CIS*2520 Mock Exam (Summer 2017)

All questions are used from Courselink via
`https://courselink.uoguelph.ca/d2l/le/content/470592/Home`.

Unit 1 Lists, Stacks and Queues

1. Define the following with respect to software design:

   a. Modularity

   b. Reuse

   c. Maintainability

   d. Coupling

   e. Cohesion

2. Define Test-Driven Development.

3. List ten different programming languages as well as their level of abstraction (Moderate, High, Very High).

   1. _____
   2. _____
   3. _____
   4. _____
   5. _____
   6. _____
   7. _____
   8. _____
   9. _____
   10. _____

4. Write pseudocode for a multiply and a subtract operation for the fraction ADT. What does the integer portion of the Fraction struct represent? List any additional operation(s) that would be required in order to make use of that part of the struct. (Fraction ADT available on Courselink)

5. Write the following three algorithms:

    1. The algorithm for the addToLocation() operation for the List ADT. This operation should add an element to the list at a specific location in the list (identified by a number). Use the same specification format as has been used to describe operations in this lesson. Include all necessary parameters and return values in the signature of your specification. Be sure to include preconditions and postconditions.

    2. The algorithm to delete a node from the nth position of a linked list. The operation should return the deleted data and ensure that the remaining elements of the list are properly connected.

    3. The algorithm for inserting a node in sorted order given an array implementation of a list. The algorithm should take the data as a parameter.

6. Would a double linked list or a single linked list be a better choice for encapsulation in a Stack ADT? Justify your opinion.

7. Write the algorithms for push() and pop() given an array implementation of a stack. Show the stack struct definition as well.

8. Create a design or prototype of a reverse polish calculator program. You should limit operations to + - * and /. Use a stack ADT in your design.

9. What is the Computation complexity of the enqueue() and dequeue() operations expressed in Big O notation?

10. What additional information must be kept track of in order to use a conventional array as a circular buffer? Give the Queue struct that you would use if you were writing a circular buffer queue implementation.

11. Write the algorithm for dequeuing an item from a circular buffer queue.

12. Which of the following statements about queues is untrue?

    a. Queues can have elements inserted at any position in the data structure.
    b. The first element inserted into a queue will be the first element taken out of the queue.
    c. Queues can be found in the real world.
    d. The size of a queue data structure is bounded only by the size of the computer memory.
    e. A queue is somewhat similar to a stack.

13. Which List operation would be most likely to be the one encapsulated if you were writing a remove operation for a queue?

    a. addHead(elementToBeAdded)
    b. length()
    c. removeBack()
    d. removeHead()
    e. insert(position, elementToBeAdded)

14. Given the following queue: A B b E r S T, where A is the front of the queue, what will the queue content be after two remove operations?

    a. A B b E r S T
    b. b E r S T
    c. A B b E r S
    d. T A B b E r
    e. The queue will be empty.

15. Given the following queue: t y 5 8 i 2 d t e, what should a call to length() return?

    a. 9

    b. 8

    c. 10

    d. It will generate an error because of the mixed data types.

    e. 3

Unit 2 Hash Table and Binary Trees

1. If you were implementing a student record system for a small music school with fewer than 200 students, and were using associative arrays as the data structure, how would you implement your associative array? Would you make a different choice if your student record system was for a university?

2. Write a 'to-do list' of the things you would need to do (including research and learning) in order to implement the borrowers' library example. Try to make the list very specific. Use the to-do list to estimate the amount of time it would take you to implement and test the application (round up to the nearest hour).

3. Using the division method, calculate hash values for the following set of keys:

$$\{54, 77, 82, 13, 991, 308, 68, 45, 1001, 73\}$$

   Calculate once using a table size of 11. Calculate a second time using a table size of 12. Do you notice anything about the distributions of the calculated values?

4. The expected search time for Linear probing can be calculated by the following formula:

   i. $O(1 + \frac{1}{1-loadFactor})$ for a successful search.

   ii. $O(1 + \frac{1}{(1-loadFactor)^2})$ for an unsuccessful search.

   Create a chart showing the expected search times (successful and unsuccessful) for the following load factors: .10, .20., .30, .40, .50, .60, .70, .80, .85, .90, .95.
   What do these numbers tell you?

