CPSC 320 2019S2: Assignment 4

Please submit this assignment via GradeScope at `https://gradescope.com`. Be sure to identify everyone in your group if you're making a group submission (which we encourage!).

Submit by the deadline **Tuesday July 30 at 11PM**. For credit, your group must make a **single** submission via one group member's account, marking all other group members in that submission **using GradeScope's interface**. Your group's submission must:

- Be on time.
- Consist of a single, clearly legible file uploadable to GradeScope with clearly indicated solutions to the problems. (PDFs produced via LaTeX, Word, Google Docs, or other editing software work well. Scanned documents will likely work well. **High-quality** photographs are OK if we agree they're legible.)
- Include prominent numbering that corresponds to the numbering used in this assignment handout. Put these **in order** starting each problem on a new page, ideally. If not, **very clearly** and prominently indicate which problem is answered where!
- Include at the start of the document the **ugrad.cs.ubc.ca e-mail addresses** of each member of your team. Please **do not** include names on the assignment. If you don't mind private information being stored outside of Canada and want an extra double-check on your identity, include your student number rather than your name.
- Include at the start of the document the statement: "All group members have read and followed the guidelines for academic conduct in CPSC 320. As part of those rules, when collaborating with anyone outside my group, (1) I and my collaborators took no record but names (and GradeScope information) away, and (2) after a suitable break, my group created the assignment I am submitting without help from anyone other than the course staff." (Go read those guidelines!)
- Include at the start of the document your outside-group collaborators' ugrad.cs.ubc.ca IDs, but **not** their names. (Be sure to get those IDs when you collaborate!)

1 A Recurring Dream

Give an asymptotic solution (which should be a Θ-bound) to each of the recurrences below. You may use whatever solution method you wish (drawing out the tree, unrolling the recurrence, proof by induction, Master Theorem, etc.), but make sure you fully justify your answer.

1. $T(n) = 2T(n-1) + c$ for $n > 1$, $T(0) = 1$.
2. $T(n) = nT(n-1) + c$ for $n > 1$, $T(1) = 1$.
3. $T(n) = 4T\left(\frac{n}{3}\right) + cn \log n$ for $n > 1$, $T(1) = 1$.
4. **[BONUS, WORTH 1 COURSE BONUS POINT]** $T(m, n) = mT(m, \frac{n}{2}) + cn$ for $n > 1$ and $m > 0$; $T(m, 1) = 1$.

This work is licensed under a Creative Commons Attribution 4.0 International License.
For license purposes, the author is the University of British Columbia.

2 The Dividing Tree

Your friend proposes a divide-and-conquer approach to compute a minimum spanning tree of a weighted, undirected, connected graph $G = (V, E)$. The helper function

```
{G_1, G_2} = subgraphs(G)
```

takes a graph $G = (V, E)$ and partitions the vertices into two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ such that G_1 and G_2 are both connected and $|V_1|$ and $|V_2|$ differ by at most 1. If (and only if) such a partitioning is **not possible**, one or both of the graphs G_1 and G_2 will be disconnected.

```
DC_MST(G = (V, E)):
   \\ Base cases:
   if |E| = 0:                        \\ no edges
      return NONE
   if |E| = 1:                        \\ 1 edge
      return E

   {G_1, G_2} = subgraphs(G)
   MST_1 = DC_MST(G_1)
   MST_2 = DC_MST(G_2)
   let e = the minimum-weight edge connecting G_1 and G_2
   return [MST_1, e, MST_2]
```

We'll start by analyzing the runtime of this algorithm. Assume that the `subgraphs` function runs in $\Theta(n+m)$ time, where $n = |V|$ and $m = |E|$. For questions 1 to 4, assume we have a connected graph where `subgraphs` can always generate connected G_1 and G_2 subgraphs.

1. The best case for this algorithm is a particular (simple and common) type of connected graph. What type of connected graph yields the best-case runtime? VERY briefly justify your answer.

2. What type of connected graph yields the worst-case runtime? VERY briefly justify your answer.

3. Give and briefly justify a good asymptotic bound on the **best-case** runtime of this algorithm in terms of n.

4. Give and briefly justify a good asymptotic bound on the **worst-case** runtime of this algorithm in terms of n.

5. Does this algorithm always generate a minimum spanning tree? (You should assume the input graph is connected, but need no longer assume that `subgraphs` can always generate connected G_1 and G_2 subgraphs.)

   - If it does, provide a proof.
   - If it does not, provide an example where the algorithm fails to produce an MST and explain it by indicating the MST and clearly explaining the answer produced by the algorithm. In cases where there are multiple possible return values for the `subgraphs` function, you will need to determine and state what the values are.

`Subgraphs` v2.0

Now suppose that we have modified our `subgraphs` function so that when it isn't possible to partition G into two connected subgraphs of equal size, `subgraphs` instead returns two **connected** graphs G_1 and G_2, where the absolute difference between $|V_1|$ and $|V_2|$ is minimized.

