Part 1: Simple trees, Operator trees, Tree traversals, Binary Search Trees, Priority Queues (heaps), Huffman Trees, AVL-Trees, B-Trees

Part 2: Asymptotic analysis, Algorithm analysis, Number representation, Bit manipulation

Part 3: Graph representations (Adjacency Matrix/List), DFS (with start/finish), BFS, Detecting DAGs, Detecting SCCs, Topological Sort

Part 4: Dijkstra’s algorithm, Prim’s algorithm, Kruskal’s algorithm, The Bellman-ford algorithm

Part 5: Hashtables, Sorting (Insertion, Selection, Heap, Merge, Quick)

About this set of questions. These questions are intended to give you practice solving problems on paper. Although they bear some resemblance to exam questions, they are not intended as a set of “practice midterm questions”, but as an aid in learning the ideas we have covered recently. They also are not guaranteed to be comprehensive, since each question aims to review only certain aspects of each topic.

As an incentive (not much of one, but better than nothing), you will receive 1 extra point for completing at least 35 of the questions presented below and turning in your solutions on paper at or before the exam. Be sure to write your name and login on the papers you hand in.

Do not use a computer while solving these problems – simply work on paper (or on a chalkboard). Feel free to consult with others when you get stuck on a question, but otherwise, try to work alone (after all, these are review questions). The questions vary in difficulty and are generally easier than homework questions. Although you may think you’ll find some questions easy, it is worth your while to complete them anyway for reinforcement (if they’re easy, they shouldn’t take long to answer anyway).
Part 1

1. **Simple Trees.** Given a simple binary tree, and two tree nodes A and B, how can you find the least common ancestor of A and B in $\Theta(lg n)$ using only constant storage (that means constant memory usage regardless of the size of the tree, and no node marking)? The least common ancestor of A and B is a node C that is the ancestor of both A and B that is furthest from the root of the tree.

2. **Operator Trees.** How can an operator tree with binary operators be converted into a linked-list such that the expression it represents can be evaluated only with left to right readings of the linked list representation?

3. **Tree Traversals.** Given the following tree, write the pre-order, in-order, and post-order traversals:

```
    A
   / \
  B   D
 / \ / \  
E  F G  C
```

4. **Binary Search Trees.** Write an iterative method that takes a BST and an int depth and prints out all elements in the BST at the specified depth. This method should not visit any elements with depth greater than the given depth, and should print out the elements with specified depth from right to left.

5. **Binary Heaps.** Given an empty minimum binary heap, perform insert with the following items with the given keys in order into the heap showing the array and tree-structure drawing of the heap after each insertion: 6, 1, 2, 9, 5, 4, 3, 0, 7

6. **Binary Heaps.** With the heap you built from the last question, perform the following extractMins from the heap, showing the array and tree-structure drawing of the heap after each removal: 3, 0, 5, 7

7. **Binary Heaps.** During a $\Theta(n)$ min-heap build-heap operation (in which we perform reHeapifyDown from the last items in the heap towards the beginning of the heap) the array within the heap contains the following elements in order: 7 5 6 1 2 3 4. If the build-heap operation is continued and reHeapifyDown calls are made on indices up to but not including index 1 and index 0 of the array, what does the array look like?

8. **Huffman Trees.** Given a huffman tree of $N$ symbols, each of which represents $L$ bits, assuming that all symbols are equally likely in the input, what are the bounds on the maximum and minimum compression ratio that can be achieved (a compression ratio is the number of bits in the original input to the number of bits that are to be output in the compressed output)?

9. **AVL Trees.** Starting with an empty AVL tree, insert elements into the tree with the following keys, showing each step and any rotations performed: 7 5 6 1 2 3 4.
10. **AVL Trees.** Given an AVL tree of height n, if an element with value k is inserted into the tree, what is the maximum number of rotate operations that would have to be performed to restore the AVL property? Give an example of this worst-case situation with a tree of height 7, show the key that is inserted, the sequence of rotations that must occur to restore balance, and the resulting tree.

11. **B-Trees.** Starting with an empty B-Tree of order 4, insert elements into the tree with the following keys, showing the insertion at each step and each split operation: 55 100 90 75 10 5 15 40 65 60 50 20 30 40.

