A Collection of Technical Interview Questions

http://www.spellscroll.com

December 3, 2008

Contents

Disclaimer

All the questions (and solutions) presented in this document were originally collected by the *spellscroll.com* team from various sources including books, mailing-lists and online forums, *etc.* The *spellscroll.com* team spent significant amount of time in selecting, editing and formatting these questions (and solutions), in the hope that our efforts will be of help to people who are preparing technical interviews. Any comments, suggestions and corrections are welcome. This document is free to be downloaded, copied and distributed so long as this disclaimer is clearly reproduced at its beginning. **However, we are NOT responsible for the possible mistakes contained in this document or any improper use of this document.**

Chapter 1

Data Structures and Algorithms

1.1 Linked List Problems

Problems 1-7 and accompanying discussions are taken from the book [Mon07].

1. [**Stack Implementation**] Discuss the stack data structure. Implement a stack in C using either a linked list or a dynamic array, and justify your decision. Design the interface to your stack to be complete, consistent, and easy to use.

Discussion:

- A stack is last-in-first-out data structure: Elements are always removed in the reverse order in which they were added. The add element and remove element operations are conventionally called push and pop, respectively. Stacks are useful data structures for tasks that are divided into multiple subtasks. Tracking return addresses, parameters, local variables for subroutines is one example of stack use; tracking tokens when parsing a programming language is another.
- One of the ways to implement a stack is by using a dynamic array, an array that changes size as needed when elements are added. The main advantage of dynamic arrays over linked lists is that arrays offer random access to the array elements. However, operations on a stack always work on one end of the data structure (the top of the stack), so the random accessibility of a dynamic array

gains you little. In addition, as a dynamic array grows, it must occasionally be resized, which can be a time-consuming operation as elements are copied from the old array to the new array.

- Conversely, linked lists usually allocate memory dynamically for each element, and that overhead can be significant when dealing with small-sized elements, especially if heuristics are used to minimize the number of times the array has to be resized. For these reasons, a stack based on a dynamic array is usually faster than one based on a linked list. In the context of an interview, though, the primary concern is ease and speed of implementation: Implementing a linked list is far less complicated than implementing a dynamic array, so a linked list is probably the best choice for your solution.
- A C-style implementation

```
typedef struct Element {
   struct Element *next;
   void *data;
} Element;


bool push( Element **stack, void *data);
bool pop( Element **stack, void **data);
bool createStack(Element **stack);
bool deleteStack(Element **stack);

bool createStack(Element **stack){
 *stack = NULL;
 return true;
}

bool push( Element **stack, void *data){
  Element *elem = new Element;
  if (!elem) return false;
  elem->data = data;
  elem->next = *stack;
  *stack = elem;
  return true;
}

bool pop(Element **stack, void **data){
```

```
  Element *elem = *stack;
  if (!elem) return false;

  *data = elem->data;
  *stack = elem->next;
  delete elem;
  return true;
}

bool deleteStack(Element **stack){
 Element *next;
 while (*stack)
 {
  next = (*stack)->next;
  delete *stack;
  *stack = next;
 }
 return true;
}
```

- A C++ version

```
class Stack  {
    public :
     Stack();
     virtual ~Stack();
     void push(void *data);
     void pop();

    protected :
      typedef struct Element{
        struct Element *next;
        void *data;
      }Element;
      Element *head;
};

Stack::Stack() {
    head = NULL;
    return;
}
```

```
Stack::~Stack() {
   while(head) {
      Element* next = head->next;
      delete head;
      head = next;
   }
   return;
}

void Stack::push( void *data) {
   Element *element = new Element;
   element->data = data;
   element->next = head;
   head = element;
}

void* Stack::pop(){
   Element *element = head;
   void *data;
   if (!element)
      throw StackError(E_EMPTY);

   data = element->data;
   head = element->next;
   delete element;
}
```

---

2. [**Maintain Linked List Tail Pointer**] *head* and *tail* are global pointers to the first and last element, respectively, of a singly-linked list of integers. Implement C functions for the following prototypes:

   ```
   bool remove(Element *elem);
   bool insertafter(Element *elem, int data);
   ```

   The argument to *remove* is the element to be deleted. The two arguments to *insertAfter* give the data for the new element and the element after which the new element is to be inserted. It should be possible to insert at the beginning of the list by calling *insertAfter* with *NULL* as

the element argument. These functions should return true if successful and false is unsuccessful.

