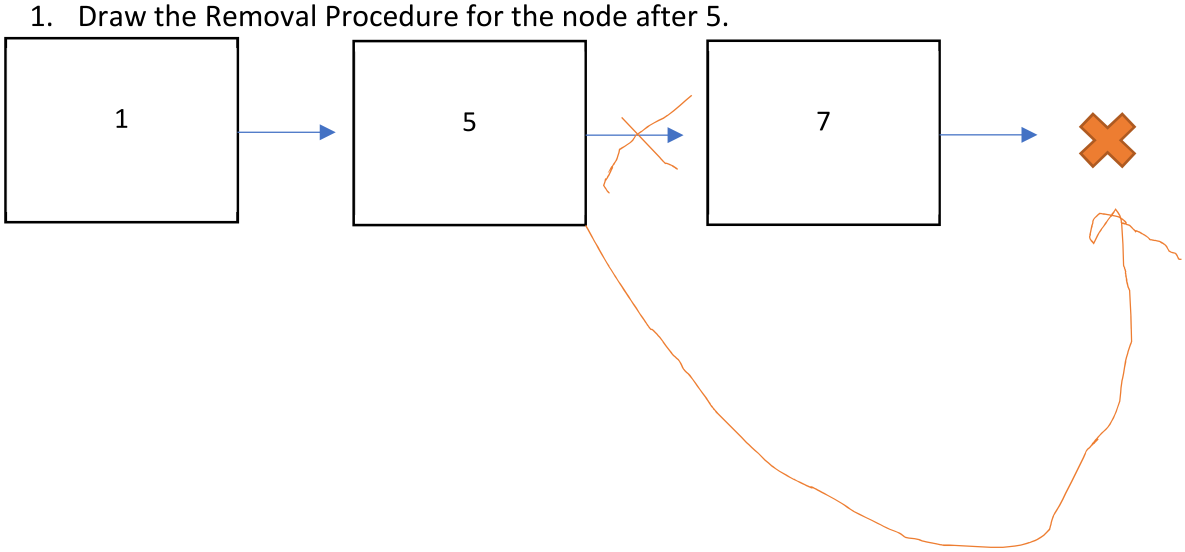
1. List a few Advantages and Disadvantages of using a Linked List over an Array.

Advantage: Resizable. Disadvantage: Slow Access

2. Draw the Insertion Procedure for adding a node after the node containing 5



1. Draw the Removal Procedure for the node after 5.



Queues

Know how to write code to Enqueue, Dequeue and Peek in a Queue

2. Draw the Queue after each Operation

Head						
5	4	8				

Enqueue 3

Head						
5	4	8	3			

Dequeue 3 times

Head						
3						

Enqueue 6 and 24

Head						
3	6	24				

Dequeue 2 times

Head						
24						

3. What will the code snippet print out?

```
Queue<Integer> q = new
LinkedQueue<Integer>();
//Assume that this Queue uses
enqueue(), dequeue(), and peek()

for (int i = 5; i >= -5; i--) {
  q.enqueue(i);
}
for (int i = 3; i < 6; i++) {

System.out.println(q.dequeue());
}
for (int i : q) {
  System.out.println(q);
}
```

5
4
3
2
1
…
-4
-5

Stacks

Know how to code Push, Pop, and Peek

4. What will the Code Snippet Print out?

```
Stack<Integer> s = new
LinkedStack<Integer>();
//Assume that this Stack uses
pop(), push(), and peek()
```

-5
-4
-3
-2
…

```
for (int i = 5; i >= -5; i--) {      4
  s.enqueue(i);                      5
}
for (int i = 3; i < 6; i++) {

System.out.println(s.dequeue());
}
for (int i : s) {
  System.out.println(s);
}
```