5. Suppose you have two hash functions H1 and H2 where H1(87) = 10, H2(87) = 3 and H1(42)=10, H2(42)=7. Further suppose that the key 87 is inserted into the table first and then the key of 42. Show the sequence of table positions tried when using random probing with a constant of 37 and a table size of 11.

6. The expected time for a search using Double hashing can be calculated by the following formula:

   i. $O(\frac{1}{loadFactor} \times (1 + ln(\frac{1}{1-loadFactr})))$ for a successful search.

   ii. $O(\frac{1}{1-loadFactor})$ for an unsuccessful search.

   Create a chart showing the expected search times (successful and unsuccessful) for the following load factors: .10, .20., .30, .40, .50, .60, .70, .80, .85, .90, .95.
   What do these numbers tell you?

7. The loadFactor of a hash table using separate chaining to resolve collisions is calculated as the number of **keys / the number of chains**. The expected length of search, successful or unsuccessful, of Separate Chaining is $O(1 + loadFactor)$.

   a. Calculate a chart of the predicted search times for the following load factors: .10, .20., .30, .40, .50, .60, .70, .80, .85, .90, .95.

   b. What do these numbers tell you?

c. What factors would you consider when selecting a collision resolution approach for a hash table?

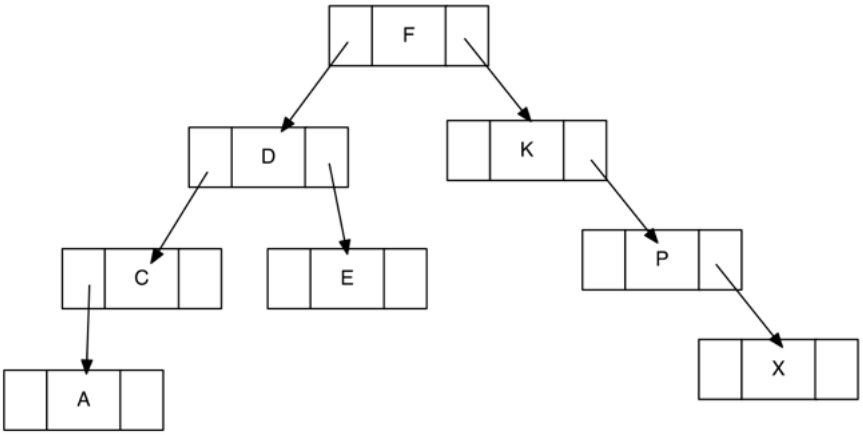
8. The order that nodes are inserted into a tree can greatly affect the structure of the tree. Draw the binary tree that is constructed by adding the seven names in the following order:

    1. Sleepy
    2. Bashful
    3. Doc
    4. Dopey
    5. Sneezy
    6. Happy
    7. Grumpy

9. Trace the algorithms for level-order traversal using a hand drawn tree. What order are the nodes of the tree printed in? Can you tweak the algorithm to reverse the order?

10. One of the most frequent uses of a binary tree is as a search tree. The object is to search through the data to find out if a particular element is part of the data. Write a recursive find procedure for a binary tree. Start with the root node of the tree. Your procedure can be in *C* or in pseudocode. Be sure to pass in the compare function pointer.

11. Give the in-order traversal and the pre-order traversal of the tree shown below.



12. Write the pseudocode or C code for a compare operator. A compare operator must return three distinct values, one for when the first parameter is larger, one for when the second parameter is larger, and one for when the two parameters are equal. The convention is to use 1, 0, and -1 for the three values. Zero for equal values, 1 for the first string being larger, -1 for the second.

13. The delete operation relies on a recursive function to find the minimum value in a subtree. The minimum value will always be the leftmost leaf of the subtree. Write the findMinimum function (in pseudocode or in C).

Unit 3 Heaps and Priority Queues

1. Develop a formula to calculate the position of a parent node in an array-implementation of a heap. The only input information to the formula is the position of the current node.

2. Create the *C* struct for a binary-tree based heap. Your struct should have members for the heap, the last added element, the next position, and function pointers to manipulate void* data.