Your functions must keep the *head* and *tail* pointers current.

```
bool remove(Element *elem){
    Element * curPos = head;
    if (!elem)
       return false;
    if (elem==head) {
        head = elem->next;
        delete elem;
        if (!head)  /* special case for 1 element list */
           tail = NULL;
        return true;
    }

    while (curPos){
          if (curPos==elem){
                curPos->next = elem->next;
                delete elem;
                if (curPos->next==NULL)
                   tail = curPos;
                return true;
          }
          curPos = curPos->next;
    }
    return false;
}

bool insertAfter(Element *elem, int data) {
   Element *newElem, *curPos = head;
   newElem = new Element;
   if (!newElem){
       return false;
   }
   newElem->data = data;
   /* Insert at beginning of list */
   if (!elem){
       newElem->next = head;
```

```
            head = newElem;
            /* special case for empty list */
            if (!tail)
               tail = newElem;
            return true;
        }
        while (curPos){
            if (curPos == elem) {
                  newElem->next = curPos->next;
                  curPos->next = newElem;
                  /* special case for inserting at the end of list */
                  if (!(newElem->next))
                     tail = newElem;
                  return true;
            }
            curPos = curPos->next;
        }

        /* Insert Position not found */
        delete newElem;
        return false;
    }
```

---

3. [**Bugs in removeHead**] Find and fix the bugs in the following C/C++ function that is supposed to remove the head element from a singly-linked list:

```
void removeHead(Node *head){
   delete head;
   head = head->next;
}
```

- Check that (i)the data comes into the function properly; (ii) each line of the function works correctly; (iii) Check that the data comes out the function correctly; (iv) Check the common error conditions.
- Corrected code:

```
void removeHead(Node **head){
   Node* temp;
   if (!(*head)){
       temp = (*head)->next;
       delete *head;
      *head = temp;
   }
}
```

---

4. [**Mth-to-Last Element of a Linked List**] Given a singly-linked list, devise a time- and space-efficient algorithm to find the mth-to-last element of the list. Implement your algorithm, taking care to handle relevant error conditions. Define mth to last such that when m=0, the last element of the list is returned. [1]

```
Element* findMToLastElement(Element* head, int m){
   Element *current, *mBehind;
   int i;
   /* Advance current m elements from beginning, checking for
     the end of the list */
   current = head;
   for (i = 0; i<m; i++){
      if (current->next) {
          current = current->next;
      }
      else{
          return NULL;
      }
   }
   /* Start mBehind at beginning and advance pointers together
   until current hits last element */
   mBehind = head;
   while (current->next){
        curren=current->next;
        mBehind = mBehind->next;
   }
```

[1] *A variant of this problem is to find the middle node of a given singly-linked list.*

```
   /* mBehind now points to the element we were searching for */
   return mBehind;
}
```

---

5. [**Null or Cycle**] Write a function that takes a pointer to the head of a list and determines whether the list is cyclic or acyclic. Your function should return false if the list is acyclic and true if it is cyclic. You may not modify the list in any way.

