CSC 226 MIDTERM EXAM

NAME: ________________________ STUDENT NO: ________________

Instructor: Nishant Mehta Duration: 80 minutes

NOTES:

1. THIS EXAM PAPER SHOULD HAVE 9 PAGES. COUNT THE NUMBER OF PAGES IN THIS EXAM PAPER BEFORE BEGINNING TO WRITE. REPORT ANY DISCREPANCY IMMEDIATELY TO THE INVIGILATOR.
2. THIS EXAM IS CLOSED BOOK; NO NOTES OR CALCULATORS ALLOWED.
3. ANSWER ALL QUESTIONS ON THE EXAM PAPER. USE THE BACK IF NEEDED.
4. SCRATCH PAPER IS AVAILABLE FROM THE INVIGILATORS.
5. THERE ARE A TOTAL OF 75 MARKS.

Question	Potential Marks	Actual Marks
1	16	
2	14	
3	18	
4	13	
5	14	
Total:	75	

1. Sorting and Selection

   (a) [4 marks] You encounter a coin with a strange property. After every flip, the Heads side becomes lighter. For the first flip, the probability of heads is 1/2. For the second flip, the probability of heads is 1/4. And, in general, for the k^{th} flip, the probability of heads is $1/(2k)$. What is the expected number of heads if you flip the coin n times? You may assume that n is large and use asymptotic notation.

   Solution: Let X_k be a Bernoulli random variables that takes the value 1 if the k^{th} coin flip is Heads and 0 otherwise. Thus, $P(X_k = 1) = \frac{1}{2k}$.
   Then the expected number of heads after n coin flips is

$$\mathsf{E}\left[\sum_{k=1}^{n} X_k\right] = \sum_{k=1}^{n} \mathsf{E}\left[X_k\right] = \sum_{k=1}^{n} P(X_k = 1) = \sum_{k=1}^{n} \frac{1}{2k}.$$

   This summation is approximately equal to

$$\frac{1}{2}\int_1^n \frac{1}{x}dx = \frac{1}{2}\left(\ln(n) - \ln(1)\right) = O(\log n).$$

   (b) [2 marks] In Quicksort, we call a pivot a "good" pivot if it falls within the middle 1/2 of the elements (if they had been sorted). Recall that Randomized Quicksort always selects the pivot uniformly at random. What is the probability of the event that *both* the first and second pivots selected by Randomized Quicksort are good pivots?

   Solution: For each pivot, the probability that it is good is $\frac{1}{2}$. So, the probability that both are good is $\frac{1}{2} \cdot \frac{1}{2} = \frac{1}{4}$.

   (c) [2 marks] Using $\mathrm{O}(\cdot)$ asymptotic notation, what is the average-case runtime of Randomized Quicksort?

   Solution: $\mathrm{O}(n \log n)$

   (d) [2 marks] Using $\Omega(\cdot)$ asymptotic notation, what is the worst-case runtime of Randomized Quicksort?

   Solution: $\Omega(n^2)$

(e) [6 marks] Let x be an array of n distinct integers, for some n divisible by 4. Originally, all the elements were in order, and the sorting algorithm knows this fact. However, the array becomes "corrupted" as follows.

- First, an even index i is selected uniformly at random from $\{0, 2, \ldots, n/2 - 2\}$ (the first half of the array), a fair coin is flipped, and $x[i]$ is swapped with $x[n/2 + i]$ *if and only if* the coin lands Heads.
- Next, an odd index j is selected uniformly at random from $\{1, 3, \ldots, n/2 - 1\}$ (the second half of the array), a fair coin is flipped, and $x[j]$ is swapped with $x[n/2 + j]$ *if and only if* the coin lands Heads.

The sorting algorithm knows that precisely one even index and one odd index is selected in this way, but it does not know which indices were selected, nor whether each swap happened or not.

Considering the information that the algorithm has, give an exact lower bound on the number of comparisons the sorting algorithm requires in the worst case (do not use $\Omega(\cdot)$ notation).

Solution: We first consider the even indices in the array, which consist of $n/2$ elements, or $n/4$ pairs $(0, n/2), (2, 2 + n/2), \ldots, (n/2 - 2, n - 2)$. Either a swap happened for precisely one of these pairs, or no swap happened at all. In order to figure out if a swap happened and fix it, we must compare $x[0]$ with $x[n/2]$, compare $x[2]$ with $x[2 + n/2]$, and so on (for a total of $n/4$) comparisons. If we fail to do even one such comparison, we might miss a swap (if one happened).

