CSE 373 Final Exam Solution

Data Structures and Algorithms

University of Washington, Autumn 2019

This exam has 10 questions worth a total of 90 points and is to be completed in 110 minutes. Two double-sided, 8.5-by-11" sheets of notes are permitted. Electronic devices are prohibited. This exam is preprocessed by a computer when grading, so please **write darkly and write your answers inside the designated spaces. Write the statement below in the given blank and sign. You may do this before the exam begins.**

> "I have neither given nor received any assistance in the taking of this exam."

I have neither given nor received any assistance in the taking of this exam.

Signature: Evil Kevin

Mid.	Points
1	1
2	14
3	13
4	6
5	11
Total	45

Final	Points
6	2
7	18
8	12
9	3
10	10
Total	45

Name	Evil Kevin
Student ID	1234567890
Name of person to left	Robbie Weber
Name of person to right	Hannah Tang

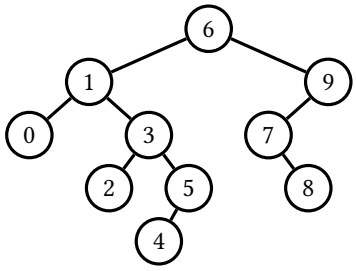
- **Work through the problems you are comfortable with first.**
- Not all information provided in a problem may be useful, and **you may not need all lines.**
- Take notes in any white space on this exam. Only the final answer region will be graded.
- Unless we specifically give you the option, the correct answer is not, 'does not compile.'
- ◯ indicates that only one circle may be filled-in.
- ☐ indicates that more than one box may be filled-in.
- For answers that involve filling-in a ◯ or ☐, fill-in the shape completely: ● or ■.

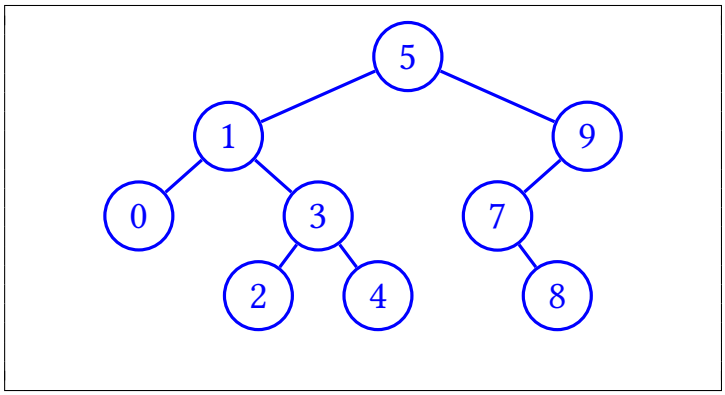
Designated Exam Relaxation Space

1. (1 pt) **So It Begins** Write the statement on the front page and sign. Write your name, ID, and the names of your neighbors. Write your name in the given blank in the corner of every other page. Enjoy a free point.

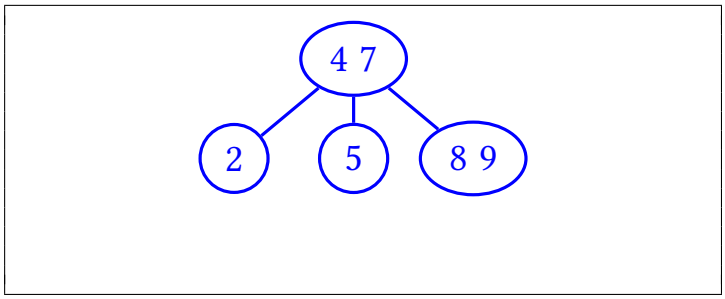
2. (14 pts) **Trees**

   (a) Draw the result of removing 6 from the binary search tree below using the procedure shown in class.

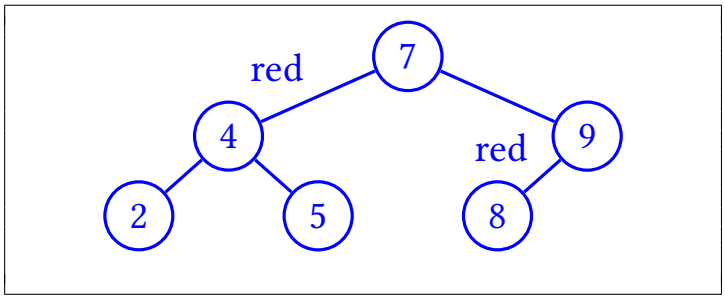




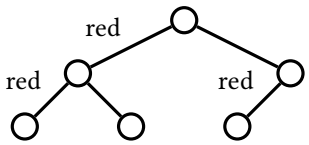
   (b) Draw the 2-3 tree that results from inserting these items in order: 8, 2, 4, 5, 7, 9.