- Algorithm outline:

```
Start two pointers at the head of the list
Loop infinitely
    If the fast pointer reaches a NULL pointer
        Return that the list is Null terminated
    If the fast pointer moves onto or over the slow pointer
        Return that there is a cycle
    Advance the slow pointer one node
    Advance the fast pointer two nodes
```

- Code

```
/* Takes a pointer to the head of a linked list and determine if the
   list ends in a cycle or is NULL terminated */
bool determineTerminate(Node *head){
   Node *fast, *slow;
   fast = slow = head;

   while(true)
   {
       if (!fast||!fast->next) return false;
       else if(fast==slow||fast->next==slow) return true;
       else {
            slow = slow->next;
            fast = fast->next->next;
       }
   }
}
```

---

6. **How do you reverse a linked list. Why don't you use recursion?**

```
Node* reverseLinkedList(Node *base ){
    Node* next; Node *current ;
    current = base;
    node *previous = NULL;

    while(current != NULL ) {
        next = current->next;
        current->next = previous;
        previous = current;
        current = next;
    }
    return (previous);
}

// Recursive version
Node *reverse(Node *head) {
    Node *temp;
    if(head->next) {
        temp = reverse(head->next);
        head>next->next = head;
        head->next = NULL;
    }
    else
      temp = head;
    return temp;
}

call as: head=reverse(head);
```

---

7. [**Sort a linked list**]

```
//sorting in descending order
struct Node { int value; Node* NEXT; }
//Assume HEAD pointer denotes the first element in the list
//only change the values, don't have to change the pointers
Sort( Node *Head) {
    node* first,second,temp;
    first = Head;
    while(first!=null) {
        second = first->NEXT;
        while(second!=null) {
            if(first->value < second->value) {
                temp = new node();
                temp->value = first->value;
                first->value = second->value;
                second->value = temp->value;
                delete temp;
            }
            second = second->NEXT;
        }
        first = first->NEXT;
    }
}
```

8. Given a linked list which is sorted. How will you insert new elements in sorted way.

9. Delete an element from a singly linked list. How about deleting an element from a doubly linked list.

10. Under what circumstances can one delete an element from a singly linked list in constant time?

11. What is the difference between arrays and linked lists? What is the advantage of contiguous memory over linked lists?

12. Write the most efficient algorithm to find the minimum element in a sorted circular list.

More linked list problems can be found at `http://cslibrary.stanford.edu/105/LinkedListProblems.pdf`

1.2 Problems on Trees and Graphs

Problems 1-6 and accompanying discussions are taken from the book [Mon07].

1. [**Preorder Traversal, No Recursion**] Perform a preorder traversal of a binary search tree, printing the value of each node, but this time you may not use recursion.

   - Algorithm outline:

     ```
     Create the stack
     Push the root node on the stack
     While the stack is not empty
        Pop a node
        If he node is not null
            print its value
            Push the node's right child on the stack
            Push the node's left child on the stack
     ```

   - Code (with no error checking)

     ```
     void preorderTraversal(Node root){
          NodeStack stack = new NodeStack();
          stack.push( root);
          while (true) {
             Node curr = stack.pop();
             if (curr==null) break;
             curr.printValue();
             Node n= curr.getRight();
             if (n!=null) stack.push(n);
             n = curr.getLeft();
             if (n!=null) stack.push(n);
          }
     }
     ```

2. [**Lowest Common Ancestor**] Given the value of two nodes in a binary search tree, find the lowest (nearest) common ancestor. You may assume that both values already exist in the tree.

   - Algorithm outline:

     ```
     Examine the current node
     If value1 and value2 are both less than the current node
     ```

```
   Examine the left child
If value1 and value2 are both greater than the current node
   Examine the right child
Otherwise
   The current node is the lowest common ancestor
```

- Code

```
Node findLowestCommonAncestor(Node root,
                  int value1, int value2){
   while( root!=null){
     int value = root.getValue();
     if (value>value1 && value>value2) {
         root = root.getLeft();
     }
     else if (value<value1 && value<value2) {
         root = root.getright();
     }
     else {
         return root;
     }
   }
}
```

3. You have a tree (not Binary) and you want to print the values of all the nodes in this tree level by level.

4. Here is a tree. It's a binary tree but in no particular order. How do you write this tree to a file so that it can be reread in an reconstructed exactly as shown?

