Big-O Review

Below are a number of short pieces of code. For each piece of code, determine its big-O runtime as a function of n. Problems marked with a star are a bit more challenging than the others. Feel free to work with a group on these problems.

1.
   ```
   void function1(int n) {
     for (int i = 0; i < n; i++) {
       print('*');
     }
   }
   ```

2.
   ```
   void function2(int n) {
     for (int i = 0; i < n; i++) {
       for (int j = 0; j < n; j++) {
         print('*');
       }
     }
   }
   ```

3.
   ```
   void function3(int n) {
     for (int i = 0; i < n; i++) {
       for (int j = i + 1; j < n; j++) {
         print('*');
       }
     }
   }
   ```

4.
   ```
   void function4(int n) {
     for (int i = 1; i <= n; i *= 2) {
       print('*');
     }
   }
   ```

5.
   ```
   void function5(int n) {
     if (n == 0) return;
     function5(n - 1);
   }
   ```

6.
   ```
   void function6(int n) {
     if (n == 0) return;
     function6(n - 1);
     function6(n - 1);
   }
   ```

7.
   ```
   void function7(int n) {
     if (n == 0) return;
     function7(n / 2);
   }
   ```

8.
   ```
   void function8(int n) {
     if (n == 0) return;
     for (int i = 0; i < n; i++) {
       print('*');
     }
     function8(n / 2);
     function8(n / 2);
   }
   ```

9.
   ```
   void function9(int n) {  // (*)
     for (int i = 1; i <= n; i *= 2) {
       for (int j = 0; j < i; j++) {
         print('*');
       }
     }
   }
   ```

10.
    ```
    void function10(int n) {  // (*)
      if (n == 0) return;
      for (int i = 0; i < n; i++) {
        print('*');
      }
      function10(n / 2);
    }
    ```

Fundamental Algorithms and Data Structures

Here's a list of conceptual and numerical questions about a variety of data structures and algorithms. Provide an answer to each question. Feel free to work on these problems in groups!

Data Structures: Concept Checks

1. What is the time complexity of adding an element to the end of a dynamic array (like Java's `ArrayList` or C++'s `std::vector`)? What is the time complexity of adding an element to the front of a dynamic array?
2. Give one advantage of representing a list using a dynamic array over using a linked list and vice-versa.
3. Give one example of an operation on a doubly-linked list that is faster than the corresponding operation on a singly-linked list.
4. Give an example of when it would be useful to store head and tail pointers in a linked list.
5. What happens if you push a series of n numbers onto a stack and then pop them all? What is the overall time complexity of performing these operations?
6. What happens if you enqueue a series of n numbers into a queue and then dequeue them all? What is the overall time complexity of performing these operations?
7. What happens if you enqueue a series of n numbers into a priority queue and then dequeue them all? What is the overall time complexity of performing these operations?
8. Draw the binary search tree that results from inserting the elements 4, 1, 3, 2, 6, 5 into an empty tree with no rebalancing. What is the height of this tree?
9. What is the cost of inserting an element into a binary search tree with n elements? How about looking up an element? Removing an element?
10. What data structures might you use to implement a set? List some tradeoffs between each implementation.
11. What data structures might you use to implement a stack? List some tradeoffs between each implementation.
12. What data structures might you use to implement a priority queue? List some tradeoffs between each implementation.
13. What data structures might you use to implement a map? List some tradeoffs between each implementation.
14. What is the average-case time complexity of looking up an element in a hash table? What is the worst-case time complexity of looking up an element in a hash table?
15. Draw a trie containing the strings **at**, **above**, **as**, **ate**, **cat**, and **car**.
16. What is the time complexity of looking up a string of length L in a trie containing n words? How about inserting a string? How about deleting a string?