5. Draw the Stack after each Operation.

Head						
5	4	8				

Push 3

Head						
3	5	4	8			

Pop 3 times

Head						
8						

Push 6 and 24

Head						
24	6	8				

Pop 2 times

Head						
8						

Recursion

6. What data Structure can be used to illustrate Recursion?

Stacks

7. What does this code do?

```
public static int f(int a) {
  if (a <= 1) return 1;
  return f(a - 1) + a;
}
```

Returns the Triangular Number of f(a) {a + (a-1) + … + 2 + 1}

Searching and Sorting

Array: {45,23,12,79,36,42,10}

8. Perform Mergesort on the Given Array

{45, 23, 12, 79} {36, 42, 10}

{45, 23} {12, 79} {36, 42} {10}

{45} {23} {12} {79} {36} {42} {10}

{23, 45} {12, 79} {36, 42} {10}

{12, 23, 45, 79} {10, 36, 42}

{10, 12, 23, 36, 42, 45, 79}

9. Perform a Binary Search for 45 for the given array (After it has been sorted)

{10, 12, 23, 36, 42, 45, 79}

{10, 12, 23, **36**, 42, 45, 79}

{10, 12, 23, 36} {42, **45**, 79}

Asymptotics

10. 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)

11. 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)

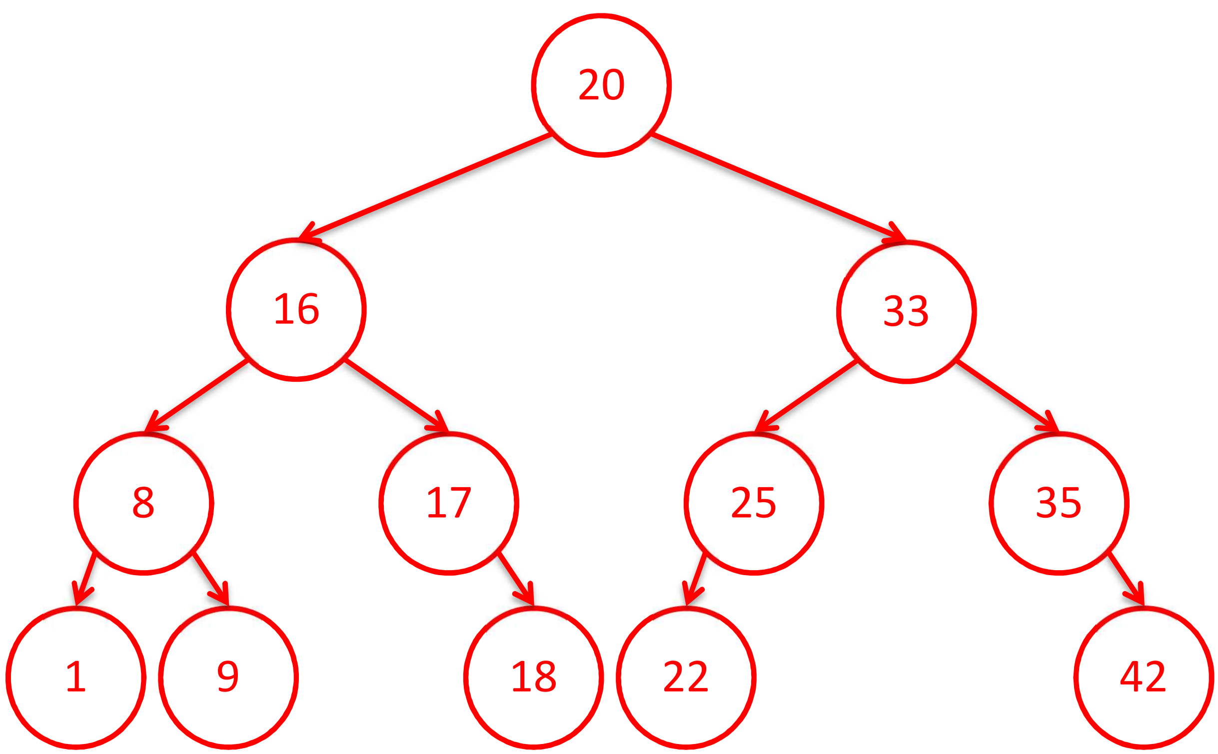
Java Code

12. 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

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



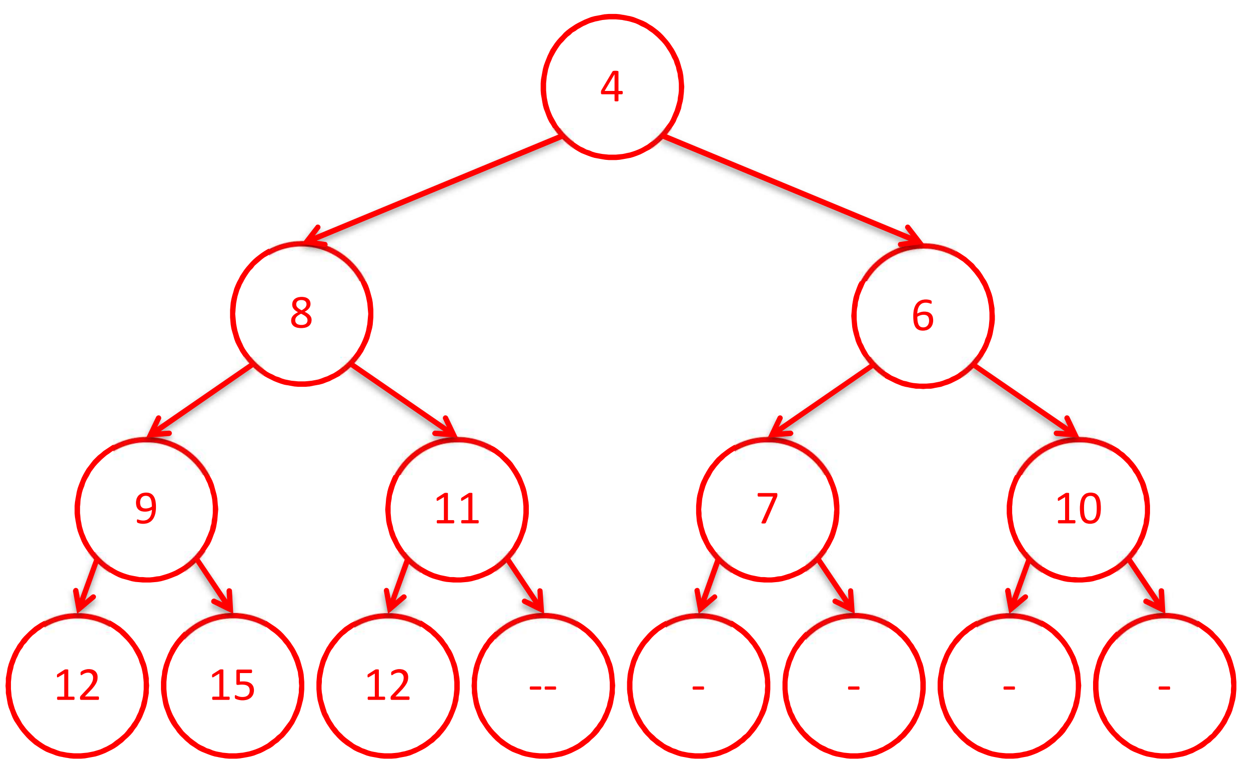
13. Show Pre-order, In-order, post-order and breadth-order traversals of this tree
14. **Pre: 20 16 8 1 9 17 18 28 25 22 35 33 42**
15. **In: 1 8 9 16 17 18 20 22 25 28 33 35 42**
16. **Post: 1 9 8 18 17 16 22 25 33 42 35 28 20**
17. **Breadth: 20 16 28 8 17 25 35 1 9 18 22 33 42**

