Asymptotics , Recurrence and Basic Algorithms

1. [1 pt] What is the solution to the recurrence **$T(n) = 2T(n-1) + 1$, $T(1) = 1$**

    1. O(logn)
    2. O(n)
    3. O(nlogn)
    4. $O(n^2)$
    5. $O(2^n)$

2. [1 pt] What is the solution to the recurrence **$T(n) = T(n/2) + n$, $T(1) = 1$**

    1. O(logn)
    2. O(n)
    3. O(nlogn)
    4. $O(n^2)$
    5. $O(2^n)$

3. [1 pt] What is the solution to the recurrence **$T(n) = 2T(n/2) + 1$, $T(1) = 1$**

    1. O(logn)
    2. O(n)
    3. O(nlogn)
    4. $O(n^2)$
    5. $O(2^n)$

4. [1 pt] Determine the **correct** statement(s) in the group below
    1. The worst-case performance of quick sort using a random partition element is the same as that of merge sort
    2. The worst-case performance of quick sort using the median as a partition element is different from that of heap sort
    3. Heap sort has the same worst-case performance as merge sort
    4. All three algorithms mentioned above are based on the divide and conquer paradigm, i.e., divide the set into two, recursively sort each part, and combine the results.
    5. Merge sort does not requires any more extra space than the other two algorithms

5. [1 pt] Determine the **incorrect** statement in the group below

    1. The worst-case performance of quick sort is more than that of merge sort
    2. Heap sort has the same worst-case performance as merge sort
    3. The worst-case performance of quick sort can be improved using the median-finding algorithm
    4. All three algorithms mentioned above are based on the divide and conquer paradigm, i.e., divide the set into two, recursively sort each part, and combine the results.
    5. Merge sort requires more extra space than the other two algorithms

6. [1 pt] Determine the **correct** statement(s) in the group below

    1. The worst-case performance of quick sort using a random partition element is the same as that of bubble sort
    2. The worst-case performance of quick sort using the median as a partition element is different

from that of bubble sort
3. Heap sort has the same worst-case performance as merge sort
4. Of the four sorting algorithms mentioned above, three algorithms are based on the divide and conquer paradigm, i.e., divide the set into two, recursively sort each part, and combine the results.
5. Quick sort does not requires any more extra space than heap sort

BST and Balanced Trees :

1.Assume that a binary search tree (not a red-black tree) tree was created by inserting the seven numbers in the sequence 1, 2, ..., 7 .

i. Draw the resultant tree

ii. Suppose you are now allowed to do rotation operations anywhere in the tree. Draw a sequence of rotation operations that will eventually complete balance the tree. Clearly mention which edges are being rotated.

2. Suppose you are given a completely balanced binary search tree (a completely balanced tree means that each path from root to leaf is exactly the same). Finding the median of the elements of such a tree takes time

1. O(1)
2. O(log*n)
3. O(logn)
4. O(n)
5. O(nlogn)

3. [1 pt] Suppose you are given a completely balanced binary search tree (a completely balanced tree means that each path from root to leaf is exactly the same). Determine the correct statement(s) below:

1. Finding the largest element of this tree can be done in O(1) time
2. Finding the second smallest element requires at least O(logn) time
3. Finding the median can be done in O(1) time
4. Finding the median requires at least O(logn) time
5. None of the above

Heaps:

1. Design an efficient algorithm to find the n/2th largest element in a binary heap of n elements. What is the running time? (Let h denote the height of the heap.)

2. Assume a heap is arranged such that the largest element is on the top. Suppose you are given two heaps S and T, each with m and n elements respectively. What is the running time of an efficient algorithm to find the 10th largest element of S U T?

3. Suppose you implement a heap (with the largest element on top) in an array. Consider the different arrays below, determine the one that cannot possibly be a heap:

1. 7654321
2. 7362145
3. 7643521
4. 7364251
5. 7463215

4. [1 pt] Suppose you implement a heap (with the largest element on top) in an array. Consider the different arrays below, determine the one(s) that **cannot possibly be a heap**:

1. 7 3 6 2 1 4 5
2. 7 3 6 4 2 5 1
3. 7 4 6 3 2 1 5
4. 7 6 4 3 5 2 1
5. 7 6 5 4 3 2 1

Red-Black Trees

1. Insert the following numbers into an initially empty red-black tree: 8 2 4 7 5 3 1 6. (To solve this problem correctly, you will need to refer to the text-book for details about red- black trees in addition to the main ideas discussed in class).

2. Design a "worst-case" red-black tree with 10 nodes, i.e., a red-black tree with the longest possible path from the root to a leaf.

3. What is minimum possible number of nodes in a red-black tree which contains two black nodes from every root-to-leaf path?

4.Suppose you are given a balanced binary search tree (such as a red-black tree). Design an algorithm to find the second largest element in the tree. What is the running time of your algorithm? Explain your answer in 1-2 sentences

5. Suppose we define a red-black tree where along each path the number of black nodes is the same, and there cannot be more than three consecutive red nodes (but there may be two consecutive red nodes).

i. What is the ratio of the longest possible path length to the shortest possible path length in this tree?

ii. For a tree that has b black nodes along each path, what ratio of the maximum possible number of nodes to the minimum possible number of nodes in the tree?

