# **CSCE146 – Practice Final Exam**

CSCE146 F2017 SI | Final Exam | JJ Shepphard’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.
2. Draw the Insertion Procedure for adding a node after the node containing 5

7

5

1

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

7

5

1

**Queues**

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

1. Draw the Queue after each Operation

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Head |  |  |  |  |  |  |
| 5 | 4 | 8 |  |  |  |  |

Enqueue 3

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Head |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

Dequeue 3 times

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Head |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

Enqueue 6 and 24

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Head |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

Dequeue 2 times

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Head |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

1. 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);  } |  |

**Stacks**

Know how to code Push, Pop, and Peek

1. What will the Code Snippet Print out?

|  |  |
| --- | --- |
| Stack<Integer> s = new LinkedStack<Integer>();  //Assume that this Stack uses pop(), push(), and peek()  for (int i = 5; i >= -5; i--) {  s.enqueue(i);  }  for (int i = 3; i < 6; i++) {  System.out.println(s.dequeue());  }  for (int i : s) {  System.out.println(s);  } |  |

1. Draw the Stack after each Operation.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Head |  |  |  |  |  |  |
| 5 | 4 | 8 |  |  |  |  |

Push 3

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Head |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

Pop 3 times

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Head |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

Push 6 and 24

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Head |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

Pop 2 times

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Head |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

**Recursion**

1. What data Structure can be used to illustrate Recursion?
2. What does this code do?

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

**Searching and Sorting**

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

1. Perform Mergesort on the Given Array
2. Perform a Binary Search for 45 for the given array (After it has been sorted)

**Asymptotics**

1. Sort the Big O times in Bounding order.

O(n) O(n2) O(n2lg(n)) O(n3) O(1) O(n!) O(nn) O(lg(n)) O(2n)

1. 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

**Java Code**

1. Write a Method for Binary Search

public static Boolean binarySearch(int[] a, int value

**Binary Search Trees**

1. Remove 28 from this BST. Show end result.
2. Show Pre-order, In-order, post-order and breadth-order traversals of this tree

**Heaps**

1. Write insert method for a heap

public void insert(int a) {

1. Remove from the Min Heap and show end result.
2. 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 |
| Value |  |  |  |  |  |  |  |

**Graphs**

1. Talk about if Graphs are trees
2. For the Following Graphs:

* Show an Adjacency Matrix (Row is From, Column is To)
* Show the DFS and BFS Traversals

**Hash Tables**

1. 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)}