Poll results

As of 1am this morning...

- 0 (0% of users)
- 3 (25% of users)
- 0 (0% of users)
- 3 (25% of users)
- 5 (42% of users)
- 6 (50% of users)
- 1 (8% of users)
- 4 (33% of users)
- 5 (42% of users)
- 0 (0% of users)
- 7 (58% of users)
- 5 (42% of users)
- 6 (50% of users)
- 6 (50% of users)
I have a bunch of practice problems.

Y’all vote on topics and we’ll do them.

I can also answer particular questions about the material.

Topics I have problems for:
- Grab-bag (multiple choice, etc)
- Hashing
- Red-Black Trees
- Ford-Fulkerson
- Dynamic Programming
- Greedy algorithms
- Divide and conquer
- Randomized algs
Multiple choice warmup!

For each of the following quantities, **identify all of the options** that correctly describe the quantity.

(a) The function $f(n)$, where $f(n) = n \log(n)$.  
   (A), (C)

(b) $T(n)$ given by $T(n) = T(n/4) + \Theta(n^2)$ with $T(n) = 1$ for all $n \leq 8$.  
   (A), (B), (C)

(c) $T(n)$ which is the running time of the following algorithm:

   ```
   mysteryAlg(n):
   if n < 3:
       return 1
   return mysteryAlg(n/2) + mysteryAlg((n/2) + 1)
   ```

   where above all division is integer division (so $a/b$ means $\lfloor a/b \rfloor$).

   (A), (C), (D)

(A) $O(n^2)$  
(B) $\Theta(n^2)$  
(C) $\Omega(n)$  
(D) $O(n)$  
(E) $O(\log^2(n))$.  

CS161 Review Session Practice Problems 12/6/2017 4 / 15
Let $G = (V, E)$ be an undirected weighted graph, and let $T$ be a minimum spanning tree in $G$. Decide whether the following statements must be true or may be false, and prove it!

(a) For any pair of distinct vertices $s, t \in V$, there is a unique path from $s$ to $t$ in $T$.

(b) For any pair of distinct vertices $s, t \in V$, the cost of a path between $s$ and $t$ in $T$ is minimal among all paths from $s$ to $t$ in $G$. 

(a) True
(b) False
Hashing warm-up

Let \( \mathcal{U} \) be a universe of size \( m \), where \( m \) is a prime, and consider the following two hash families which hash \( \mathcal{U} \) into \( n \) buckets, where \( n \) is much smaller than \( m \).

- First, consider \( \mathcal{H}_1 \), which is the set of all functions from \( \mathcal{U} \) to \( \{1, \ldots, n\} \):
  \[
  \mathcal{H}_1 = \{ h \mid h : \mathcal{U} \to \{1, \ldots, n\} \}
  \]

- Second, let \( p = m \) (so \( p \) is prime since we assumed \( m \) to be prime), and choose \( \mathcal{H}_2 \) to be
  \[
  \mathcal{H}_2 = \{ h_{a,b} \mid a \in \{1, \ldots, p - 1\}, b \in \{0, \ldots, p - 1\} \},
  \]
  where \( h_{a,b}(x) = (ax + b \mod p) \mod n \).

You want to implement a hash table using one of these two families. Why would you choose \( \mathcal{H}_2 \) over \( \mathcal{H}_1 \)? Choose the best answer.

(A) \( \mathcal{H}_1 \) isn't a universal hash family.

(B) Storing an element of \( \mathcal{H}_1 \) takes a lot of space.

(C) Storing all of \( \mathcal{H}_1 \) takes a lot of space.
Shortest Paths

- When might you prefer breadth-first search to Dijkstra’s algorithm?
  
  If the graph is unweighted

- When might you prefer Floyd-Warshall to Bellman-Ford?
  
  If you want shortest paths between all pairs of vertices.

- When might you prefer Bellman-Ford to Dijkstra’s algorithm?
  
  If there are negative edge weights
Suppose that $b_1, \ldots, b_n$ are $n$ distinct integers in a uniformly random order. Consider the following algorithm:

```python
findMax(b_1, \ldots, b_n):
    currentMax = -\infty
    for i = 1, \ldots, n:
        if b_i > currentMax:
            currentMax = b_i
    return currentMax
```

What is the expected number of times that `currentMax` is updated? (Asymptotic notation is fine).

$$
\mathbb{E}\{\text{# times currentMax updated}\} = \sum_{i=1}^{n} \mathbb{P}\{b_i > b_1, \ldots, b_{i-1}\}
= \sum_{i=1}^{n} \frac{1}{i}
= \Theta(\log(n))
$$

Since $b_1, \ldots, b_i$ are uniformly random, the probability that any one is largest is $\frac{1}{i}$. 