Algorithms: Concept Checks

1. Give one advantage of binary search over linear search and vice-versa.
2. What is the worst-case time complexity of looking up an element in an array of length n using binary search? How about the best-case complexity?
3. What are the best-case and worst-case runtimes for quicksort on an array of length n?
4. What is the average-case runtime for quicksort on an array of length n when the pivots are chosen randomly?
5. What is the worst-case time complexity of insertion sort? How about the best-case time complexity?
6. Give one advantage of mergesort over quicksort and vice-versa.
7. Give one advantage of insertion sort over quicksort and vice-versa.
8. Give one advantage of heapsort over mergesort and vice-versa.
9. What is counting sort? When would you use it? If the numbers in your input array range from 1 to U, inclusive, and your input has length n, what is the time complexity of counting sort?
10. What is radix sort? When would you use it? If the numbers in your input array range from 1 to U, inclusive, and your input has length n, what is the time complexity of radix sort?
11. Which graph algorithm or algorithms would you use to find the shortest path in a graph between two nodes? Does your answer rely on any assumptions about the graph?
12. Which graph algorithm or algorithms might you use to find a minimum spanning tree in a graph? Does your answer rely on any assumptions about the graph?
13. What is an adjacency list? What is an adjacency matrix? Give one advantage of each over the other.

Binary Search Trees: Coding Problems

1. Write a function that, given a pointer to the root of a binary search tree of integers and a key k, returns whether k is present in the tree.
2. Write a function that, given a pointer to the root of a binary search tree of integers and a key k, inserts k into the tree if it's not already present. Don't worry about keeping the tree balanced.
3. Write a function that, given a pointer to the root of a binary search tree of integers and a key k, returns a pointer to the biggest element of the binary search tree that's less than k, or a null pointer if there is no element in the tree with this property.
4. Write a function that, given a pointer to the root of a binary search tree, prints out the nodes in that tree in a level-by-level order. That is, if you were to draw the tree, the function would print out the root, then all the values of the children of the root from left to right, then all the values of the grandchildren of the root from left to right, etc.

Linked Lists: Coding Problems

1. Write a function that takes as input a pointer to the first element of a linked list of integers and a number k, then appends k to the end of the linked list.
2. Write a function that takes as input a pointer to the first element of a linked list of integers and a number k, then prepends k to the front of the linked list.
3. Write a function that takes as input a pointer to the first element of a linked list, then reverses that linked list. Then, analyze the time complexity of your solution (how long it takes) and the space complexity of your solution (how much memory is needed beyond just the space for the input.)
4. Write a function that takes as input a pointer to the first element of a linked list of integers and a number k, then updates the list by deleting all copies of the number k from the linked list. Then, analyze the time complexity of your solution (how long it takes) and the space complexity of your solution (how much memory is needed beyond just the space for the input.)
5. Write a function that takes as input a pointer to the first element of two different linked lists, then shuffles them together into a single sorted list by interleaving the elements of the two lists together. For example, given the inputs [1, 2, 3, 4, 5] and [a, b, c, d, e], the resulting list should contain [1, a, 2, b, 3, c, 4, d, 5, e]. You can assume the inputs have the same length. Your solution should not allocate any new linked list cells and should instead just rewire existing cells into the appropriate final ordering.
6. Write a function that takes as input a pointer to the first element of a linked list and a number k, then returns a pointer to the kth-to-last element of the linked list. Then, analyze the time complexity of your solution (how long it takes) and the space complexity of your solution (how much memory is needed beyond just the space for the input.) Can you solve this problem in $O(n)$ time and $O(1)$ space?
7. Write a function that takes as input a pointer to the first element of a linked list, then sorts that linked list using mergesort. Your solution should run in time $O(n \log n)$.
8. Write a function that takes as input a pointer to the first element of a linked list, then sorts that linked list using quicksort. Your solution should run in time $O(n \log n)$.

Stacks and Queues: Coding Problems

1. Write a function that takes as input a string of text, then reports whether all the parentheses, square brackets, and curly braces in that text are balanced. Then, analyze the time complexity and space complexity of your solution.
2. Design a data structure that models a stack of integers with the normal stack operations of *push* and *pop*, but which also supports the operation *find-min*, which returns the minimum element in the stack. How efficient can you make these operations?

