CSE 373: Practice Final

1 Short Answer

a) Provide two orderings [0,1,2,3,4,5,6,7] that are worst-case for quick sort. Assume that you select the first element as the pivot. Explain why this is the worst-case. You do not have to perform the quicksort

[0,1,2,3,4,5,6,7]
[7,6,5,4,3,2,1,0]

Because the elements are in order, each partition only eliminates one element rather than dividing the data roughly in half. This gives it a n^2 runtime.

b) Explain why a divide-and-conquer approach would or would not be effective in finding an element in an unsorted array.

Find in an unsorted array is $O(n)$, so dividing the ~~work~~ array in half only halves the work. The recurrence for this divide-and-conquer approach would be $T(n) = O(1) + 2T(n/2)$ - which is also $O(n)$, so no benefit is reached.

c) You are given a batch of UW ID numbers. Given that they are all seven digits, and that the first two numbers correspond to the year of the student's entry, propose a method to put these into sorted order. Provide the time complexity of your algorithm and explain why you believe it is fastest.

Perform a radix sort, the runtime will be $O(7 \cdot (n+10))$ from the analysis. Since there are many more than 70 students, this is $O(n)$ time and better than any comparison sort

d) Provide and explain the two types of locality relevant to caching and memory accesses.

Temporal locality - items recently accessed are likely to be accessed again. Move them into cache. Alternatively, items which have not been accessed in a while can be taken out of cache

Spatial locality - if an ~~ele~~ index of memory is accessed, memory near it is likely to be accessed. Bring elements into ~~memor~~ cache in chunks. For memory and the disk, this a chunk is called a page

e) For each of the following sorts, explain their best-case and worst case runtime, whether or not they are stable (in the method given in class), whether they can be halted to provide the top k elements and whether or not they are done in place.

Insertion sort:

BC- $O(n)$ WC- $O(n^2)$

stable, in-place, not-interruptible

Selection sort:

BC- $O(n^2)$ WC- $O(n^2)$

stable, in-place, interruptible

Merge sort:

BC- $O(n \lg n)$ WC- $O(n \lg n)$

Stable, not in place, not-interruptible

Quick sort:

BC- $O(n \lg n)$ WC- $O(n^2)$

not stable, in place, not interruptible

Heap sort:

BC- $O(n \lg n)$ WC- $O(n \lg n)$

not stable, in place, interruptible

f) For the Uptree data structure, show the worst-case runtime for find() with and without the weighted union optimization and with and without the path compression optimization. In all, you should provide four worst-case runtimes.

Path Compression \ Weighted union	y	n
y	O(lg n)	O(n)
n	O(lg n)	O(n)

g) Provide pseudocode for an iterator that returns all of the vertices that are exactly two edges away from an input node in an undirected, unweighted graph.

From the input node, input all of its neighbors into an array.

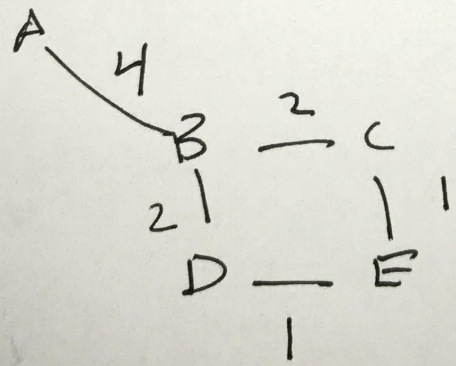
For each of the neighbors, output any nodes connected to it that are not in the array and not the input node.

h) Explain the difference between primary and secondary clustering in hash tables. Which types of clustering are relevant for which probing techniques?

Primary clustering occurs with linear probing and worsens runtimes when large contiguous blocks of memory get filled and any new adds have to linearly search through the block to find the free space at the end.

Secondary clustering occurs with quadratic probing. Some numbers, when they mod the quadratic values 1, 4, 9, 16... do not completely distribute

i) Draw an unweighted graph of at least five vertices which has two unique minimum spanning trees.



