Technical Interview Workshop Problems

Kenny Yu

November 4, 2012

Directions. For each of these problems, solve the problem with the most efficient algorithm (in terms of time and space complexity) and state the time and space complexity of your algorithm.

1 Data Structures

1. **Doubly Linked Lists.** In your most comfortable programming language, implement a doubly linked list. What is the time complexity of inserting to the head of the list? To the tail of the list? To somewhere in the middle of the list? Removing from the head? Removing from the tail? Removing from the middle?

2. **Reverse a Linked List.** Given a pointer to the first node of a singly linked list, write a procedure to reverse the linked list. Do this recursively and iteratively. What is the space and time complexity of each implementation? Which way is better, and why?

3. **Cycle in a Linked List.** Given a pointer to the first node of a single linked list, detect if there exists a cycle in the linked list. What is the time complexity of your implementation?

4. **Queues and Stacks.** Implement a queue using stacks. Implement a stack using queues.

5. **Valid Binary Search Trees.** A binary tree rooted at r is a binary search tree if for all nodes u in the left subtree of r, we have $u < r$, and for all nodes v in the right subtree of r, we have $r < v$. Given a binary tree, determine if the binary tree is a binary search tree.

6. **Merge k sorted lists**. Given k sorted lists, each of length n, provide an algorithm to merge the k sorted lists into a single list of length kn. What data structure might you want to use? What is the time complexity of your solution?

2 Bitwise Operators

1. **Powers of 2.** Write a function that determines if a positive integer n is a power of 2.

2. **Number of one bits.** Given a 32-bit unsigned integer n, write a function to return the number of one-bits in the binary representation of n.

3. **Bit Masking.** Given a string containing only lower case letters from the English alphabet, output the letters that are not present in the string.

4. **Division.** Implement integer division without using multiplication or repeated subtraction (i.e. to divide n by d, you may not repeatedly subtract off d from n).

3 Greedy, Divide & Conquer, Dynamic Programming

1. **Making Change in US Currency.** Given an amount n in cents, determine the minimum number of coins needed to make change for n in US currency $(1, 5, 10, 25$ cents).

2. **Making Change in an Arbitrary Currency.** Given an amount n in cents, and a currency system with m different coins valued at $c_1, c_2, c_3, ..., c_m$ cents, determine the minimum number of coins needed to make change for n in this currency system.

3. **Maximal Subarray.** Given an array of n integers, determine the maximum sum that can be generated by a *continuous* subarray of the input array. For example:

$$\begin{aligned}
1, 2, 3, 4 &\rightarrow 1 + 2 + 3 + 4 = 10 \\
2, 3, -1, -3 &\rightarrow 2 + 3 = 5 \\
-1, 5, 100, -1000 &\rightarrow 5 + 100 = 105 \\
-1, -2, -3, -4 &\rightarrow 0 \\
\textit{empty} &\rightarrow 0 \\
1000, 2000, -1, 3000 &\rightarrow 1000 + 2000 - 1 + 3000 = 5999
\end{aligned}$$

4. **Stocks.** Given an array of n integers representing the price of a stock over the course of n days, determine the maximum profit you can make if you can buy and sell exactly 1 stock over these n days. Provide a divide & conquer algorithm, and a dynamic programming algorithm. Which is better? What is the time and space complexity for both solutions?

4 Miscellaneous Questions

1. **Shuffle.** Given an array of n integers, write a function `shuffle` that when given the array, shuffles the array in place such that all n! permutations of the n integers are

equally likely. You may assume that you have a random number generator that when given a non-negative integer $i < 2^{32}$, can generate integers in the range $[0, i)$ such that any integer in this range is equally likely. Prove that your algorithm generates every permutation with equal probability.

2. **Sorting.** In your most comfortable programming language, implement merge sort, quick sort, insertion sort, and selection sort.

3. `atoi` **(ASCII to Integer).** Implement the function `int atoi(char *str)` that when given a string `str`, returns the numeric value of this string. For example, `atoi("42") == 42`, `atoi("-35") == -35`. You may assume that `str` contains only numeric characters and possibly a leading negative sign.

4. **Reverse a String.** Write a function that when given a string `char *str`, reverses the string in place.

5. **Two-Sum.** Given an array of n integers and a target integer t, return $TRUE$ if there exists $0 \leqslant i < j < n$ such that $a[i] + a[j] = t$, otherwise return FALSE.

6. **Three-Sum.** Given an array of n integers and a target integer t, return $TRUE$ if there exists $0 \leqslant i < j < k < n$ such that $a[i] + a[j] + a[k] = t$, otherwise return FALSE.

7. **Exponentiation.** Write a function that when given non-negative integers n and m, returns n^m. What is the time complexity of your algorithm?

8. **Square Root.** Write a function `float sqrt(float n, float epsilon)` that when given some input float `n` and float `epsilon`, returns a float `m`, the square root approximation of `n` such that `|m - n| < epsilon`.

9. **Largest Prime Factor.** Given a positive integer $p \geqslant 2$, write a function that returns the largest prime factor of p. What is the time complexity of your algorithm?