5. Here is a graph of distribution centers and the weight to go from one to another. How would you represent this using a data structure? Code an algorithm that allows me to check if there is a path from one node to another and what the weight of this path is.

6. **Determine whether a graph is acyclic.** Use the Floyd algorithm to find all shortest path within the adjacency algorithm. If any element on the main diagonal ends up being non-infinity, you know there is a cycle in the graph.

If you want a simple algorithm, just take the 0-1 adjacency matrix and calculate its N+1 power, if it is an $N \times N$ matrix. If the result is all zeroes then there are no cycles. Otherwise there are.

- If the graph has no nodes, stop. The graph is acyclic.
- If the graph has no leaf, stop. The graph is cyclic.
- Choose a leaf of the graph. Remove this leaf and all arcs going into the leaf to get a new graph.
- Go to 1.

7. In a general tree, how would you find the lowest common ancestor of two nodes that are given to you as parameters?

8. If there are two structs, TreeNode and Tree. TreeNode contains 3 elements, data, lChild and rChild. Tree contains 2 elements, int size and TreeNode *root. The tree is a complete tree. So how to find a O(logN) approach to insert a new node.

9. Design an algorithm and write code to serialize and deserialize a binary tree/graph

10. Given two binary trees, find whether or not they are equal. Being equal means that they have the same values and same structure.

11. Given a binary tree, write code to check whether it is a binary search tree or not.

12. Design a graph class. Write the C++ interface for it.

13. Given a binary tree, convert it into a doubly linked list in place.

14. Given a binary tree with nodes, print out the values in pre-order/in-order/post-order without using any extra space.

15. Write a function to find the depth/width of a binary tree.

16. How do you represent an n-ary tree? Write a program to print the nodes of such a tree in breadth first order.

17. Implement a breadth first traversal of a (binary) tree.

18. Find the n-th node in an in-order search of a tree.

19. Given a binary tree with the following constraints: a) A node has either both a left and right child OR no children b) The right child of a node is either a leaf or NULL. write code to invert this tree.

20. What is a balanced tree?

21. Given a graph (any type - Directed acyclic graph or undirected graphs with loops), find a minimal set of vertices which affect all the edges of the graph.

    An edge is affected if the edge is either originating or terminating from that vertex.

22. Tree search algorithms. Write BFS and DFS code, explain run time and space requirements. Modify the code to handle trees with weighted edges and loops with BFS and DFS, make the code print out the path to goal state.

23. Suppose you have N companies, and we want to eventually merge them into one big company. How many ways are there to merge?

1.3 String Manipulation Problems

Problems 1-4 and accompanying discussions are taken from the book [Mon07].

1. [**Find the first nonrepeated character**] Write an efficient function to find the first nonrepeated character in a string.

   - Algorithm outline:

     ```
     First, build the character count hash table:
       For each character
         If no value is stored for the character. store 1
         Otherwise, increment the value
       Second, scan the string:
         For each character
           Return character if count in hash table is 1
         If no characters have count 1, return null
     ```

   - Code

     ```
     public static Character fiestNonRepeated (String str)
     {
         Hashtable charHash = new HashTable();
     ```

```
        int i, length;
        Character c'
        Object seenOnce = new Object();
        Object seenTwice = new Object();

        length =  str.length();
        //Scan str, building hash table
        for (i=0; i< length; i++) {
            c = new Character(str.charAt(i));
            Object o = charHash.get(c);
            if (o==null) {charHash.put(c, seenOnce);}
            else if (o==seenonce) {
            charHash.put(c, seenTwice);
            }
        }
        for (i=0; i< length; i++){
            c = new Character(str.charAt(i));
            if (charHash.get(c) == seenOnce)
                return c;
        }
        return null;
    }
```

A (significantly) further speedup could be achieved by implementing a faster char to Character mapping, possibly using an array to cache the mappings, or at least the most frequent mapping (such as for ASCII characters). Or use a hash table implementation that could directly store character char values.