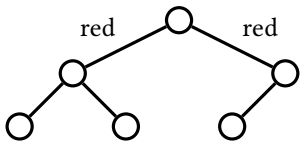
   (c) Draw the left-leaning red-black tree that results from inserting these items in order: 8, 2, 4, 5, 7, 9.



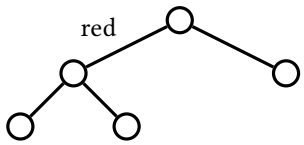
   (d) Select all valid left-leaning red-black tree structures.



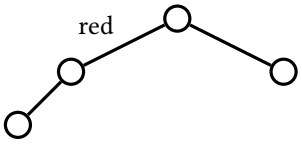
☐



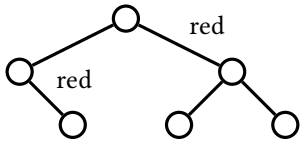
☐



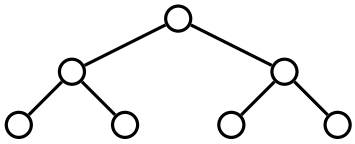
■



☐

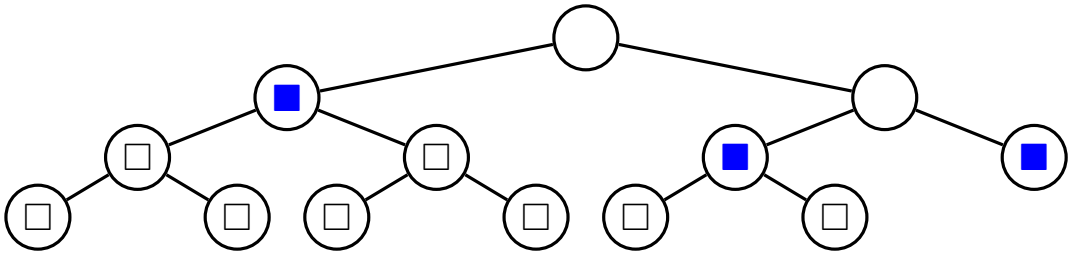


☐

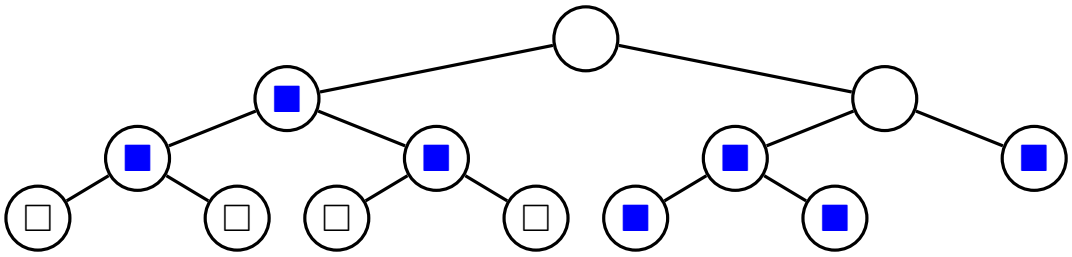


■

(e) If the largest value in a max-heap is in the root node and the second-largest value is in the root's right child, select all nodes where the **third-largest** value could be found. Assume all values are distinct.



(f) If the largest value in a max-heap is in the root node and the second-largest value is in the root's right child, select all nodes where the **fourth-largest** value could be found. Assume all values are distinct.



3. (13 pts) **Algorithm Analysis** Give the worst-case order of growth of the runtime in $\Theta(\cdot)$ notation as a function of N. Your answer should be simple with no unnecessary leading constants or summations.

