HCS Bootcamp 1:
Technical Interview Workshop Problems

Harvard Computer Society

January 30, 2013

DISCLAIMER. These problems were compiled mainly through searching online and CS124. Your interview preparation should **not** be limited to this list. It is simply to give you a flavor of the kinds of problems you **might** see in an interview. Solutions will **not** be provided–the fun is in figuring out these problems yourself!

DIRECTIONS. For each of these problems, solve the problem in your most familiar language 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. How much space are you using? Time?

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).

5. **Absolute Value.** Without using subtraction, write a function that computes the absolute value of two integers x and y.

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?

5. **Longest Path in a Graph**. Given a weighted directed acyclic graph $G = (V, E)$ where edges have associated positive weights, compute the length of the longest path in the graph.

6. **Interval Scheduling Problem (Tardos, 4.1).** We have a set of requests $\{1, 2, ..., n\}$; the i-th request corresponds to an interval of time starting at $s(i)$ and finishing at $f(i)$. A subset of the requests is *compatible* if no two of them overlap in time. Write a function that when given the array of start times s and finish times f, computes the largest subset of requests that are compatible.

7. **Weighted Interval Scheduling Problem (Tardos, 6.1)**. This is similar to the Interval Scheduling Problem, except now each request has an associated *value* or *weight* $v_i > 0$. Write a function to compute a compatible subset of the intervals of maximum total value.

8. **Knapsack Problem (Tardos, 6.4)**. We are given n items $\{1, 2, ..., n\}$, and each item has a given nonnegative weight w_i and given value v_i for $i = 1, ..., n$. We are also given a bound W on the total weight. Write a function that when given W and the array of weights and values, computes the subset of items that maximizes the total value and has total weight not exceeding W.

9. **Rod-Cutting Problem (CLRS, 15.1)**. Serling Enterprises buys long steel rods and cuts them into shorter rods, which it then sells. Each cut is free. The management of Serling Enterprises wants to know the best way to cut up the rods. We assume that we know, for $i = 1, 2, ...$, the price p_i in dollars that Serling Enterprises charges for a rod of length i inches. Rod lengths are always an integral number of inches. Given a rod of length n inches and a table of prices p_i for $i = 1, 2, ..., n$ determine the maximum revenue obtainable by cutting up the rod and selling the pieces. Note that if the price p_n for a rod of length n is large enough, an optimal solution may require no cutting at all.

4 Miscellaneous Questions

1. `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.

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

3. **Needle in Haystack.** Given a string *needle* and string *haystack*, return $TRUE$ if *needle* is a substring of *haystack*, otherwise return $FALSE$.

4. **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.

5. **Find repeat.** You are given an array of n integers from the interval $[0, n-2]$. By the pigeonhole principle, there exists at least one integer i in this interval that is repeated in this array. Write a function that when given an array of this form, returns any number that is repeated in the array. Can you do this linear time and constant space?

6. **Dutch Flag Problem.** Given an array of n integers and two numbers *low* and *high*, partition the array in place such that all entries v appear in the array such that all entries such that $v < low$ appear first (in any order), followed by all entries such that $low < v < high$ followed by all entries such that $v > high$.

7. **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.

8. **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.

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

10. **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 $|\sqrt{\texttt{n}} - \texttt{m}| < \texttt{epsilon}$.

11. **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?

12. **LRU Cache.** A cache is a data structure that allows you to store a limited number of items and retrieve them very quickly. In the language of your choice, write down the interface (whatever an interface means in your language) for a cache. Now implement the cache using an LRU policy (this means that if the cache is full, the least recently used item is evicted to make space for the new item).