Similarly, for the odd indices, we need to do $n/4$ comparisons to ensure that we find a swap if one happened. The total number of comparisons is thus $n/2$ in the worst case.

2. Divide and Conquer

(a) [6 marks] Suppose that we are using QuickSelect with the median-of-medians pivot, where the latter medians are over groups of size 11. In the first call to QuickSelect, the array contains n elements. If $n/11$ is an odd integer, what is the minimum number of these elements that can be *greater than or equal to* the median-of-medians?

Solution: Since the group size is 11, there are $n/11$ groups and hence $n/11$ medians. The number of medians that are greater than or equal to the median-of-medians is $\left\lceil \frac{n/11}{2} \right\rceil = \frac{\frac{n}{11}+1}{2}$. Consider one such median m. The median m belongs to a group of 11 elements, and within m's group there are an additional 5 elements that are greater than or equal to m and hence that also are greater than or equal to the median-of-medians; if we also count m, then this group has 6 elements which are greater than or equal to the median-of-medians. Thus, there are at least $6\left(\frac{\frac{n}{11}+1}{2}\right) = \frac{6n}{22} + 3$ elements that are greater than or equal to the median-of-medians.

(b) [2 marks] Continuing from part (a), suppose that for any n, the number of elements less than the median-of-medians pivot is at least δn, and the number of elements greater than the median-of-medians pivot also is at least δn (for some $\delta < 1$). Let c be an appropriately chosen positive constant. Which of the following is the correct recurrence relationship for the runtime $T(n)$?

A. $T(n) = T(10n/11) + T((1-\delta)n) + cn$
B. $T(n) = T(10n/11) + T(\delta n) + cn$
C. $T(n) = T(n/11) + T((1-\delta)n) + cn$
D. $T(n) = T(n/11) + T(\delta n) + cn$
E. $T(n) = T((1-\delta)n) + cn$
F. $T(n) = T(\delta n) + cn$

(c) [4 marks] Consider running QuickSelect with the median-of-medians pivot for groups of size $\sqrt{n}$. Briefly explain why this algorithm's worst-case runtime is not $O(n)$.

Solution: Consider the begining of the algorithm, before any recursion has taken place. There are $n/\sqrt{n} = \sqrt{n}$ groups, each of size $\sqrt{n}$. Finding the median of each group via sorting costs $\sqrt{n}\log\sqrt{n}$, and there are $\sqrt{n}$ such groups, so the total cost to find all of these medians is $\sqrt{n}\cdot(\sqrt{n}\log\sqrt{n}) = n\log n^{1/2} = \frac{1}{2}n\log n = \Omega(n\log n)$.

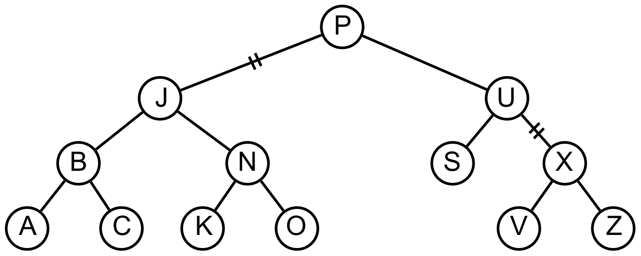
(d) [2 marks] Consider the recurrence relationship $T(n) = T(\alpha n) + T(\beta n) + 3n$, where α and β both are strictly positive and $\alpha + \beta < 1$. For all $n < 10$, assume that $T(n)$ is upper bounded by 50. Which of the following is the best (i.e. smallest) asymptotic upper bound on $T(n)$?

A. $O(n/\log n)$ **B.** $O(n)$ C. $O(n\log\log n)$ D. $O(n\log n)$ E. $O(n^2)$

3. Balanced Binary Search Trees

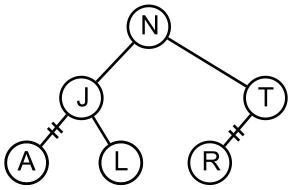
(a) [4 marks] For each of the two trees below, indicate whether or not it is a valid left-leaning red-black tree. If any tree is not, explain why not. Red links are indicated by two short, parallel lines passing through an edge.

First tree:



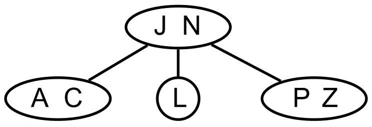
Solution: Not Valid. There is a right-leaning red link from "U" to "X".