Randomized algorithms
Consider the following flow on a graph. The notation $x/y$ means that an edge has flow $x$ out of capacity $y$.

- Draw the residual graph for this flow.
- Find an augmenting path in the residual graph and use it to increase the flow. The path highlighted above results in the flow marked above.
- Find a minimum cut and prove (not by exhaustion) that it is a minimum cut. The cut $\{s,a\}, \{b,t\}$ has value 2, which is minimal since $2 \leq \text{max flow} = \text{min cut} \leq 2$.
Dynamic Programming!

- Suppose that roads in a city are laid out in an \( n \times n \) grid, but some of the roads are obstructed.

- For example, for \( n = 3 \), the city may look like this:

Define \( M[i,j] = \text{#paths from } (0,0) \text{ to } (i,j) \).

\[
M[i,j] = \begin{cases} 
1 & \text{if there is a road from } (0,0) \text{ to } (i,j) \\
M[i-1,j] + M[i,j-1] & \text{otherwise}
\end{cases}
\]

Algorithm:
- Initialize \( M[0,0] = 1 \).
- For \( i = 0, \ldots, n-1 \):
  - For \( j = 0, \ldots, n-1 \):
    - if \( i > 0 \) and there is a road from \( (i-1,j) \) to \( (i,j) \): \( M[i,j] += M[i-1,j] \)
    - if \( j > 0 \) and there is a road from \( (i,j-1) \) to \( (i,j) \): \( M[i,j] += M[i,j-1] \)
- Return \( M[n-1,n-1] \).

where we have only drawn the roads that are not blocked. You want to count the number of ways to get from \((0,0)\) to \((n-1, n-1)\), using paths that only go up and to the right. In the example above, the number of paths is 3.

- Design a DP algorithm to solve this problem.
Divide and Conquer!

- Given an array $A$ of length $n$, we say that an array $B$ is a **circular shift** of $A$ if there is an integer $k$ between 1 and $n$ (inclusive) so that


where $+$ denotes concatenation.

- For example, if $A = [2, 5, 6, 8, 9]$, then $B = [6, 8, 9, 2, 5]$ is a circular shift of $A$ (with $k = 2$). The sorted array $A$ itself is also a circular shift of $A$ (with $k = 1$).

- Design a $O(\log(n))$-time algorithm that takes as input an array $B$ which is a circular shift of a sorted array which contains distinct positive integers, and returns the value of the largest element in $B$. For example, give $B$ as above, your algorithm should return 9.

Solution on next page.
def findMax(B):
    n ← ln(n)
    if B[0] ≤ B[n-1]: \ case 1
        return B[n-1]
    mid = \left\lceil n/2 \right\rceil + 1
    if B[mid] > B[0]: \ case 2
        return findMax(B[mid:n])
    if B[mid] < B[0]: \ case 3
        return findMax(B[mid+1:])

Idea:
- In case 1, the situation looks like
  \[ O \quad n-1 \]
  So we return B[n-1]
- In case 2, it looks like
  \[ O \mid n-1 \]
  So the max is on the right side and we recurse on B[mid:]
- In case 3, it looks like
  \[ \text{mid} \quad n-1 \]
  So the max is on the left side and we recurse on B[mid+1]
There are $n$ final exams on Dec. 13 at Stanford; exam $i$ is scheduled to begin at time $a_i$ and end at time $b_i$. Two exams which overlap cannot be administered in the same classroom; two exams $i$ and $j$ are defined to be overlapping if $[a_i, b_i] \cap [a_j, b_j] \neq \emptyset$ (including if $b_i = a_j$, so one starts exactly at the time that the other ends). Design an algorithm which solves the following problem.