(a) $\Theta(N$ _____ $)$

```
static void accumulate(int N) {
    if (N > 1) {
        int M = 0;
        for (int i = 1; i < N; i += 2) {
            M += 1;
        }
        accumulate(M);
    } }
```

(b) $\Theta(N \log N$ _____ $)$

```
static void threeWay(int N) {
    if (N > 1) {
        for (int i = 1; i < N / 2; i += 1) {
            System.out.println(i);
        }
        threeWay(N / 3);
        threeWay(N / 3);
        threeWay(N / 3);
        for (int i = 1; i < N / 2; i += 1) {
            System.out.println(i);
        }
    } }
```

(c) Assume that `TreeSet` uses a left-leaning red-black tree and the N `input` values are all distinct.

```
static void treeSort(LinkedDeque<Integer> input) {
    TreeSet<Integer> tree = new TreeSet<>();
    for (int i = 0; i < input.size(); i += 1)
        tree.add(input.get(i));
    for (int i = 0; i < input.size(); i += 1)
        System.out.println(tree.pollFirst()); // Remove the min
}
```

$\Theta($ N^2 $)$

(d) Suppose we know that the order of growth of a function is in $\Theta(\log N)$. Mark all of the true expressions.

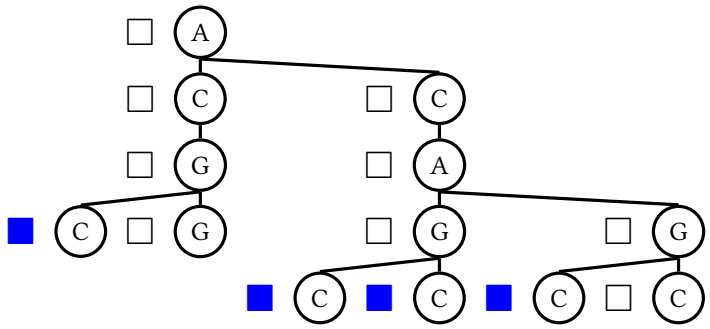
■ $O(\log N)$	■ $\Theta(\log N)$	■ $\Omega(\log N)$
■ $O(N)$	☐ $\Theta(N)$	☐ $\Omega(N)$
■ $O(N^2)$	☐ $\Theta(N^2)$	☐ $\Omega(N^2)$

(e) Suppose we know that the order of growth of a function is in $O(N)$. Mark all of the true expressions.

☐ $O(\log N)$	☐ $\Theta(\log N)$	☐ $\Omega(\log N)$
■ $O(N)$	☐ $\Theta(N)$	☐ $\Omega(N)$
■ $O(N^2)$	☐ $\Theta(N^2)$	☐ $\Omega(N^2)$

4. (6 pts) **Ternary Search Tries**

(a) Mark the ☐ to the left of each node in the ternary search trie (TST) for ACC, CAC, CAGC, CC.



(b) Give the best-case and worst-case runtime for `contains` on a TST in terms of R, the size of the alphabet, L, the length of the search string, and N, the size of the TST. **Assume the search string is in the TST.**

Best case $\Theta(L)$ **Worst case** $\Theta(LR)$

5. (11 pts) **Specialized Data Structures** Before hash tables, we introduced `DataIndexedStringSet`, which maintained a `boolean[] data`. Suppose we optimize the structure by using the hash code mod `data.length`. To add a `String s`, assign `data[s.hashCode() % data.length] = true`.

(a) Select the pairs of string inputs that reveal the issue with this optimization. Assume that `String.hashCode` returns the length of the string and that `data.length` is 3.

☐ `a, ab` ☐ `a, abc` ■ `a, abcd` ☐ `ab, ab` ☐ `ab, abc` ☐ `ab, abcd`

Worst-case runtime for `contains` assuming a fixed-length underlying array. $\Theta($ 1 $)$

Worst-case runtime for `contains` assuming a resizing underlying array. $\Theta($ 1 $)$

Suppose we also hash the last character of the string in addition to the `String.hashCode` described previously so that two values are reassigned when a string is added to the set. (Assume non-empty strings.)

```
public void add(String s) {
    data[s.hashCode() % data.length] = true; // s.hashCode() == s.length()
    data[s.charAt(s.length() - 1) % data.length] = true;
}
public boolean contains(String s) {
    return data[s.hashCode() % data.length]  // s.hashCode() == s.length()
        && data[s.charAt(s.length() - 1) % data.length];
}
```

Assume that `'a' = 0`, `'b' = 1`, `'c' = 2`, `'d' = 3`, `'e' = 4`, `'f' = 5`, `'g' = 6`, `'h' = 7`.

(b) Select the true values in the `data` array after adding the strings: `abc`, `abcd`.

	0	1	2	3	4	5	6	7
data	☐	☐	■	■	■	☐	☐	☐

For the following parts, refer to this `data` array representing the set of strings `{bab, ahhh}`.

	0	1	2	3	4	5	6	7
data	☐	■	☐	■	■	☐	☐	■

(c) Select all inputs to `contains` that return true instead of false.