Heaps

18. 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;
    }
  }
```

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



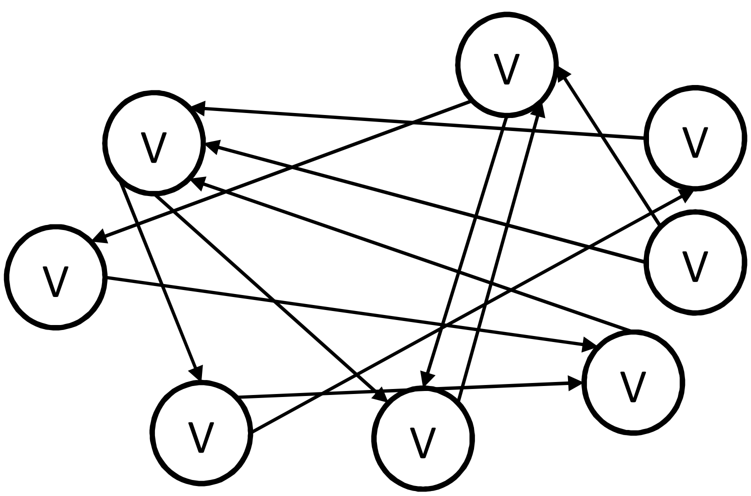
19. Using the array implementation of a min heap, show the array after inserting 7

Index	0	1	2	3	4	5	6	7
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**

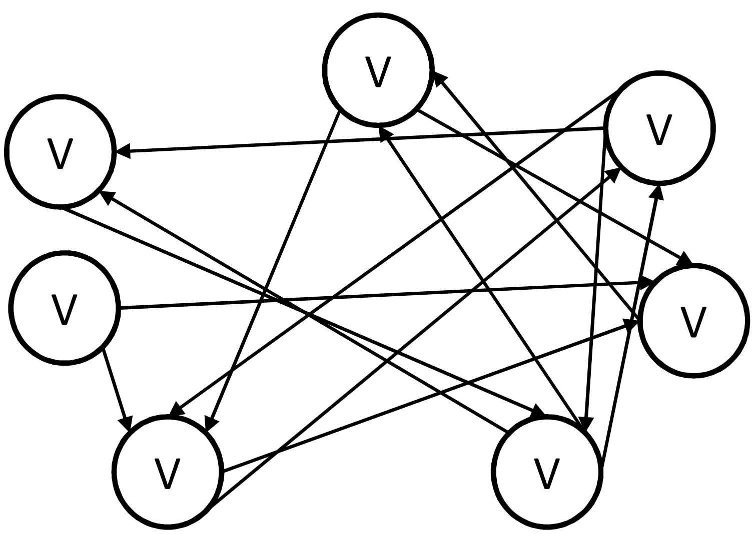
Graphs

20. Talk about if Graphs are trees
21. For the Following Graphs:
    - Show an Adjacency Matrix (Row is From, Column is To)
    - Show the DFS and BFS Traversals



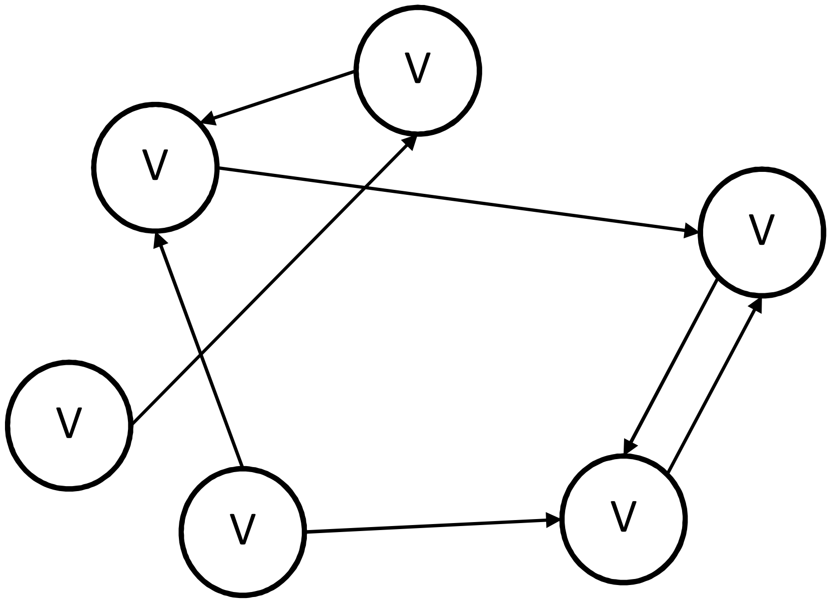
DFS(v0):v0, v2, v5, v1, v3, v7, v4

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

Hash Tables

22. Put the following Tuples in a Hash Table, where the first value is the key and the second is the value.

{(1,"a"), (2,"b"), (2,"g"), (4,"z)}