- **Input**: Arrays $A$ and $B$ of length $n$ so that $A[i] = a_i$ and $B[i] = b_i$.
- **Output**: The smallest number of classrooms necessary to schedule all of the exams, and an optimal assignment of exams to classrooms.
- **Running time**: $O(n \log(n) + nk)$, where $k$ is the minimum number of classrooms needed.
- **For example**: Suppose there are three exams, with start and finish times as given below:

<table>
<thead>
<tr>
<th>$i$</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_i$</td>
<td>12pm</td>
<td>4pm</td>
<td>2pm</td>
</tr>
<tr>
<td>$b_i$</td>
<td>3pm</td>
<td>6pm</td>
<td>5pm</td>
</tr>
</tbody>
</table>

Then the exams can be scheduled in two rooms; Exam 1 and Exam 2 can be scheduled in Room 1 and Exam 3 can be scheduled in Room 2.
def scheduleRooms(A, B):
    # IIDEA: Sort exams by start time.
    n ← len(A)
    C = \[ (A[i], : ) \mid i = 0, \ldots, n - 1 \]
    # Sort C increasing order by start time.
    sort C
    # increase time by start time.
    rooms = []  # list of rooms
    endTimes = []
    for i = 0, \ldots, n - 1:
        for r = 0, \ldots, len(rooms) - 1:
            if C[i][1] > endTimes[r]:
                rooms[r].append(C[i][1])
                endTimes[r] = B[C[i][1]]
        else:
            # did not break
            rooms.append(C[i][1])
            endTimes.append(B[C[i][1]])
    return rooms.

Correctness by induction

Inductive Hyp: After adding the ith exam, there is an optimal schedule that extends the current schedule.

Base Case: After adding 0 exams, there is an optimal schedule extending times.

Inductive Step: Suppose the inductive hyp holds for i - 1, and let S be the optimal schedule that extends it.

If S puts exam i where we would put it (say, room r), then we are done, so suppose that S puts exam i in room r:

Let j > i be the next exam scheduled in room r.
Then aj ≥ ai, since ai had the smallest start time of all exams not yet picked.
So consider the schedule S' where we swap the rest of room r with the rest of room r:

This is still a valid schedule, and uses the same 14 rooms as S, so it's also optimal. And it puts exam j in room r, so we're done.

Conclusion: At the end of the algo, there's still an optimal schedule extending the current one to the current one is optimal.
Universal Hash Families

- Definition: A hash family $H$ (mapping $U$ into $n$ buckets) is **2-universal** if for all $x \neq y \in U$ and for all $a, b \in \{1, \ldots, n\}$,

$$P((h(x), h(y)) = (a, b)) = \frac{1}{n^2}.$$

(a) Show that if $H$ is 2-universal, then it is universal.

(b) Show that the converse is not true. That is, there is a universal family that’s not 2-universal.

(a) Suppose that $H$ is 2-universal. Then for all $x \neq y \in U$,

$$P\left\{ h(x) = h(y) \right\} = \sum_{t \in \{1, \ldots, n\}} P\{ (h(x), h(y)) = (t, t) \} = \frac{1}{n}.$$

So by definition $H$ is universal.

(b) Consider: $U = \{x, y\}$, $H = \{h_x, h_y\}$, where:

- $h_x$:

  \[
  \begin{array}{cc}
  x & y \\
  0 & 0
  \end{array}
  \]

  Then $P\{ h(x) = h(y) \} = \frac{1}{2}$.

- $h_y$:

  \[
  \begin{array}{cc}
  x & y \\
  1 & 0
  \end{array}
  \]

  But $P\{ (h(x), h(y)) = (0, 0) \} = \frac{1}{2}$, not $\frac{1}{n^2}$. 
More universal hash families

Say that $\mathcal{H}$ is a universal hash family, containing functions $h : \mathcal{U} \rightarrow \{1, \ldots, n\}$. Consider the following game.

- You choose $h \in \mathcal{H}$ uniformly at random and keep it secret.
- A bad guy chooses $x \in \mathcal{U}$, and asks you for $h(x)$. (You give it to them).
- The bad guy chooses $y \in \mathcal{U} \setminus \{x\}$, and tries to get $h(y) = h(x)$.
- If $h(x) = h(y)$, the bad guy wins. Otherwise, you win.

One of the following two is true.

1. There is a universal hash family $\mathcal{H}$ so that the bad guy wins with probability 1.
2. For any universal hash family $\mathcal{H}$, the probability that the bad guy wins is at most $1/n$.

Which is true and why?
Which of the following can be colored as a red-black tree? Either give a coloring or explain why not.