2. [**Remove Specified Characters**] Write an efficient function that deletes characters from a string. Use the prototype

```
string removeChars(string str, string remove);
```

where any character existing in *remove* must be deleted from *str*. Justify any design decisions you make and discuss the efficiency of your solution.

- Algorithm outline O(m+n):

  1. Set all the elements in your lookup array to false.

  2. Iterate through each character in remove, setting the corresponding value in the lookup array to true.

3. Iterate through str with a source and destination index, copying each character only if its corresponding value in the lookup array is false.

- Code

```
string removeChars(string str, string remove){
  char[] s = str.toCharArray();
  char[] r = remove.toCharArray();
  bool[] flags = new bool[128]; //assume ASCII!
  int len = s.Length;
  int src, dst;

  //Set flags for characters to be removed
  for (src=0; src<len; ++src){
     flags[r[src]] = true;
  }
  src = 0; dst = 0;
  //Now loop through all the characters,
  //Copying only if they are not flagged
  while (src<len){
     if (!flags[(int)s[src]]) { s[dst++] = s[src];}
     ++src;
  }
  return new string(s,0,dst);
}
```

3. [**Reverse Words**] Write a function that reverses the order of the words in a string. For example, your function should transform the string "Do or do not, there is no try." to "try. no is there not, do or Do". Assume that all words are space delimited and treat punctuation the same as letters.

4. [**Integer/String Conversions**] Write two conversion routines. The first routine converts a string to a signed integer. You may assume that the string only contains digits and the minus character('-'), that it is a properly formatted integer number, and that the number is within the range of an int type. The second routine converts a signed integer stored as an int back to a string.

5. Suggest an algorithm to find the first non-repeating character in a given string.

6. Write code to tokenize a string.

7. If you are given a number as a parameter, write a function that would put commas after every third digit from the right.

8. Implement the unix command WordCount (wc) using language of your choice. The method takes a string and returns the number of words in the string, number of chars and the number of lines.

9. Write a function that returns the longest run of a letter in a string. e.g. “cccc” in the string “abccccdef”.

10. Write an atoi() function for decimal. Once this was done, generalize it for any radix.

11. Write a program to convert a decimal number to Roman numerals.

12. Algorithm and code to detect occurrence of a string(patterns) in another string ex: aab, aababa.

13. Write a function that returns the longest palindrome in a given string. e.g “ccddcc” in the string “abaccddccefe”.

14. Develop an algorithm for printing different permutations of the string. Try both recursive and non recursive solution.

15. Write a function that takes a string and converts ’%20’ sequences to spaces (in place).

1.4 Recursion Problems

Problems 1-4 and accompanying discussions are taken from the book [Mon07].

1. [**Binary Search**] Implement a function to perform a binary search on a sorted array of integers to find the index of a given integer. Use this method declaration.

   int binarySearch(int[ ] array, int lower, int upper, int target);

   Comment on the efficiency of this search and compare it with other search methods.

2. [**Permutations of a String**] Implement a routine that prints all possible orderings of the characters in a string. Treat each character in the input string as a distinct character, even if it is repeated. Given

the string "aaa", your routine should print "aaa" six times. You may print the permutation in any order you choose.

3. [**Combinations of a String**] Implement a routine that prints all possible combinations of the characters in a string. These combinations range in length from one to the length of the string. Two combinations that differ only in ordering of their characters are the same combinations.

4. [**Telephone Words**] Write a routine that takes a seven-digit telephone number and prints out all of the possible "words" or combinations of letters that can represent the given number. Because the 0 and 1 keys have no letters on them, you should change only the digits 2-9 to letters. You'll be passed an array of seven integers, with each element being one digit in the number. You may assume that only valid phone numbers will be passed to your routine. You can use the helper routine

```
char getCharKey(int telephoneKey, int place)
```

which takes a telephone key (0-9) and a place of either 1,2,3 and returns the character corresponding to the letter in that position on the specified key.

