Question 1 (Minimum Spanning Tree, 30 points). Give a minimum spanning tree for the graph below: The answer is given above. This can be computed using Kruskal’s algorithm. The highlighted edges are the ones that do not form cycles with lower numbered edges.
**Question 2** (Heavier than Average, 35 points). *Professor Hayle has a collection of n rocks. He would like to find a rock from his collection whose weight is at least as heavy as the average. He has a scale that given any subset S of the rocks can weigh them to find the total weight of all rocks in S. Show how Hayle can use the scale to find a rock whose weight at least as heavy as the average. For full credit, your solution should use at most $O(\log(n))$ weighings.*

There is a simple divide and conquer algorithm for this problem.

```plaintext
HeavierThanAverage(S)
If |S| = 1
   Return the element of S
Else
   Split S into nearly equally sized sets S1 and S2
   Weight S1 and S2
   If Weight(S1)/|S1| > Weight(S2)/|S2|
      Return HeavierThanAverage(S1)
   Else
      Return HeavierThanAverage(S2)
```

For the runtime, note that this algorithm makes a constant number of weighings before reducing to a problem of half the size. Thus the runtime satisfies a recurrence $T(n) = T(n/2 + O(1)) + O(1)$ and by the master theorem, $T(n) = O(\log(n))$.

To show correctness we proceed by induction on |S|. If |S| = 1, then the single element of S has weight equal to the average and so returning it is acceptable. Assuming that our algorithm works for all smaller sized sets, we note that if Weight(S1)/|S1| > Weight(S2)/|S2|, then it is not hard to see that Weight(S1)/|S1| > (Weight(S1) + Weight(S2))/(|S1| + |S2|) = Weight(S)/|S|. Thus, the average weight of a rock in S1 is more than the average weight of a rock in S. Therefore, since by our inductive hypothesis the algorithm in this case returns a rock whose weight is at least the average weight of rocks in S1, we return a correct output. Otherwise, we have Weight(S2)/|S2| ≥ Weight(S1)/|S1|, which similarly implies that the average weight of a rock in S2 is at least the average weight of a rock in S, and so similarly our answer will be correct.
**Question 3** (Class Scheduling with Breaks, 35 points). *Ashton is trying to schedule as many classes as possible in his daily schedule. There are $n$ possible classes that he can take each with a start and end time. He would like to schedule as many as possible subject to the condition that no two overlap. However, Ashton also likes to have a long nap break in the middle of his day. So, subject to scheduling as many classes as possible without overlap, he would also like for there to be a gap as large as possible between two of his classes.*

In other words, Ashton’s schedule must contain no two overlapping classes, and must contain as many non-overlapping classes as possible. But subject to this, he would like to have a gap of length $T$ between some two of his consecutive classes for $T$ as large as possible.

Give an algorithm to compute the largest possible such gap. For full credit your algorithm should run in time $O(n \log(n))$ or better.

The algorithm begins with the standard greedy algorithm for computing a maximal set of classes with the earliest possible end times.

**EarlyEnds(S)**

Sort the intervals in $S$ by end time
$L \leftarrow \{\}$
For $I$ in $S$ in increasing order of end time
  If $\text{start}(I) > \text{end}(\text{last element of } L)$
    Append $I$ to $L$
Return $L$

We also run the algorithm from the other side to compute the maximal set of classes with the latest possible start times.

**LateStarts(S)**

Sort the intervals in $S$ by start time
$L \leftarrow \{\}$
For $I$ in $S$ in decreasing order of start time
  If $\text{end}(I) < \text{start}(\text{last element of } L)$
    Append $I$ to $L$
Return $L$

Our final algorithm is as follows:

**LongestGap(S)**

$\text{EarlyEnds} \leftarrow \text{EarlyEnds}(S)$
$\text{LateStarts} \leftarrow \text{Reverse}(\text{LateStarts}(S))$
$\text{Best} \leftarrow 0$
For $k = 1$ to $\text{Length}(\text{EarlyEnds}) - 1$
  $\text{Best} \leftarrow \max(\text{Best}, \text{Start}(\text{LateStarts}(k+1)) - \text{End}(\text{EarlyEnds}(k)))$
Return $\text{Best}$

The runtime is clearly $O(n \log(n))$ as it is dominated by the two sorting operations, and everything else is linear in $n$.

To prove correctness we first note that EarlyEnds computes a maximal sized schedule with the end time of the $k^{th}$ class as early as possible for each value of $k$. Similarly, LateStarts computes a maximal schedule whose $k^{th}$ class starts as late as possible for each $k$. We know that the gap $T$ in Ashton’s final schedule must be the difference between the end time of his $k^{th}$ class and the start of his $(k+1)^{th}$ for some value of $k$. This is therefore at most the maximum value over $k$ of the difference between the end time of the $k^{th}$ interval of EarlyEnds and the start of the $(k + 1)^{th}$ of LateStarts, which is what we compute. However, this value is achievable, as it is not hard to see that if Ashton’s schedule consists of the first $k$ intervals of EarlyEnds and the last $n - k$ of LateStarts that this will be a valid schedule.