■ `b` ☐ `bb` ☐ `bag` ■ `eebe` ☐ `beebe` ☐ `agf` ☐ `ggfg` ☐ `ga`

(d) Suppose we add a `Set<String>` to the class. In `add`, if any of the `data` indices we want to reassign are already true, then we will also add the string to the internal set. In `contains`, if all of the `data` indices we want to check are true, then only return true if the input is also in the internal set. Does this work?

◯ Yes

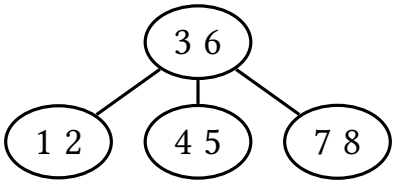
● No, because `contains(bab)` will return false rather than true.

Suppose that this `Set<String>` is implemented using an R-way `TrieSet` with an internal `CharMap.get` that takes $\Theta(\log R)$ time. Give the best-case and worst-case runtime for `TrieSet.contains` in terms of R, the size of the alphabet, L, the length of the search string, and N, the size of the `TrieSet`.

Best case $\Theta(\log R)$ **Worst case** $\Theta(L \log R)$

6. (2 pts) **Traversals** Unscramble the 5 lines to perform an inorder traversal on a 3-tree.

```
class ThreeTreeNode {
    int keyOne, keyTwo;
    ThreeTreeNode left, middle, right;
    void inorder() {
1       if (left != null) left.inorder();
2       System.out.println(keyOne);
3       System.out.println(keyTwo);
4       if (right != null) right.inorder();
5       if (middle != null) middle.inorder();
    } }
```



Line Numbers: 1 2 5 3 4

7. (18 pts) **Sorting** Assume the naïve version of each sorting algorithm as presented in lecture.

(a) Give the two arrays that will be merged by the final step of merge sort on `[8, 2, 2, 4, 5, 5, 7, 9]`.

Left: 2 2 4 8 Right: 5 5 7 9

(b) For the specific array `[8, 2, 2, 4, 5, 5, 7, 9]`, which sort will be the fastest in nanoseconds?

◯ Selection sort ◯ Heapsort ◯ Merge sort ● Insertion sort ◯ Quicksort

(c) Which sorts can be used to determine whether an array has duplicates in worst-case $O(N \log N)$ time?

1. No transformation of the input array is necessary.
2. ☐ Selection sort ■ Heapsort ■ Merge sort ☐ Insertion sort ☐ Quicksort
3. Scan across the sorted array to identify duplicate pairs of items.

Is a stable sort necessary? ◯ Yes ● No ◯ Not enough information

(d) Suppose we have a sorted list of N unique integers and add 2 more integers to the end. Give the **exact** least and greatest possible number of inversions in the list.

Least: 0 Greatest: $2N + 1$

(e) Choose the partitions in quicksort to ensure stability if we pivot around the **rightmost item** p.

◯ $> p$ ◯ $\geq p$ ◯ $< p$ ● $\leq p$ p ● $> p$ ◯ $\geq p$ ◯ $< p$ ◯ $\leq p$

(f) Suppose we have a partitioning algorithm that takes $\Theta(N^2)$ time where N is the length of the array. Give the best-case runtime of quicksort using this partitioning algorithm.

Best-case runtime: $\Theta(N^2)$

(g) Consider a heap optimization that achieves $O(\alpha(N))$ runtime for removing the min or max item on any input, where $\alpha(N) \notin \Omega(\log N)$. Give a runtime argument for why this optimization is impossible.

Such runtime would violate the sorting lower bound for heapsort!

(h) Show the output of counting sort on the **rightmost digit** for the input `[234, 834, 286, 495, 874]`.

234 834 874 495 286

(i) Show the output after two passes in LSD radix sort for the input `[234, 834, 286, 495, 874]`, i.e. after applying counting sort on the rightmost digit and the middle digit.

234 834 874 286 495

(j) Suppose we replace the counting sort used in radix sort with selection sort, resulting in two algorithms: LSD Radix Selection Sort and MSD Radix Selection Sort. Are these sorts guaranteed to be correct?

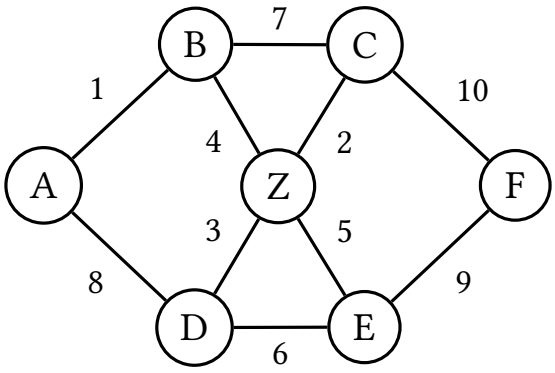
◯ Yes, LSD Radix Selection Sort will return a correct sort for any input.