3. Heaps do not have to be a min/max binary tree. Many variations of heaps are possible. Use internet resources to learn about one of the following different types of heaps: skew heaps, binomial heap, or leftist heap. Write the pseudocode for insert and delete operations for the type of heap you chose.

4. The complexity of insert and delete for a binary heap is $O(log(N))$. Why is that? Create a written explanation, chart, or diagram that explains why $log(N)$ is the complexity for those operations.

5. Consider the implementation of a priority queue using an array, a linked list, and a heap. For each implementation, provide the pseudocode for the insert and removeMax operation. What is the complexity of those operations in each case?

   a. Array Implementation

   b. Linked List Implementation

   c. Heap Implementation

6. Use the Internet to find an additional approach to avoiding starvation in a priority queue, other than the two mentioned in the notes. Describe this third approach.

7. Assume you have a computer that can execute 1 billion instructions per second. How long (in seconds / minutes / days / months / years) would that computer take to execute an algorithm of the complexity give for each of the number of data items shown in the table below.

Algorithm order	10,000 data items	1 million data items	1 billion data items
$O(\log N)$			
$O(N)$			
$O(N \log N)$			
$O(N^2)$			
$O(2^N)$			

8. Create a table showing the count of the primitive operations for the algorithm for adding a node to the end of a singly linked list. The algorithm is given in the Linked List materials for this course. What is the big O complexity for that algorithm?

9. Consider the following three algorithms for determining whether anyone in a room of people has the same first name as a target person. What is the worst case scenario for these algorithms? For each algorithm, how many questions will be asked in the worst case? What is the Big O complexity class for each algorithm?

    1. The target person says their name. All the people with the same name stand up.

    2. The target person approaches each person in the room to ask them their name. If the person approached has the same name as the target person, the search is successful and stops.

    3. The target person asks one person about their name. If person one does not have the same name, the target person asks person one to ask the next person (person two). Person two responds to person one, who tells the target person the answer. If that name doesn't match, the target person asks person one to tell person two to ask person three. Person three responds to person two who responds to person one who tells the target person. This algorithm continues until the last person has been asked.

10. Consider the following four functions:

    - function 1: $O(2^N)$
    - function 2: $O(N^{5/3})$
    - function 3: $O(N \log N)$
    - function 4: $O(N^{\log N})$

    List the four functions in order of increasing computational complexity:

    1. _____
    2. _____
    3. _____
    4. _____

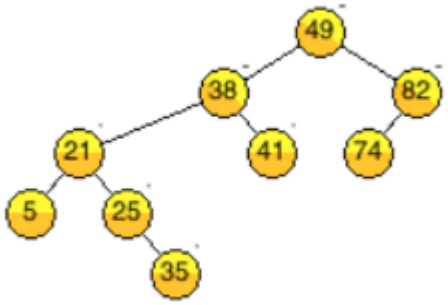
11. Create a table showing the count of primitive operations for a bubble sort. What is the worst case big O complexity class for a bubble sort?

Unit 4 Balanced Trees and Sorting

1. Using a tree graph, create a binary search tree (BST) by inserting the numbers between 1 and 9 into the tree, in order. What do you notice about the tree?

   Using another tree graph, insert the same numbers in the following order 5 3 7 4 2 8 6 9 1. What do you notice about the tree this time?

2. The AVL tree shown in this figure is unbalanced. It needs a right rotation on node 38 to complete the balancing step.



List the steps required to complete the rotation.

3. Write *C* code showing the struct you would implement for a fully abstracted b-tree.

Write the *C* code for the insert function for the b-tree.

4. Describe the key similarities and differences between red-black trees and AVL trees.

5. What is the maximum height of an AVL tree with 9 nodes? Assume that an AVL tree with one node is height 1. Can you find a formula for the maximum height of an AVL tree with N nodes?

6. Design an augmented AVL tree which can quickly compute the number of nodes in any tree or subtree. Discuss changes to the struct for the AVL tree as well as necessary changes to any of the algorithms for the ADT.

7. Which of the following statements is true about Insertion Sort?

   a. It is a comparison Sort.
   b. It is an adaptive Sort.
   c. It has an $O(n \log n)$ complexity for the best case.
   d. It has an $O(n^2)$ complexity for the worst case.
   e. It performs more comparisons than Selection Sort.
   f. It is always outperformed by divide and conquer sorts.
   g. It requires $O(1)$ extra memory.
   h. It is an in-place sort.