12. **B-Trees.** Starting with the B-Tree that results from the previous question’s insertions, delete the elements with the following keys, showing the deletion at each step, and each merge and regroup operation: 90 75 10 5 20 40 55.

**Part 2**

1. **Asymptotic Analysis.** Is \( O(n^2) = \Theta(n^2) \cap \Omega(n^2) \)? Is \( \Theta(n^2) = O(n^2) \cap \Omega(n^2) \)? Why or why not?

2. **Asymptotic Analysis.** Data Structures (Into Java) 1.1, 1.2

3. **Asymptotic Analysis.** Sort the following running times such that if a running time \( f \) occurs before a running time \( g \), \( f \in O(g) \): 14n, 2^n, 6!, \( n^{1.6} \), \( lg n \), 12, 1000(\( lg n \)^2), \( n lg n \), \( n^3 \), \( lg n^2 \).

4. **Algorithm Analysis.** Give a tight asymptotic bound on the space used (memory allocated) in a run of Depth-First Search on an undirected graph \( G = (V, E) \) in terms of properties of the graph (number of vertices or edges, degree, minimum-degree, maximum-degree, connectivity, etc...). Give a similar bound on the space used in a run of Breadth-First Search on such a graph.

5. **Algorithm Analysis.** Give a tight asymptotic bound on the time to find a key \( k_{n+1} \) in a hashtable given a key \( k_n \) such that \( k_{n+1} > k_n \) as defined by some ordering property on all keys \( k_i \) such that \( \forall i \), \( k_{i-1} < k_i < k_{i+1} \). Assume the hash function distributes elements perfectly randomly.

6. **Algorithm Analysis.** Is an \( O(n) \) algorithm always better than an \( O(n^2) \) algorithm for some sufficiently large input size \( n \)? Why or why not?^1

7. **Number Representation.** For what values (generally speaking – not particular example values) of an int \( x \) will byte \( b = \text{byte}(x) \) result in a positive value of \( b \)? Why?

8. **Bit Manipulation.** If \( x \) is a positive int and \((x<<10)==0\), must \( x \) be a multiple of \( 2^{22} \). Why?^2

9. **Bit Manipulation.** If \( x \) is an int then is \((x&3)==0\) if and only if \( x \) is evenly divisible by 4? Why?^3

**Part 3**

1. **Adjacency Matricies.** Draw the digraph that corresponds to the matrix below, labelling nodes alphabetically starting at \( A \).

\[
\begin{pmatrix}
6 & 1 & 0 & 17 & 3 & 10 \\
1 & 2 & 3 & 8 & 11 \\
9 & 2 & 3 & 2 & 4 \\
6 & 1 & 17 & 3 & 4 \\
22 & 0 & 5 & 3 & 6 \\
12 & 1 & 0 & 2 \\
2 & 19
\end{pmatrix}
\]

---

^1PNH – CS61B Exam #2, Fall 2000
^2PNH – CS61B Exam #2, Fall 1999
^3PNH – CS61B Exam #2, Fall 2000
2. **Adjacency Lists.** Write the above digraph in its adjacency list representation.

3. **Depth First Search.** Perform a depth-first search on the above graph. Label start/finish times for nodes and tree/forward/back/cross edge indicators for edges. Start the search at node A and always select successors in alphabetically increasing order. What is the maximum size of the stack during this DFS run?

4. **Breadth First Search.** Perform a breadth-first search on the above graph and write the labels in the order you visit them. What is the maximum size of the queue during this BFS run?

5. **DAGs.** Given two DAGs, \( G_0 = (V_0, E_0) \) and \( G_1 = (V_1, E_1) \), is it possible that an edge \((u, v), u \in V_0, v \in V_1\) could cause the supergraph that includes \( G_0 \) and \( G_1 \) to not be a DAG? If another such edge were added, could the supergraph not be a DAG? Why or why not?

6. **SCCs.** Given the above graph, run the strongly connected components algorithm (showing its steps), and write the resulting list of strongly connected components.

7. **SCCs.** If a graph \( G = (V, E), |V| \geq 1 \) has \( N \) strongly connected components, and an edge \((u, v) \in E\) is removed, what are the upper and lower bounds on the number of strongly connected components in the resulting graph? Give an example of each boundary case.