● No, LSD Radix Selection Sort can fail because: Selection sort is unstable.

● Yes, MSD Radix Selection Sort will return a correct sort for any input.

◯ No, MSD Radix Selection Sort can fail because: _____

8. (12 pts) **Graphs** Consider this undirected graph, G, with 7 vertices and 10 distinct weighted edges.



(a) Give the edges in the unique minimum spanning tree (MST) of G in the order added by each algorithm. Refer to each edge by **weight**, e.g. 1 refers to the edge A–B. You may not need all the blanks.

Prim's algorithm from F: 9 5 2 3 4 1 ___ ___ ___ ___

Kruskal's algorithm: 1 2 3 4 5 9 ___ ___ ___ ___

(b) What is the minimum weight edge on the cut between ABCZ and the rest of G? Weight: 3

(c) Say we find the shortest paths tree (SPT) from B. Does this SPT share the same edges as the MST of G?

○ Yes ● No, the edge with weight 10 is in the SPT from B but not the MST of G.

(d) If we run DFS from B, which edge will we use to reach Z? Visit neighbors alphabetically. Weight: 2

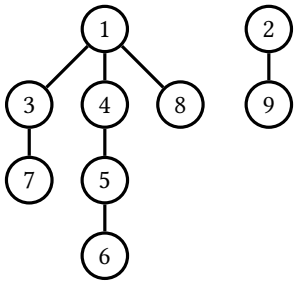
(e) Consider a directed graph with distinct, non-negative edge weights. Given a starting vertex s with at least one outgoing edge, the outgoing edge with the smallest weight must be in the SPT from s.

● Always true ○ Sometimes false ○ Always false

(f) In a undirected graph with distinct, non-negative edge weights, the smallest-weight edge is in the MST.

● Always true ○ Sometimes false ○ Always false

9. (3 pts) **Disjoint Sets** Consider the following tree structures.



(a) The above tree structures can be a part of which of the following data structures?

■ Quick union (QU) ☐ Weighted QU (WQU) by size ☐ WQU with Path Compression

(b) Suppose the above tree structures are part of a weighted quick union (by size) with path compression. What is the height of the resulting component after `connect(9, 6)`?

Height: 2

10. (10 pts) **Algorithms** Recall that the energy of a pixel is defined as the non-negative difference between the pixel's color and the colors of its neighbors. In the homework, we used this to find the lowest-cost seam. Suppose we now want to implement `findBlemish`, which returns the **highest-energy pixel** in a picture.

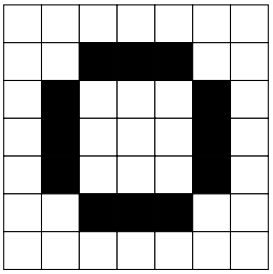
    (a) Give a tight lower bound for the time complexity of `findBlemish` in terms of N, the number of pixels in the picture.

    $\Omega(N)$

    (b) Describe an $O(N)$ time algorithm for `findBlemish`.

    Scan across all N pixels in the image and return the pixel with the maximum energy.

    Today, images are very high resolution. Blemishes are no longer single pixels but rather **patches** of low-energy pixels enclosed by high-energy pixels. Assume that there are only two types of pixels: high-energy (black) and low-energy (white).



    (c) Give a crisp and concise English description of how to find **all high-energy patches** using algorithm ideas from this course. Give your algorithm as a **numbered list of steps**. Then, briefly justify the runtime in terms of H, the number of high-energy pixels, and L, the number of low-energy pixels.

1. Construct an unweighted, undirected graph where vertices represent high-energy pixels and edges between adjacent, high-energy pixels by scanning across the image.
2. Consider each vertex unmarked.
3. For each unmarked vertex, run DFS and mark vertices as we visit them.
4. When an edge points to a marked vertex (aside from where we came from), stop the DFS and add this cycle to the result.
5. If we ever fail to find a neighbor (aside from where we came from), stop the DFS but do not add the cycle to the result.

$\Theta(H + L)$ overall since it takes $\Theta(H + L)$ time to generate the graph and $\Theta(H)$ time to run DFS since each high-energy pixel is adjacent to at most 8 other pixels.