8. Which of the following scenarios might be a good situation in which to use insertion sort?

    a. When only alphabetical sort is required.
    b. When you are sorting more than 20 items.
    c. When you are sorting fewer than 20 items.
    d. When you have very little extra memory space.
    e. When you need an extremely fast sort.
    f. When the comparison operator is computationally expensive.
    g. When the list must be sorted as the data is generated, one item at a time.
    h. When only large numbers will be sorted.

9. Which part of the algorithm for insertion sort causes the average case running time?

    a. The assignment of different values into the temporary variable.
    b. The loop that goes through the array one element at a time.
    c. The nested loops that each go through the array one element at a time.
    d. The multiple places where the boolean variable finished is assigned a value.

10. Which of the following statements are true about Selection Sort?

    a. It is $O(n^2)$ for the average case.
    b. It is an adaptive sort.
    c. It requires fewer swaps than many algorithms.
    d. It generally performs worse than insertion sort.
    e. It is sometimes faster than a divide and conquer algorithm.
    f. It is not an in-place sorting algorithm.
    g. It is similar to a heapsort.
    h. It can be written to be a stable sort.

11. Which of the following scenarios might be a good situation in which to use selection sort?

    a. When you are sorting 20 items or fewer.
    b. When you are sorting more than 20 items.
    c. When you know that your data will be partially sorted.
    d. When the algorithm for swapping data values is computationally expensive.
    e. When the data to be sorted is in a linked list.
    f. When you must minimize the amount of auxiliary memory used.
    g. When you are sorting a set of complex data structures.
    h. When your data must be sorted twice, such as by last name and then age.

12. Selection sort has the same computational cost for worst, average and best cases. Which statement below best describes why this is so?

    a. Because comparisons are only made outside the inner loop.
    b. Because the number of times through the loops is not influenced by the values in the array.
    c. Because the sort must go through the array of values the same number of times no matter what the data is.
    d. Because it swaps the values rather than storing one value in a temporary location.

13. Which of the following statements are true about Shell Sort?

    a. The solution is recursive.
    b. Shell sort is an unstable sort.
    c. Shell sort is an in-place sort.
    d. Shell sort is always more efficient than Merge Sort.
    e. The time complexity of Shell sort is dependent on the increment sequence of the gaps.

14. Which of the following statements are true about Merge Sort?

    a. The solution can use recursion
    b. The worst case complexity is $O(n \log n)$.
    c. It is not a stable sort.
    d. It is adaptive.
    e. It is similar to a binary tree sort.
    f. It works well when sorting structures that do not permit random access (linked lists).
    g. It is difficult to write a parallel version.
    h. Is worse than quicksort (running time, number of comparisons, memory used, and number of writes).

15. Merge sort has a running time of $O(n \log n)$ for worst and average cases. Which of the following statements adds correct details to the description of merge sort's running time?

    a. The number of comparisons merge sort makes is $O(n \log 2)$.
    b. Merge sort requires 2n auxiliary space.
    c. Merge sort makes O(n) recursive calls.
    d. Merge sort can be written as a non-recursive sort.

16. Which of the following scenarios might be a good situation in which to use a merge sort?

    a. When your sorting must be stable.
    b. When you know the data will be partially sorted.
    c. When you need to use the sort on data that is on a tape drive.
    d. When you need the sort to have the same running time, no matter what the data is.
    e. When you have a lot of data to sort.

17. Which of the following statements are true about Quick Sort?

    a. It is a stable sort.
    b. It can be implemented as in in-place sort.
    c. It can be implemented recursively.
    d. It is $O(n \log n)$ for the average case.
    e. The worst case running time is the same as the average case.
    f. It is an adaptive sort.

18. Which of the following scenarios might be good situation in which to use quick sort?

    a. When you know that your data is going to be partially sorted.
    b. When the order of duplicate keys in your data does not matter.
    c. When you must guarantee no more than $n \log n$ running time.
    d. When the amount of memory available is limited.
    e. When the amount of data to be sorted is large.
    f. When all the data to be sorted must be encrypted.