Second tree:

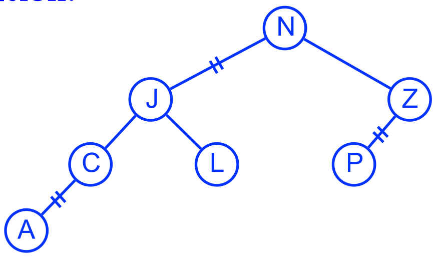


Solution: Not Valid. The path from N to L has two black links, while each of the other paths has only one black link. Hence, perfect black balance does not hold.

(b) [3 marks] Draw the left-leaning red-black tree corresponding to the following 2-3 tree. Please indicate red links as was done in part (a).



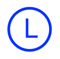
Solution:



(c) [5 marks] Starting from an empty 2-3 tree, show the tree after each of the following insertions: L, E, M, O, N. During each insertion, be sure to indicate the intermediate stages of the tree.

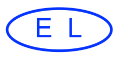
After L:

Solution:



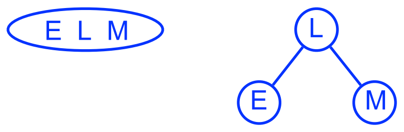
After E:

Solution:



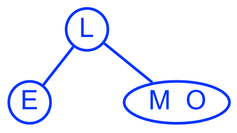
After M:

Solution:



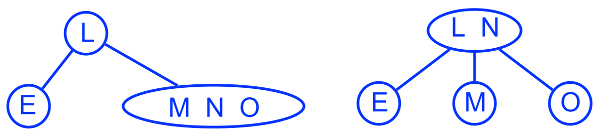
After O:

Solution:



After N:

Solution:



(d) [6 marks] *For this question, use the convention that the height of a tree is the length of the longest root-to-leaf path. Thus a tree with only a root node has height 0.*

Consider the following "Ternary AVL Tree" (for which each internal node has exactly three children): For each internal node, all the heights of the three child subtrees are within 1 of each other. That is, for any internal node x and for a height function H that maps a node to the height of the node's corresponding subtree, we have

$$\max\{H(x.\text{left}), H(x.\text{mid}), H(x.\text{right})\} - \min\{H(x.\text{left}), H(x.\text{mid}), H(x.\text{right})\} \leq 1.$$

Define $N(h)$ to be the minimum number of elements that can be stored in a Ternary AVL Tree of height h. Prove that there is a positive constant c and a constant $k > 1$ for which, for all $h \geq 0$, it holds that $N(h) \geq c \cdot k^h$. You do not have to find the largest k for which this holds.

Solution:

We use proof by induction

Base cases

$N(0) = 1$ and $N(1) = 2$. Let $c = 1/k$ for some k to be set later.

Then $N(0) = 1 \geq ck^0 = c$ and $N(1) = 2 \geq ck = 1$ are both true.

Inductive step

Assume the induction hypothesis: $N(\ell) \geq ck^\ell$ for all $\ell < h$.

We prove that $N(h) \geq ck^h$.

First, $N(h) = 1 + N(h-1) + 2N(h-2)$.

From the induction hypothesis,

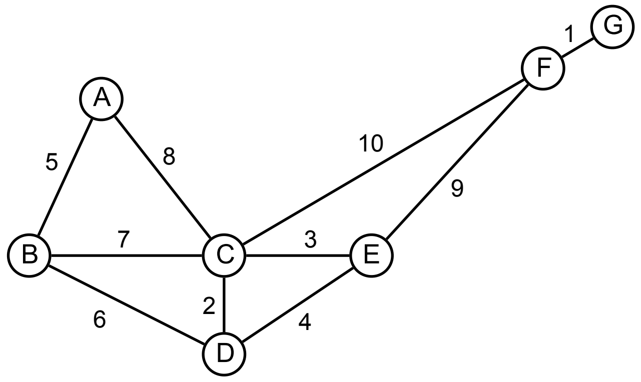
$$N(h) \geq 1 + ck^{h-1} + 2ck^{h-2} \geq ck^{h-1} + 2ck^{h-2}.$$

Thus, if it holds that $ck^{h-1} + 2ck^{h-2} \geq ck^h$, then we are done.

Dividing by ck^{h-2}, this is equivalent to $0 \geq k^2 - k - 2$, which is satisfied for $k = 2$.

4. Minimum Spanning Trees

(a) [6 marks] Suppose that Prim's algorithm is run on the following weighted graph, with vertex A as the initial vertex. List, in order, the sequence of edges added by the algorithm.



