Homework - Data Structures and Complexity

Due: 03-13-2020 (Friday @ 3:30 p.m.)

- Given a collection of algorithms that runs on **O(1), O(n log n), O(n), O(n^2), O(log n), O(n!)**, order the algorithms from fastest to slowest.
  Solution: O(1), O(log n), O(n), O(n log n), O(n^2), O(n!)

- Suppose that the complexity of an algorithm is O(n^2). Suppose that the program that uses the algorithm run in 10 seconds for a data set of size n. If the data size is doubled, how long will it take (approximately) to run the program?
  Solution: 40 seconds
  Reasoning: First, plug in the factor that the data size was increased by (2) into the complexity (n^2). You multiply the time (10 seconds) by that result (4) to get the answer.

- Complexities : **O(1), O(n log n), O(n), O(n^2), O(log n), O(n!) , O(h)**
- Finding the max element in an unordered stack would require **O(n)** operations.
- Finding the min element in an unordered queue would require **O(n)** operations.
- Finding some element in some Binary Search Tree would require **O(h)** operations.
- Finding some element in a balanced Binary Search Tree would require **O(log n)** operations.
- Finding some element in an ordered linked list would require **O(n)** operations (worst case).
- Finding some element in an ordered linked list would require **O(n/2)** operations (average case).
- Finding some element in an unordered linked list would require **O(n)** operations (worst case).

1. For each of the following, count the number of operations where some_statement is executed based on the loops

```
//A
for (int I = 0; I < n; I++)
    for (int j = 1; j < n; j++)
        {some_statement;}
```

Solution: n^2 operations.
Reasoning: The outer loop will run n times, and for every time the outer loop runs once the inner loop runs n times. This produces n X n operations, which is n^2 operations.

```
//B
for (int I = 0; I < n; I +=2)
    for (int j = 1; j < n; j++)
        {some_statement;}
```

Solution: n^2 operations.
Reasoning: The outer loop will run n/2 times, and every time the outer loop runs once the inner loop will run n times. The solution is n^2 because this produces $n^2/2$, and when considering complexities, all constants are disregarded, so the solution is n^2.

```
//C
for (int j = 1 ; j < n ; j *= 2)
    for (int I = 1; i<n; i++)
        {some_statement;}
```

Solution: n log(n) operations.
Reasoning: The outer loop will run log(n) times, and every time the outer loop runs once the inner loop will run n times. This produces n log(n) operations total.

1. At most, how many comparisons are required to search a sorted vector of 1023 elements using the binary search algorithm?
   Solution: It would take, at most, 10 comparisons.
   Reasoning: Dakota and I wrote a replit and actually tried this with a sorted array of 1023 elements, and the result was 10 comparisons.

In each of the following examples, please choose the best data structure(s).

- Options are: **Array, Linked Lists, Stack, Queues, Trees, Graphs, Sets, Hash Tables**.
- Note that there may not be one clear answer.
- You have to store social network "feeds". You do not know the size, and things may need to be dynamically added.
  Solution: List-based Priority Queue
  Reasoning: I chose this implementation because I wanted to make priority be the level of activity on a post (node). I would do it where likes and reactions incremented priority by 1 and comments on posts would increment priority by 2.
- You need to store undo/redo operations in a word processor.
  Solution: Stack
  Reasoning: To access the most recent version of the document, just pop one edit off the top of the stack.
- You need to evaluate an expression (i.e., parse).
  Solution: Doubly Linked List
  Reasoning: I would have a search algorithm that found where the furthest open parenthesis is, and then using pointers to locate the close parenthesis, I would work my way from the inside moving outward and editing the list as it goes.
- You need to store the friendship information on a social networking site. I.e., who is friends with who.
  Solution: Graph
  Reasoning: Nodes are people, and edges are relationships.
- You need to store an image (1000 by 1000 pixels) as a bitmap.
  Solution: 2D Array

Reasoning: Quick and efficient.

- To implement printer spooler so that jobs can be printed in the order of their arrival.
  Solution: Queue
  Reasoning: Processes jobs in the order of their arrival (FIFO).

- To implement back functionality in the internet browser.
  Solution: Stack
  Reasoning: So that the last entry is easily retrieved by popping off the top.

- To store the possible moves in a chess game.
  Solution: Graph
  Reasoning: Each move respresents a node, and the graph is fully connected. However, if the player were to lose a piece, then moves that go down that path for that piece could not be travelled on, and therefore the same for all other impossible moves at any given point.

- To store a set of fixed key words which are referenced very frequently.
  Solution: Priority Queue
  Reasoning: Whenever a word is accessed, increment its priority and reorder the queue.

- To store the customer order information in a drive-in burger place. (Customers keep on coming and they have to get their correct food at the payment/food collection window.)
  Solution: Queue
  Reasoning: FIFO.

- To store the genealogy information of biological species.
  Solution: Tree
  Reasoning: Trees are good for showing descendants and parent species.

- To store an alphabetized list of names in order to look up quickly.
  Solution: Array
  Reasoning: I would use a binary search to find names.

Deliverables

- Edit this file and add your answers using markdown!
- Create a folder called `H03` in your assignments folder.
- Put a copy of your markdown file in this folder, and call it README.md.
- Upload to github sometime close to the due date.
- Print out your banner ON ITS OWN PAGE

```
H03
3013
LASTNAME
```

- Print out a hard copy of the file as well. Do not print directly from github. Either use `gitprint` or make it a pdf and print it.
- Make sure you answer thoroughly using complexities where appropriate and/or

explaining your choices etc.