5. Code a function that returns the n-th fibonacci number. Do it in the most efficient way by using 3 variable. What are the pros and cons of your(iterative) approach and recursive approach.

1.5 Problems on Searching and Sorting

Recommended readings: [Knu98a; Knu98b; Ben99; Gus97]

1. How do you merge n sorted lists with average length K in O(n*log(K)) time?

2. Implement a binary search in a sorted array.

```
int binarysearch(DataType t) {
    int l, u, m;
    l = 0;
    u = n-1;
    while (l <= u) {
        m = (l + u) / 2;
        if (x[m] < t)
            l = m+1;
        else if (x[m] == t)
            return m;
        else /* x[m] > t */
            u = m-1;
    }
    return -1;
}
```

3. Design an algorithm and write code to find two numbers in an array whose sum equals a given value.

4. Given a list of numbers and a target sum, how would you efficiently determine whether a pair of numbers from the list can add up to the target sum? (list based approach is $O(n^2)$ while hash-table based approach is $O(n)$)

5. Given an array of integers, find the contiguous sub-array with the largest sum.

6. Suppose you have an $N \times N$ matrix of positive and negative integers. Write some code that finds the sub-matrix with the maximum sum of its elements.

7. How to find the two numbers whose difference is minimum among the set of numbers.

8. Given an array of length N containing integers between 1 and N, determine if it contains any duplicates.

9. Design an algorithm to find duplicates in an array. Discuss different approaches.

10. How would you detect a repeated element in an integer array. Discuss various solutions and the order of the algorithm.

11. Write a program to remove duplicates from a sorted array.

12. Sort an array of size n containing integers between 1 and K, given a temporary scratch integer array of size K.

13. You have a fixed list of numbers. Now given any other list, how can you efficiently find out if there is any element in the second list that is an element of the first list (fixed list).

14. Given 2 files, each with names of thousands of customers who bought something from Amazon.com on that particular day. Find the common customers (i.e. common names in file1 and file2)

15. If you are given a set of 1000 integers in a set A , and 10,000,000 integers in a set B, how would you create a set C that would only contain numbers that are both in A and B?

16. Find the intersection of 2 sorted integer arrays. What if one of them is huge? What if one of them is so huge, it can't fit in memory. How do you minimize the number of disk seeks?

17. From K sorted arrays, each of size N, how would you construct one big array, and what would the big-O of the procedure be? What if you only had memory of size 2N.

18. You have a file with millions of lines of data. Only two lines are identical; the rest are all unique. Each line is so long that it may not even fit in memory. What is the most efficient solution for finding the identical lines?

19. Design an algorithm to find the 100 shortest distances to stars in the universe.

20. Given 1 million customer addresses, how to store and search them efficiently. Suppose you can search the address by phone number or name or account number.

21. Given two log files, each with a billion usernames (each username appended to the log file), find the usernames existing in both documents in the most efficient manner? Use pseudo-code or code. If your code calls pre-existing library functions, create each library function from scratch.

22. You have 50,000 html files, some of which contain phone numbers. How would you create a list of all the files which contain phone numbers?

23. Given a file containing approx 10 million words, Design a data structure for finding all the anagrams in that set.

24. Design an algorithm to find the most user viewed pages.

25. You're looking for a set of words in a file. Return the location of the shortest excerpt containing all of these target words.

26. Given a unsorted queue, sort it via using another queue and some variable. what if use stack instead of queue?

27. What method would you use to look up a word in a dictionary?

28. You have a collection of numbers that's so big that it would not fit onto one single computer. It is therefore scattered across N number of machines, with roughly L number of numbers in each machine, how would you find the median in this entire collection?

29. Design an algorithm which can find all the dictionary words correspond to a n-digit telephone number, assuming each digit maps to several alphabet letter.