8. **Topological Sort.** Using the topological sort algorithm on some DAG, what output would result if nodes were output in order of *increasing* finishing time? What would result if nodes were output in order of *decreasing starting* time?

---

**Part 4**

1. **Dijkstra’s Algorithm.** Given the digraph in Part 3, run Dijkstra’s algorithm starting at node A, show all updates made to the keys for nodes stored in the priority queue, all resulting shortest-path distances, and all resulting shortest-paths (as a list of edges) from A.

2. **Dijkstra’s Algorithm.** Suppose you use Dijkstra’s algorithm as follows to find shortest-paths in a graph with some negative weight edges:
   - Find the edge with the most negative weight, \(-C\)
   - Add \( C \) to the weights of all edges in the graph
   - Compute the shortest path desired using Dijkstra’s algorithm
   - Subtract \( C \) from each edge and recompute the actual shortest path length

   For any undirected graph \( G = (V, E) \), will this produce a correct result? If not, give a counterexample.

3. **Dijkstra’s Algorithm.** Given a *vertex-weighted* undirected graph \( G_v \) (that is, instead of edges having weights, vertices have weights and edges are unweighted), how would you run Dijkstra’s algorithm on \( G_v \) to give shortest-paths (based on vertex-weights)? You may need to change the format of the input or modify Dijkstra’s algorithm.

4. **Prim’s Algorithm.** Given the the matrix

\[
\begin{pmatrix}
6 & 1 & 0 & 17 & 3 \\
1 & 2 & 3 & 8 \\
& 3 & 2 \\
0 & 2 & 17 & 3 \\
17 & 3 & 17 & 5 & 3 \\
3 & 8 & 2 & 3 & 3
\end{pmatrix}
\]
that represents an undirected graph, run Prim’s algorithm starting at any node, show all updates made to the keys for nodes and edges stored in the priority queue and the resulting MST.

5. **Kruskal’s Algorithm.** Run Kruskal’s algorithm on the graph that you used for Prim’s algorithm, show the priority queue, the disjoint-sets, and the resulting MST. Will Kruskal’s algorithm always give the same MST as Prim’s algorithm given a graph $G$? If not, for which graphs will it not?

6. **The Bellman-Ford Algorithm.** Given a weighted DAG (which may include edges with negative weights), is it possible to reduce the number of iterations in Bellman-Ford and still yield a correct answer if edges are examined based on a topologically-sorted ordering? In which cases will a reduced number of iterations yield a correct answer? In which will it not?

## Part 5

1. **Hashtables.** Write a good hash function for an object that represents a chess board – that is, given a chess board with some pieces on it, produce some hash code value that uniquely (or nearly uniquely) identifies the board, its pieces, the type of pieces, the position of those pieces, etc... In the process of writing your hash function, you will need to describe the way the chess board itself is stored.

2. **Hashtables.** Suppose a function $f$ computes the hash code for a value $N$ that is $L$ bits long and returns an `int` value by returning the low-order 32 bits of $N$. For what values will this hash function be poor?

3. **Insertion Sort.** Run insertion sort on the following elements in an array showing each step of the sort and the resulting array: 17 0 2 12 6 8 1 10 9 11

4. **Selection Sort.** Run selection sort on the following elements in an array showing each step of the sort and the resulting array: 17 0 2 12 6 8 1 10 9 11

5. **Heap Sort.** Given an array of elements in reverse sorted order, will heap sort (using bottom up heap-construction) perform fewer or greater swap operations than a quick sort of the array (choosing the median value of the first, middle, and last as the pivot)? Which algorithm will perform fewer comparisons? Why?

6. **Merge Sort.** Run merge sort on the following elements in a linked-list showing each step of the sort, the splits that occur, the merges that occur, and the resulting list: 17 0 2 12 6 8 1 10 9 11

7. **Quick Sort.** Run quick sort on the following elements in an array showing each step of the sort, the pivot chosen, and the result, choosing the pivot to be the median of the first, middle, and last element of the current subarray: 17 0 2 12 6 8 1 10 9 11

8. **Quick Sort.** For what input array orderings will quick sort be slow if the first element is always chosen as the pivot? What if the middle element is always chosen as the pivot?