COS 226	Algorithms and Data Structures	Fall 2021

Final Exam Solutions

1. **Initialization.** Don't forget to do this.

2. **Memory usage.**

   (2.1) 32 bytes

   - 16 bytes object overhead
   - 8 bytes for string reference
   - 4 bytes for `int`
   - 4 bytes padding

   (2.2) $\sim 40n$ bytes

   - $\sim 32n$ bytes for the n `Suffix` objects
   - $\sim 8n$ for the array of n references

3. **String operations.**

   (3.1) $\Theta(n^4)$
   String concatenation takes time proportional to the length of the resulting string, so the running time is $\Theta(1 + 2 + 3 + \ldots + n^2) = \Theta(n^4)$.

   (3.2) $\Theta(n^3)$
   String concatenation takes time proportional to the length of the resulting string, so each iteration of the inner loop takes $\Theta(1{+}2{+}3{+}\ldots{+}n) = \Theta(n^2)$ *time, for a total of* $\Theta(n^3)$ *time. The outer loop, which repeatedly appends string of length n, takes* $\Theta(n{+}2n{+}3n{+}\ldots{+}n^2) = \Theta(n^3)$ *time as well.*

   (3.3) $\Theta(n^2)$
   Appending each character to the end of a `StringBuilder` *takes* $\Theta(1)$ *amortized time. So, starting from an empty* `StringBuilder`, *appending* n^2 *characters takes* $\Theta(n^2)$ *time.*

4. **String sorts.**

   (4.1) *LSD radix sort (after 1 pass)*

   (4.2) *3-way radix quicksort (after the first partitioning step)*

   (4.3) *LSD radix sort (after 2 passes)*

   (4.4) *MSD radix sort (after the second call to key-indexed counting)*

   (4.5) *MSD radix sort (after the first call to key-indexed counting)*

5. **Graph search.**

   (5.1) 0 2 5 4 6 3 8 9 7 1

   (5.2) 5 2 8 9 3 6 1 7 4 0

   (5.3) No. The reverse of the DFS postorder (0 4 7 1 6 3 9 8 2 5) is a topological order.
   A digraph has a topological order if and only if it has no directed cycle.

   (5.4) 0 2 4 5 6 7 3 9 1 8

6. **Minimum spanning trees.**

   (6.1) 0 10 20 40 50 80 100 110

   (6.2) 10 20 50 40 80 0 100 110

7. **Shortest paths.**

   (7.1) 0 1 3 2 5 4
   Dijkstra's algorithm relaxes the vertices in increasing order of distance from s.

   (7.2) 0 3 1 2 4 5
   The topological order happens to be unique (because of the path $0 \to 3 \to 1 \to 2 \to 4 \to 5$).

   (7.3) 110, pass 2

8. **Self-adjusting data structures.**
   This question was based on a guest lecture in Fall 2021.

   (8.1) ☐ ☒ ☐ ☒

   (8.2) ☒ ☐ ☒ ☒

   (8.3) ☒ ☐ ☐ ☒

   (8.4) Bob Tarjan

9. **Dynamic programming.**

   (9.1) A C D H L
   The last three letters can be permuted in any order.

   (9.2) $\Theta(n)$

10. **Ternary search tries.**

(10.1) I, IN, OF, TIP, TRUE, TRY

(10.2) JADWIN, MATHEY, NASSAU

(10.3) ☒ ☐ ☐ ☐

11. **Data compression.**

(11.1) S P A R S E

(11.2) C C B B C C A D

(11.3) 41 42 81 83 82 85 80

(11.4) C A B
The compression ratios for A, B, and C are 16/255, 8, and 8/255, respectively.

(11.5) A B C
The compression ratios for A, B, and C are 1.4/8, 1.8/8, and 2/8, respectively.

12. **Min-weight crossing edge.**

(12.1) 0, 1, 4, 5 or 2, 3, 6, 7

(12.2) Remove edge $e = v$–w from the MST. This defines a cut, with the vertices in the connected component containing v on one side and the vertices in the connected component containing w on the other. This cut achieves our goal:

- By construction of the cut, e is a crossing edge.
- No other crossing edge f could have smaller weight because, if it did, we could replace e with f in our MST and obtain a strictly lighter spanning tree—a contradiction.

To construct the cut efficiently:

- Create a new edge-weighted graph H with V vertices, adding all edges in the MST except e.
- Run DFS in H from either vertex v or w.
- The marked vertices define one side of the cut.

In this application, DFS takes $\Theta(V)$ time because the number of edges in H is $V - 2$, not E.

Alternative solutions. There are a few variants that also work:

- Run DFS from any vertex—it doesn't need to be v or w.
- Use BFS instead of DFS.
- Instead of creating H, run DFS in the MST graph, but modify DFS to skip over edge e.
- When performing the graph search from either v or w, consider only those edges whose weight is strictly less than the weight of e. (This might produce a different cut than the approach discussed earlier.)

13. **Writing seminar preferences.**

The key idea is to treat the preferences for each student as an 8-digit number over an alphabet of size m. To check for duplicates:

- Sort the n numbers using LSD radix sort.
- Check adjacent entries for duplicates.

Sorting takes $\Theta(m+n)$ time because $R = m$ and the number of characters per string is a constant. Checking adjacent entries takes a total of $\Theta(n)$ time because comparing two strings, each of length 8, takes $O(1)$ time.

Partial-credit solutions.

- *MSD radix sort.* While it makes $\Theta(m+n)$ calls to `charAt()` in the worst case, it can still take $\Theta(mn)$ time, e.g., if there are $\Theta(n)$ pairs of students with equal preferences lists.
- *Compare-base sorting.* Mergesort (or heapsort) makes $\Theta(n \log n)$ compares in the worst case. Each compare takes constant time, so the overall running time for sorting is $O(n \log n)$.
- *3-way radix quicksort.* All of the partitions might be degenerate, which could lead to $\Theta(n^2)$ time in the worst case. Even probabilistically, the expected running could be $\Theta(n \log n)$ if $\Theta(n)$ students have different first choices.
- *Multiway trie.* Inserting the n strings into an R-way trie uses $\Theta(Rn)$ space (and time). In this application, $R = m$, which leads to $\Theta(mn)$ space (and time), not $\Theta(m+n)$.