30. What is the best and worst performance time for a hash tree and binary search tree?

31. If you had a million integers how would you sort them efficiently, and how much memory would that consume? (modify a specific sorting algorithm to solve this)

32. Find or determine non existence of a number in a sorted list of N numbers where the numbers range over M, $M >> N$ and N large enough to span multiple disks. Algorithm to beat $O(\log n)$; bonus points for a constant time algorithm.

33. How do you put a Binary Search Tree in an array in a efficient manner?

34. How do you find out the fifth maximum element in a Binary Search Tree in an efficient manner?

35. Given two sequences of items, find the items whose absolute number increases or decreases the most when comparing one sequence with the other by reading the sequence only once.

36. Given that you have one string of length N and M small strings of length L. How do you efficiently find the occurrence of each small string in the larger one?

    Given That One of the strings is very very long, and the other one could be of various sizes. Windowing will result in O(N+M) solution but could it be better? May be NlogM or even better?

37. You are given a small sorted list of numbers, and a very very long sorted list of numbers - so long that it had to be put on a disk in different blocks. How would you find those short list numbers in the bigger one?

38. Given a file of 4 billion 32-bit integers, how to find one that appears at least twice?

39. Write a program for displaying the ten most frequent words in a file such that your program should be efficient in all complexity measures.

40. Given an array, i) find the longest continuously increasing subsequence. ii) find the longest increasing subsequence.

41. You are given with three sorted arrays (in ascending order), you are required to find a triplet (one element from each array) such that distance is minimum. Distance is defined like this : If a[i], b[j] and c[k] are three elements then distance=max(abs(a[i]-b[j]),abs(a[i]-c[k]),abs(b[j]-c[k])). Please give a solution in O(n) time complexity.

1.6 Problems on Numbers

1. Design an algorithm to find the kth number divisible by only 3 or 5 or 7 i.e the only factors of these numbers should be 3,5 and 7.

2. Write a function that prints out all sets of consecutive integers that add up to all and any numbers within a given range. For example, given a range of 4-9, your function must print out the fact that $5 = 2 + 3, 6 = 1 + 2 + 3, 7 = 3 + 4, 9 = 2 + 3 + 4, 9 = 4 + 5$.

3. Write a function to determine whether a number is a power of two (used a bit-shifting based algorithm).

4. Describe an algorithm to find out if an integer is a square? (e.g. 16 is, 15 isn't)

5. Given a function that returns a random number between 1-5, write one that returns a random number between 1-7.

6. Given a number, describe an algorithm to find the next number which is prime.

7. Write a function that outputs an integer in ASCII format.

1.7 Geometry Problems

1. In a plane, n points are given i.e. the input is $(x_1, y_1), (x_2, y_2), \cdots (x_n, y_n)$. Now given these $npoints$, find the maximum number of collinear points.

2. There is a convex polygon with n vertices. Given a point, determine whether this point is inside the polygon.

3. Given n points in the plane, find the pair that is closet together.

4. How many lines can be drawn in a 2D plane such that they are equidistant from 3 non-collinear points ?

1.8 Miscellaneous Problems

1. Implement a queue in an array.

2. How would you implement a stack to achieve constant time for "push", "pop" and "find minimum" operations?

3. Implement division (without using the divide operator, obviously).

4. Develop an algorithm to find out all valid combinations of n brackets. Like for n =3 possible combinations can be ((())) or ()()() or (()()) and so on.

5. You need read a lot of records, you don't know how many records here before you complete it. After you read all those records, you need select k record randomly. In another words, every record has same chance to be selected. The memory is limited, that means you cannot store all those records at one time.

6. Your input is a string which is composed from bracket characters. The allowed characters are: ()[]{} <>. Your mission is to determine

whether the brackets structure is legal or not. Example of a legal expression: ([](< {} >)). Example of an illegal expression: ({<) >}. Provide the most efficient, elegant and simple solution for that problem.