Solution:

(A,B)
(B,D)
(C,D)
(C,E)
(E,F)
(F,G)

(b) [2 marks] Consider the eager version of Prim's algorithm; this version uses a priority queue of edges that contains at most 1 edge for each vertex that is yet to be included in the tree being built up. Using big-O notation in terms of $|V|$ and $|E|$, what is the worst-case runtime of this algorithm? How does this runtime compare to the lazy version of the algorithm? Assume that both algorithms use a binary heap.

Solution: Eager: $O(E \log V)$. Lazy: $O(E \log E)$. Lazy is worse asymptotically.

(c) [5 marks] Assume G is a connected graph with distinct edge weights. Recall the Cut Property Theorem:

> Let A be a subset of the edges of the minimum spanning tree of G. If $(S, V \setminus S)$ is a cut for which no edge of A crosses the cut, then the minimum weight edge crossing the cut is contained in the minimum spanning tree.

Using the Cut Property Theorem, prove that the edges added by Kruskal's algorithm always belong to the minimum spanning tree.

Solution: Let A be the current set of edges in the forest being built up by Kruskal's algorithm. Let $e = (u, v)$ be the next edge added by Kruskal's algorithm. Since Kruskal's algorithm never adds an edge that induces a cycle, we have that $u \in C_i$ and $v \in C_j$, where C_i and C_j are distinct components in the subgraph induced by A. Let $S = C_i$ and consider the cut $(S, V \setminus S)$. Observe that e is a crossing edge for this cut since $u \in S$ and $v \in V \setminus S$. Moreover, e has the minimum weight among all such crossing edges since, if there were a crossing edge of lower weight, Kruskal's algorithm would have selected this crossing edge. Thus, by the Cut Property Theorem, e is contained in the minimum spanning tree.

5. Union-Find

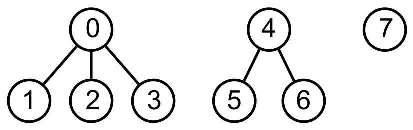
(a) [2 marks] Consider two versions of Kruskal's algorithm:

- The first one uses Weighted Quick-Union *without* Path Compression.
- The second one uses Weighted Quick-Union *with* Path Compression.

When viewed in terms of big-O notation, is the worst-case runtime of the version of Kruskal's algorithm with Path Compression lower, the same, or higher than the worst-case runtime of the version of Kruskal's algorithm without Path Compression? Please justify your answer.

Solution: The same (because the cost is dominated by operations on the priority queue or initial sorting, which costs $O(E \log E)$)

(b) [6 marks] Below is a forest of rooted trees. Fill in the values of the array id[].



id[]

0	1	2	3	4	5	6	7
0	0	0	0	4	4	4	7

Consider the following two lines of code:

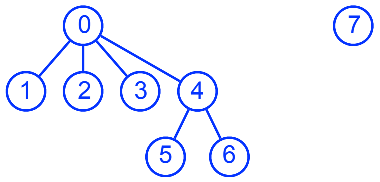
1: If(CONNECTED$(1, 6) == 0)$ UNION$(1, 6)$

2: If(CONNECTED$(6, 7) == 0)$ UNION$(6, 7)$

For the following questions, assume that the version of Union-Find being used is Weighted Quick-Union with Path Compression. Assume that Path Compression can occur only immediately after a call to FIND.

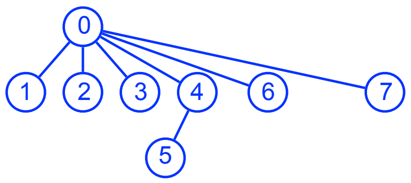
Show how the forest looks after line 1.

Solution:



Show how the forest looks after line 2 (after already having run line 1).

Solution:



(c) [2 marks] What is the worst-case runtime for a single call to CONNECT when using Quick-Union?

A. $O(1)$ B. $O(\log\log n)$ C. $O(\log n)$ D. $O(\sqrt{n})$ E. $O(n)$

(d) [2 marks] What is the worst-case runtime for a single call to CONNECT when using Weighted Quick-Union?

A. $O(1)$ B. $O(\log\log n)$ C. $O(\log n)$ D. $O(\sqrt{n})$ E. $O(n)$

(e) [2 marks] Briefly explain the reason for the difference (or lack thereof) in the worst-case runtimes from parts (c) and (d).

Solution: With Weighted Quick-Union, the root of a tree with fewer nodes becomes a child of the root of a tree with more nodes. Consequently, each time a node's tree's root changes, the node's height increases by 1 and the node's tree at least doubles in size. This can happen at most $\log n$ times since there are only n nodes.