CS314 Design and Analysis of Algorithms
Some practice problems for the assumed material

1 Data Structures

1. Suppose instead of the binary heap, we implement the priority queue operations using a k-ary ($k \geq 2$) heap.
   (a) How can the elements of the k-ary heap be arranged implicitly in an array? Specifically, where are the children of a node at position i of the array located?
   (b) Analyse the priority queue operations (insert, deletemin) implemented in a k-ary heap using the standard algorithms described for binary heap. What are the number of comparisons performed in the worst case for these operations?
   (c) Discuss about the optimal value of k to get the best performance on the number of comparisons used in these operations?
   (d) Suppose we are only interested in minimizing the number of moves (swaps) performed in the priority queue operations. What value of k would you choose?
2. The standard heap algorithms have the following time complexity for the number of comparisons in the worst case: insert - $\log n$; deletemin - $2 \log n$.
   (a) Develop an algorithm to insert an element in a heap that uses only $\log \log n$ comparisons in the worst case. How many swaps does your algorithm use?
   (b) Develop an algorithm to delete the minimum (and rebalance) from the heap that uses only $\log n + \log \log n + O(1)$ comparisons in the worst case. (Hint: Use the algorithm developed in (a)).
   (c) Develop an algorithm to delete the minimum (and rebalance) from the heap using only $\log n + \log \log \log n + O(1)$ comparisons in the worst case. How far can you carry this idea forward?
3. Suppose we want to maintain a sequence S of n numbers to support, besides the usual dictionary operations insert, search and delete, find(k, S) which finds the k-th smallest element in the sequence.
   How would you augment a balanced binary search tree (say an AVL tree) to support this operation in $O(\log n)$ time? Explain, how insert and delete operations can still be maintained in $O(\log n)$ time?

2 Graph Algorithms

1. In BFS what are the properties of the non-tree edges? What about in DFS? For example, show that two leaves of a DFS tree are not adjacent in the graph.
2. How do you determine whether a graph is connected? What is the complexity of your algorithm?
3. How do you determine whether a given graph is bipartite? What is the complexity of your algorithm¿
4. How do you determine whether a given graph has a cycle in $O(n)$ time (regardless of the number of edges in the graph)?
5. A bridge in a connected undirected graph is an edge whose removal disconnects the graph.
   (a) Prove than an edge is a bridge if and only if there is no cycle containing that edge.
   (b) Give an $O(n + m)$ algorithm to find all bridges in a connected undirected graph using depth first search.
6. A directed graph G has an Euler tour if there is a directed cycle that visits every edge exactly once (though vertices can be repeated several times).
   (a) Show that a directed graph has a directed cycle if and only if for every vertex, its indegree and outdegree are the same.
   (b) Give an $O(m + n)$ algorithm to find an Euler tour in a directed graph where every vertex has indegree and outdegree the same.
7. A cut vertex x in an undirected graph G is a vertex such that $G \setminus x$ is disconnected.
   (a) Given an undirected graph on n vertices and m edges, how will you find all the cut vertices? How much time does your algorithm take?
   (b) Suppose that the graph has no cut vertex, and consider the DFS tree of the graph. Show that the root can not have more than one child in the tree.

3 Sorting and Searching

1. Suppose I modify a given sorted list of $4n$ distinct numbers as follows:

   Keep elements in even positions (positions 2, 4, 6, ... $4n$) as they are. Form n disjoint pairs (i, j) on the odd numbered positions where $i = 2k + 1$ for some $k = 0$ to $n - 1$ and $j = 2k + 1$ for some $k = n$ to $2n - 1$.

   Now swap elements in positions i and j for every such pair. (I.e. every element in an odd numbered position in the first half of the array is swapped with *some* element in an odd numbered position in the second half of the array. No element is involved in more than one swap (i.e. the swaps are disjoint). You don't know these (i, j) pairs except that an element in an odd numbered position in the first half of the array is swapped with *some* element in an odd numbered position in the second half.

   Now given an element x, explain how you can determine whether or not x is in the (new reshuffled) array in $O(\log n)$ time.

2. Given two sets A and B of n elements each from a totally ordered set, give an efficient algorithm to report all the elements that are common to both A and B. If the integers in A are in the range 1 to 100, can you design a better algorithm?

3. Given two sets A and B of integers, and an integer x, give an efficient algorithm to determine whether $x = a+b$ for some $a \in A$ and $b \in B$. If the integers in A are in the range 1 to 100, can you design a better algorithm?

4. You are trying to estimate the limit of a database which is an unknown integer N, beyond which adding any data results in an error. Design an efficient algorithm that will determine N by trying the addition of some elements of data to the database. I.e. all you can do is to try the database with some number n of elements. You know whether you have crossed the limit by seeing whether or not you got the error. What is the complexity of your algorithm? I.e. How many trials do you need to do as a function of N?

5. Given an integer a and a positive integer n, describe a method to compute a^n using only multiplications. How many multiplications does your algorithm use? How many does it use to compute a^{100}?

6. Given a list of n numbers, give an efficient algorithm to determine whether they are all distinct.

7. Given a list of $2n$ distinct numbers, determine whether the list can be grouped into n groups of two elements each such that the sum of the pairs in each group is the same.

8. Give an algorithm to find the second largest element from a list of n numbers. Focus on minimizing the number of comparisons (Hint: there is an algorithm that takes $n + o(n)$ comparisons).

9. Give an algorithm to find the smallest and the largest element from a list of n numbers. How many comparisons does your algorithm use?

10. Suppose I think of an integer between 1 and n and you need to guess it by asking me questions that result in an yes or no answer. How many questions do you need?

    Suppose I lie at most once, how many comparisons do you need now to find the number?