This work is licensed under a Creative Commons Attribution 4.0 International License.
For license purposes, the author is the University of British Columbia.

6. Does **this** version of the algorithm always generate a minimum spanning tree (again, assuming a connected input graph)?

   - If it does, provide a proof.
   - If it does not, provide an example where the algorithm fails to produce an MST and explain it by indicating the MST and clearly explaining the answer produced by the algorithm. In cases where there are multiple possible return values for the `subgraphs` function, you will need to determine and state what the values are.

3 Conquering the Auction Market

You own an online sales company called DCAuctions.com that sells goods both on auction and on a fixed-price basis. You want to use historical auction data to investigate your fixed price choices.

Over n minutes, you have a good's price in each minute of the auction. You want to find the largest *price-over-time stretch* in your data. That is, given an array A of n price points, you want to find the largest possible value of

$$f(i,d) = d \cdot \min(A[i], A[i+1], \ldots, A[i+d-1]),$$

where i is the index of the left end of a stretch of minutes, d is the duration (number of minutes) of the stretch, and the function f computes the duration times the minimum price over that period. (Prices are positive, $d \geq 0$, and for all values of i, $f(i,0) = 0$ and $f(i,1) = A[i]$.)

For example, the best stretch is underlined in the following price array: $[8, 2, \underline{9, 5, 6, 5}, 3, 1]$. Using 1-based indexing, the value for this optimal stretch starting at index 3 and running for 4 minutes is $f(3,4) = 4 \cdot \min(9,5,6,5) = 4 \cdot 5 = 20$.

1. Give a **brute force** algorithm to solve this problem. Your algorithm must run in polynomial time.
2. Give and briefly justify a good asymptotic bound on the runtime of your algorithm.

D + C = Profit

In this part, you will give a divide-and-conquer algorithm that is more efficient than the brute force approach.

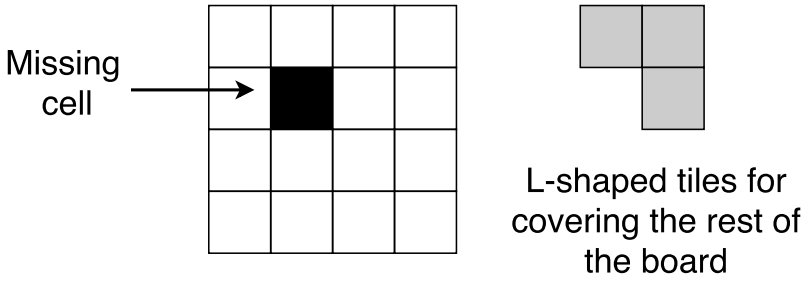
3. Suppose the minimum element in A is $A[k] = 4$, and suppose that A has length n. What's the best price stretch for all intervals that include $A[k]$? Briefly justify your answer.
4. Using the insight from the previous question, give an efficient algorithm to find the best price stretch.
5. Give and briefly justify a good asymptotic bound on the **worst-case** runtime of your algorithm.
6. Give and briefly justify a good asymptotic bound on the **average-case** runtime of your algorithm.
7. **[BONUS, WORTH 1 COURSE BONUS POINT]** A *segment tree* is a data structure that lets you obtain the maximum value of an interval $A[i]$ to $A[j]$, for $1 \leq i \leq j \leq n$, in $O(\log n)$ time, where n is the length of A. The segment tree takes $O(n)$ time to build. Describe how you could modify your algorithm to obtain a better worst-case runtime than you computed in question 3.

4 Tiles and Tribulations

NOTE: FRIDAY'S TUTORIAL QUIZ WILL BE BASED ON THIS QUESTION. The quiz will ask **different questions** than the ones below, but will provide a **substantial hint** to help with the assignment questions. Before Friday, we recommend you prioritize the other questions on this assignment (because this question will probably be easier if you wait until after we've released the hint on the quiz).

This work is licensed under a Creative Commons Attribution 4.0 International License.
For license purposes, the author is the University of British Columbia.

You're given a grid of size $n \times n$, where $n = 2^k$ for some $k \geq 1$, with one cell missing. Your job is to cover the board with *L-tiles*. An L-tile is three squares that form an L shape (i.e., a 2×2 square with one square missing). Below is a 4×4 board, with a missing cell at position $(2, 3)$ (using 1-based indexing from bottom left).



The L-tiles:

- Must cover every white square of the board.
- Must NOT cover the single black square on the board.
- Are not allowed to overlap.

1. Prove that, for any $2^k \times 2^k$ board with an arbitrary missing cell, we can come up with a way to cover the board with L-tiles.

2. Design a divide-and-conquer algorithm to place L-tiles to cover a $2^k \times 2^k$ board with a single missing cell.

3. Give and briefly justify a good asymptotic bound on the runtime of your algorithm in terms of n (the dimension of the board).

This work is licensed under a Creative Commons Attribution 4.0 International License.
For license purposes, the author is the University of British Columbia.