Sorting and Hashing: Coding Problems

Here is a collection of practice interview questions to work through. In each case, there is an elegant solution involving either sorting or hashing (and, in some cases, both!) For each of these problems, find the most efficient solutions you can. Some solutions might be incomparable to one another (for example, an O(n log n)-time, O(1)-space algorithm versus and O(n)-time, O(n)-space algorithm).

1. You are given an array of n values. Determine which value appears most frequently in the array. If there is a tie, you can return any of the most-frequently-occurring values.
2. You are given an array of n numbers. Given a number k, determine whether the array contains two numbers whose sum is exactly k.
3. You are given a list of n closed intervals. Find the smallest nonnegative integer that does not belong to any of those intervals.
4. You have an array of n integers. Find the largest range [a, b] where every integer in the range appears at least once in the array. For example, given the array 4, 1, 9, 0, 11, 3, 10, 2, the answer would be [0, 4], since all of the values 0, 1, 2, 3, 4 appear at least once in the array.
5. You are given a list of n sets containing exactly two elements (for example, you might get the list {1, 2}, {1, 4}, {3, 5}, {3, 2}, {4, 5}). Determine whether there are any two disjoint sets in the list (two sets are disjoint if they contain no elements in common).
6. You are given an array of n distinct values. The *rank* of a value is the number of values in the original array that are smaller than it. For example, given the array 42, 137, 16, 9, the ranks of those elements would be 2, 3, 1, 0. Given an array of n distinct values, determine the rank of each element in the array.
7. You are given a list of n closed intervals whose endpoints represent times in a day. Some of these intervals might overlap one another. We'll say that a set of intervals is *overlapping* if all of the intervals in the range overlap one another. For example, the set {[0, 4], [1, 3], [2, 8]} is overlapping, while the set {[0, 2], [1, 3], [2, 4], [3, 5]} is not (because [0, 2] and [3, 5] don't overlap). Find the maximum number of intervals from the list that overlap.
8. You are given a list of positive integers. Determine the smallest positive integer that cannot be written as a sum of some of those integers. For example, given the list [4, 8, 1, 2], the answer is 17. Given the list [1, 2, 5, 8], the answer is 4.
9. The Fibonacci sequence is the sequence that starts with 0, 1 and where each successive term is the sum of the two previous terms. The Fibonacci numbers start off 0, 1, 1, 2, 3, 5, 8, 13, 21, … . You are given a list of n positive integers. Determine, for each number in the list, whether it's a Fibonacci number. When analyzing your algorithm, you should come up with a runtime in terms of n, the number of integers given, and U, the largest number given in the array.

Graphs: Algorithm Design Problems

Below is a series of problems that pertain to graphs and graph theory. For each problem, describe an algorithm that would solve that problem and analyze its runtime. Feel free to code up solutions to these problems if you'd like.

1. You are given a social network and a person in that network. Design an algorithm to find everyone in that network who is not a friend of that person, but who is a friend of a friend of that person.
2. Generalize your answer to part (1) of this problem by taking in a social network, a person, and a number k, then finding everyone who is exactly k hops away from the starting person.
3. You are given a country's freeway system as an undirected graph – each edge represents a freeway and nodes represent cities where the freeways meet. Given a starting city, list all the cities that you can reach from that city by driving on the freeway network.
4. Consider a social network where nodes represent people and edges represent who knows who. For example, an edge from person A to person B indicates that person A knows person B, and if there is no reverse edge from person B to person A, then person B doesn't know person A. A *celebrity* is a person who everyone else knows, but doesn't know anyone else. A *spy* is someone who knows everyone else, but who no one else knows. Given a network where you know in advance there's either a spy or a celebrity, find the spy or celebrity. (As a challenge: if you're given the graph as an adjacency matrix, solve this in time O(n).)