6. Assume you are given a red-black tree with seven nodes containing the numbers 1, 2...7. Assume that all the nodes are black.

i. Draw the tree

ii. Insert the number 4.5 into the tree and show the sequence of steps required to rebalance the tree.

Union-find

1. Can you use any of the previously studied data structures (e.g. heaps, red-black trees) for the Union-Find problem? Explain your answer.

2. What is the the worst-case performance of the Union-Find data structure we discussed (with union by rank and path-compression)?

3. Suppose we are working with a Union-Find data structure as described in class (assume no path compression is applied). Suppose we consider a set of 10 items that are eventually joined into a single set via 9 union operations

i. Describe the sequence of union operations that will result in the root having the largest number of children. What is the final degree of the root in this case?

ii. Can you have a sequence of union operations that will result in the root having three children? Explain why or why not.

4.Suppose we are working with a Union-Find data structure as described in class (this time path compression is allowed for Find operations). Node z is at a distance of 4 from the root r of its tree (i.e. 4 edges away). We now perform a Find on z.

i. By how much will the degree of r change? Explain your answer.

ii. By how much will the degree of z change? Explain your answer.

5. Recall the definition of the "iterated logarithm function" log*(n).

i. Similarly, try and define the "iterated square root function" sqrt*(n).

ii. What is the value of sqrt*(2^32)?

6. [1 pt] Determine the incorrect statement below concerning the Union-Find data structure we discussed in class (with union by rank and path-compression).

1. The union operation sometimes may not increase the height of the resultant tree
2. The union operation can at most increase the height of the resultant tree by one
3. The union operation always increases the height of the resultant tree by one
4. The find operation sometimes may not increase the degree of the root of the resultant tree
5. The find operation requires two traversals from node to root

Hash Tables

1. Consider a hash table with 10 slots. The hash function is h(k) = k mod 10. Show how the hash table looks after the elements <80, 32, 38, 12, 34, 83, 45> are inserted when collisions are resolved using

a. separate chaining

b. linear probing

2. Using the same hash table as above. Suppose you use linear probing. What is the size of the largest primary cluster?

3. Given the set of elements <80, 32, 38, 12, 34, 83, 45> which of the following hash functions is perfect – i.e. does not cause any collisions.

a. h(k) = k mod 10

b. h(k) = k mod 100

c. h(k) = k mod 7

d. h(k) = k mod 2

4. Given a group of n people you want to find out if any two of them have same birthday (day and month only). You decide to solve it using hash tables. Describe how would you go about solving this problem – what is the hash function? What is the size of the hash table? What is the high level idea?

5. Assume that the universe contains numbers between 0 and 1023. There are no satellite information and you only need to store whether a number is present or not. You decide to use DAT for representing this data. Since there are no satellite information, you decide to represent DAT using 1024 bits. Describe how the three DAT operations – insert, search, delete – work in this case. For eg, how will you insert k=16 ?

6. In the class we briefly discussed the importance of load factor. It has other important applications also. Consider the corollaries 11.7 and 11.8 in the CLRS book that provide the expected number of probes for successful (search) and unsuccessful (insert) scenarios. Using this information, it is possible reverse engineer hash table parameters.

Suppose you want a hash table that can hold at least 1000 elements and you want successful searches to take no more than 4 probes on average.

a. What is the maximum load factor you can tolerate in your hash table?
b. If the table size must be prime, what is the smallest table size you can use?

7. Consider an open-address hash table with uniform hashing. Most programming languages do a rehashing when alpha = 3/4. Why do you think they do that? Using the corollaries above, and a load factor alpha = 3/4, what is the expected number of probes in an unsuccessful search? Successful search?

8. Given a large file, find the first non repeated word.

9. Consider the 2-sum problem where you are given an array A with n elements and a number S. You want to design an O(n) algorithm to find if there are two elements in A that sum to S.

10. Given two arrays, how do you find the common elements in O(n) time?

Data Structure Augmentation:

1. Suppose you are give a long file. Design an algorithm with time complexity of O(n lg k) [hint: quiz2] and O(n) space to find k words with highest frequency.

2. Design an efficient version of LRU cache that needs O(1) time for all operations.

3. Consider the augmented dictionary ADT where in addition to insert, delete and search, you also want to support three other operations: minimum, successor and sort. You are willing to relax the requirement that insert/delete/search needs to do O(1) time. What data structure would you choose for this ADT?

4. Recall the idea of radix sort where at each iteration, you sort the items based on a particular position. Suppose you decide to use hash table for sorting. Provide a detailed description about how you will go about designing the hash table for such a purpose – what is the hash function? What is its size? What is the collision resolution strategy? How will you maintain stability of sort etc?

5. Consider the problem of All nearest smaller values (see http://en.wikipedia.org/wiki/All_nearest_smaller_values). Consider a dynamic version of the problem where you get a sequence of integers and you are asked to augment an existing data structure so as to find the nearest smaller value in O(lg n) time. Hint: RBT.