7. An array of integers of size n. Generate a random permutation of the array, given a function rand_n() that returns an integer between 1 and n, both inclusive, with equal probability. What is the expected time of your algorithm?

8. How could a linked list and a hash table be combined to allow someone to run through the list from item to item while still maintaining the ability to access an individual element in O(1) time?

9. **Shuffling**

```
/*
 * get_shuffle --
 *  Construct a random shuffle array of t elements
 */
static size_t * get_shuffle(size_t t) {
    size_t *shuffle;
    size_t i, j, k, temp;

    shuffle = emalloc(t * sizeof(size_t));

    for (i = 0; i < t; i++)
        shuffle[i] = i;

    /*
     * This algorithm is taken from D. E. Knuth,
     * The Art of Computer Programming, Volume 2:
     * Seminumerical Algorithms, 2nd Ed., page 139.
     */

    for (j = t - 1; j > 0; j--) {
        k = arc4random() % (j + 1);
        temp = shuffle[j];
        shuffle[j] = shuffle[k];
        shuffle[k] = temp;
    }
```

```
        return shuffle;
    }
```

10. You have a stream of infinite queries (ie: real time Google search queries that people are entering). Describe how you would go about finding a good estimate of 1000 samples from this never ending set of data and then write code for it.

11. There are a set of n integers. Describe an algorithm to find for each of all its subsets of n-1 integers the product of its integers.

12. Given a set of coin denominators, find the minimum number of coins to give a certain amount of change.

Bibliography

[Ben99] J. Bentley. *Programming Pearls*. ACM Press, 2nd edn., 1999. Buy at Amazon.com: http://www.amazon.com/exec/obidos/ASIN/0201657880/spellscrollco-20.

[Cor01] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, 2nd edn., 2001. Buy at Amazon.com: http://www.amazon.com/exec/obidos/ASIN/0262032937/spellscrollco-20.

[Das06] S. Dasgupta, C. H. Papadimitriou, and U. Vazirani. *Algorithms*. McGraw-Hill, 2006. Buy at Amazon.com: http://www.amazon.com/exec/obidos/ASIN/0073523402/spellscrollco-20.

[Gus97] D. Gusfield. *Algorithms on strings, trees, and sequences: computer science and computational biology*. Cambridge University Press, 1997. Buy at Amazon.com: http://www.amazon.com/exec/obidos/ASIN/0521585198/spellscrollco-20.

[Kle05] J. Kleinberg and Éva Tardos. *Algorithm Design*. Addison Wesley, 2005. Buy at Amazon.com: http://www.amazon.com/exec/obidos/ASIN/0321295358/spellscrollco-20.

[Knu98a] D. E. Knuth. *The Art of Computer Programming, Volume 1: Fundamental Algorithms*. Addison-Wesley Professional, 2nd edn., 1998. Buy at Amazon.com: http://www.amazon.com/exec/obidos/ASIN/0201896834/spellscrollco-20.

[Knu98b] D. E. Knuth. *The Art of Computer Programming, Volume 3: Sorting and Searching*. Addison-Wesley Professional, 2nd edn., 1998. Buy at Amazon.com: http://www.amazon.com/exec/obidos/ASIN/0201896850/spellscrollco-20.

[Mon07] J. Mongan, N. Suojanen, and E. Giguére. *Programming Interviews Exposed: Secrets to Landing Your Next Job*. Wrox, 2nd edn.,

2007. Buy at Amazon.com: http://www.amazon.com/exec/obidos/ASIN/047012167X/spellscrollco-20.

[Sed01] R. Sedgewick. *Algorithms in C++, Parts 1-5: Fundamentals, Data Structure, Sorting, Searching and Graph Algorithms*. Addison-Wesley Professional, 3rd edn., 2001. Buy at Amazon.com: http://www.amazon.com/exec/obidos/ASIN/0201756080/spellscrollco-20.