#1: AB, BC, CE, DE

#2 AB, BD, CE, DE

2 Big O notation

For the following functions, determine the tightest bigO upper bound in terms of n. Write your answers on the line provided.

a)
```
void f1(int n) {
    int i = 1
    while(i < n^4){
        j = n;
        while (j > 1) {
            j = j / 2;
        }
        i = i + n
    }
}
```

lg n } n^3

O($n^3 \lg n$)

b)
```
void f2(int n) {
    for(int i=0; i < n; i++) {
        for(int j=0; j < 10; j++) {
            for(int k=0; k < n; k++) {
                for(int m=0; m < 10; m++) {
                    System.out.println("!");
    }   }   }   }
}
```

n · O(1) · n · O(1)

O(n^2)

c)
```
int f3(int n){
    if (n < 10) return n;
    else if(n < 1000) return f3(n-2);
    else return f3(n/4)+f3(n/4);
}
```

$T(n) = 1 + 2T(\frac{n}{4})$

$= 1 + 2(1 + 2T(\frac{n}{16}))$

$= 1 + 2 + 4T(\frac{n}{16})$

$= \sum_{i=0}^{\log_4 n} 2^i$

O($\sqrt{n}$)

d)
```
int f4(int n){
        if(n < 1000){
             return n;
        }
        if (n%5 == 0){
             return f4(n/2)+f4(n/2)+f4(n/2);
        } else {
             return f4(n-2);
  }
```

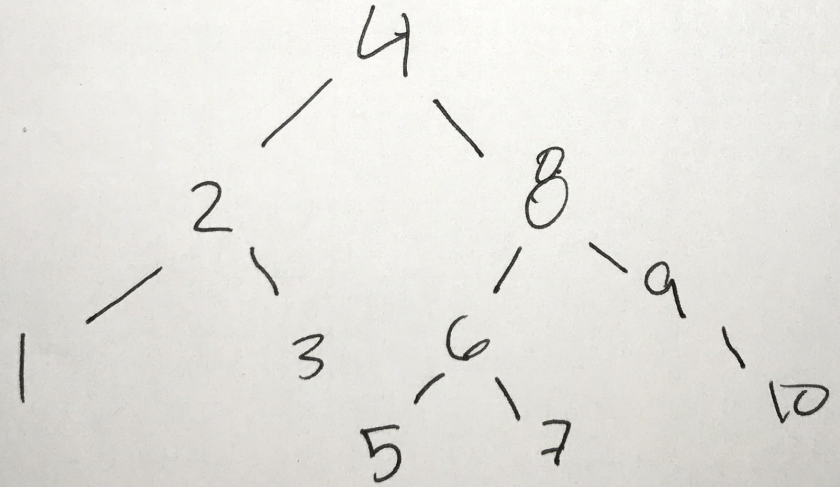
$O(n^{\lg_2 3})$

3 AVL insertions

Show an AVL tree after performing the following inserts:

$$[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]$$

You only need to show the final result, but showing intermediate steps may earn you partial credit if a mistake is made.



Provide the pre-order, in-order, post-order and BFS traversal for the AVL tree you produced

Pre-order:

4, 2, 1, 3, 8, 6, 5, 7, 9, 10

In-order:

1, 2, 3, 4, 5, 6, 7, 8, 9, 10

Post-order:

1, 3, 2, 5, 7, 6, 10, 9, 8, 4

Breadth-first Search:

4, 2, 8, 1, 3, 6, 9, 5, 7, 10

4 Debugging

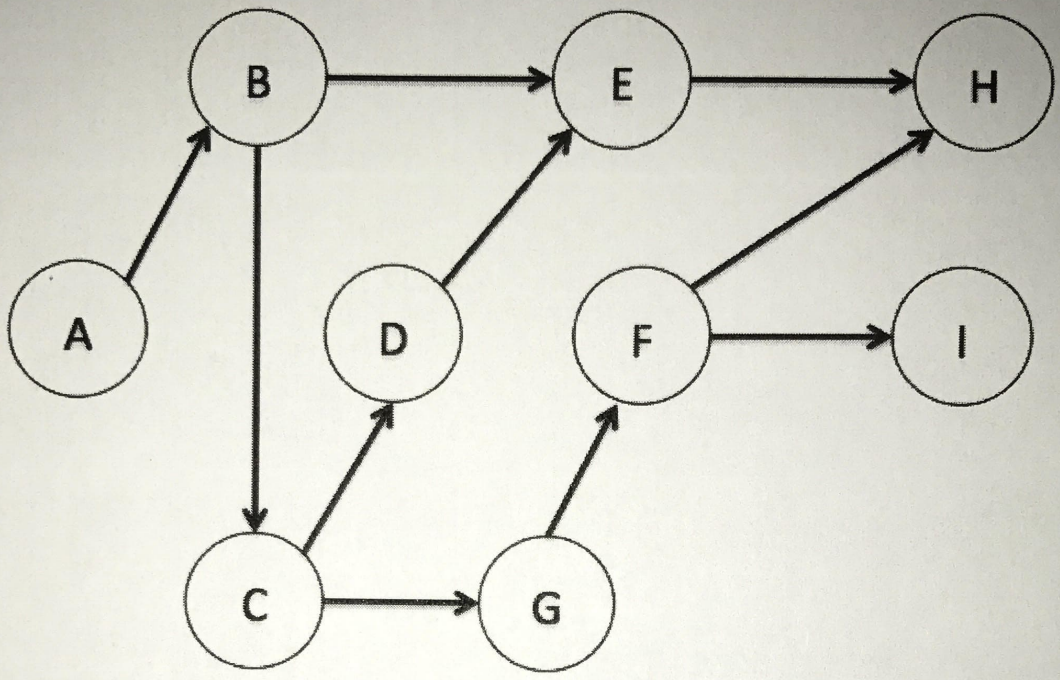
Debug the following Java code which attempts to implement a Queue. Circle any incorrect areas and provide the correct portion. Syntax is not important.
Additionally, provide a sequence of enqueues and dequeues which would get this TestQueue to exhibit unexpected behavior.

```
public class TestQueue{
    String[] data;
    int fp;
    int lp;
    public TestQueue(){
        data = new String[10];
        fp = 0;
        lp = 0;
    }
    public void enqueue(String toInput) {
        data[lp] = toInput;
        lp = (lp + 1)%data.length;
    }
    public String dequeue(){
        if(fp!=lp){
            String toRet = data[fp];
            fp = (fp + 1)%data.length;
            return toRet;
        }
        return null;
    }
    public String front(){
        return data[fp];
    }
}
```

no resize, looping around could overwrite

5 Topological Sort

Provide two topological orderings for the following graph.



A B C D E G F H I

A B C G D E F I H

Explain how you could modify the topological sort algorithm to find cycles in directed graphs.

Topological orderings are only possible in acyclic graphs, if no elements at a certain point have in-degree zero then there must be a cycle.

10 Algorithm Design

Design an algorithm, which given two integers (i, j) greater than two, determines whether they have any common factors. Recall that if two numbers are both prime and not equal to each other, then they have no common factors. Also, if a number k is composite (not-prime) then it must have one factor that is at most $\sqrt{k}$.

Provide pseudocode for this algorithm and then explain the runtime of your algorithm. Additionally, if it needs more than a constant amount of additional memory, explain this memory usage.

```
if i > j
    max = i
else
    max = j
for (k from 2 to ⌈√max⌉)
    check if i % k = j mod k = 0
        if so return true
if no factors were found
    return false
```

Checking if i and j are divisible by k takes constant time. This constant time operation will be performed $\sqrt{i} - 1$ or $\sqrt{j} - 1$ ~~whatever~~ whichever is larger. This is $O(\sqrt{i} + \sqrt{j})$. It can be performed with only a constant amount of memory.

Euclid's algorithm is also acceptable but not ~~would~~ would be difficult.