COMP 3804 – Design and Analysis of Algorithms
Assignment 1

Due: January 27, 2017 at 16:30

- Your assignment should be submitted online on cuLearn. Your answers should be precise, concise and clear.
- Unless indicated otherwise, every part of every question is worth 2 marks. The grading scheme is 2 points for a correct answer, 0 for a completely incorrect answer, and 1 point for something in-between.

1. Solve the following recurrences. Give the answer in asymptotic (big-O) notation. You may assume that n is "nice", but be explicit about the exact assumption you make. For example, for question (a), you may assume that n is a power of 2. Make sure that your assumption does not limit how big n can be.

   (a)
   $$T(n) = \begin{cases} 1 & \text{if } n = 1 \\ 3 \cdot T(n/2) + 5 & \text{otherwise} \end{cases}$$

   (b)
   $$T(n) = \begin{cases} 1 & \text{if } n = 1 \\ 5 \cdot T(n/7) + n & \text{otherwise} \end{cases}$$

   (c)
   $$T(n) = \begin{cases} 1 & \text{if } n = 1 \\ 8 \cdot T(n/2) + n^3 & \text{otherwise} \end{cases}$$

   (d)
   $$T(n) = \begin{cases} 1 & \text{if } n = 1 \\ 2 \cdot T(n-2) + 1 & \text{otherwise} \end{cases}$$

2. Consider the following recursive algorithm BEER(n), which takes as input an integer $n \geq 1$:

   Algorithm BEER(n)
   if $n = 1$ **then**
   eat some peanuts
   else
   drink one pint of beer
   choose an arbitrary integer m with $1 \leq m \leq n - 1$
   BEER(m)
   BEER($n - m$)

   (a) Draw a possible recursion tree for BEER(6).

   (b) Explain why, for any integer $n \geq 1$, algorithm BEER(n) terminates.

   (c) Let $B(n)$ be the number of pints of beer you drink when running algorithm BEER(n). Give the recurrence relation for $B(n)$.

   (d) Use your recursion tree to guess what the exact value of $B(n)$ is. Then use induction to prove that your guess is correct.

3. Let S be a set of n points in the plane. Each point p of S is given by its x- and y-coordinates p_x and p_y, respectively. A point p of S is called *maximal* in S if its top-right quadrant is empty:

$$\neg\exists\, q \in S : q_x > p_x \text{ and } q_y > p_y.$$

See Figure 1 for an example. Observe that, in general, there is more than one maximal element in S. To avoid special cases, you may assume that no two points in S have the same x-coordinate, and no two points in S have the same y-coordinate.

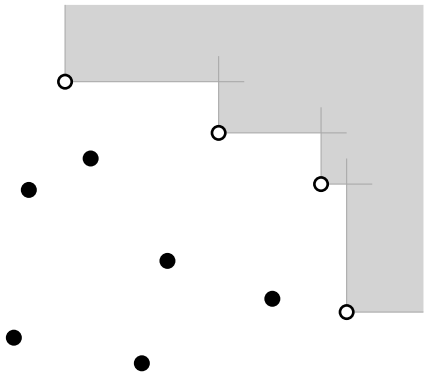

Figure 1: A set of points with the maximal points and their top-right quadrants highlighted.

(a) Give a divide-and-conquer algorithm (in plain English or pseudocode) that computes all maximal elements in S in $O(n \log n)$ time. (*Hint:* At the start of the algorithm, sort the points of S from left to right. Use this ordering to divide the input set into two subsets. You don't have to give pseudocode for the sorting step.)

(b) Explain why your algorithm is correct.

(c) Explain why your algorithm always terminates.

(d) Analyze the running time of your algorithm when called on a set of n points. Show that it is indeed $O(n \log n)$.

4. Let S be a set of n points in the plane. A *split tree* is a special binary tree that stores all the points in S. The root of the split tree stores the point p with the median x-coordinate among all points in S. This splits S into two sets, S_{left} and S_{right}, which consist of the points to the left and right of p, respectively. The left child is a split tree on S_{left} and the right child is a split tree on S_{right}. But there is a twist: their root stores the point with the median y-coordinate and splits the rest of the points based on their y-coordinate, not x-coordinate. The direction of the split alternates with each level. See Figure 2 for an example.

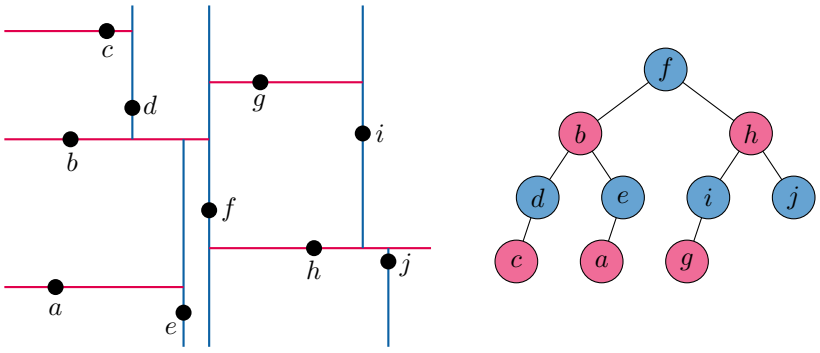

Figure 2: A set of points in the plane with the split tree built on these points. Vertical splits (based on x-coordinate) are indicated in blue, while horizontal splits (on y-coordinate) are red.

The following algorithm builds a split tree. Given a set of points S and the direction of the next split, it returns the root of the split tree:

```
1:  Algorithm BuildSplitTree(S, split direction)
2:      if S = ∅ then
3:          return Nil
4:      else
5:          p ← the median point in the split direction
6:          S1 ← all points in S before p in the split direction
7:          T1 ← BuildSplitTree(S1, other split direction)
8:          S2 ← all points in S after p in the split direction
9:          T2 ← BuildSplitTree(S2, other split direction)
10:         return a new node, storing p, with left child T1 and right child T2
11:
```

(a) Line 5 can be implemented by sorting S in the split direction and taking the middle point. What is the total construction time in that case?

(b) How can we perform this step in $O(n)$ time instead? (*Hint:* There is a lot of repeated sorting of the same elements.)

We can assign a *cell* to every node in the split tree. The cell of a node X represents the region where the points in the subtree rooted at X can be. For example, in Figure 2, the cell of f is the entire plane. The cell of h is everything to the right of the vertical line through f. And the cell of g is everything between the vertical lines through f and i and above the horizontal line through h (see Figure 3).

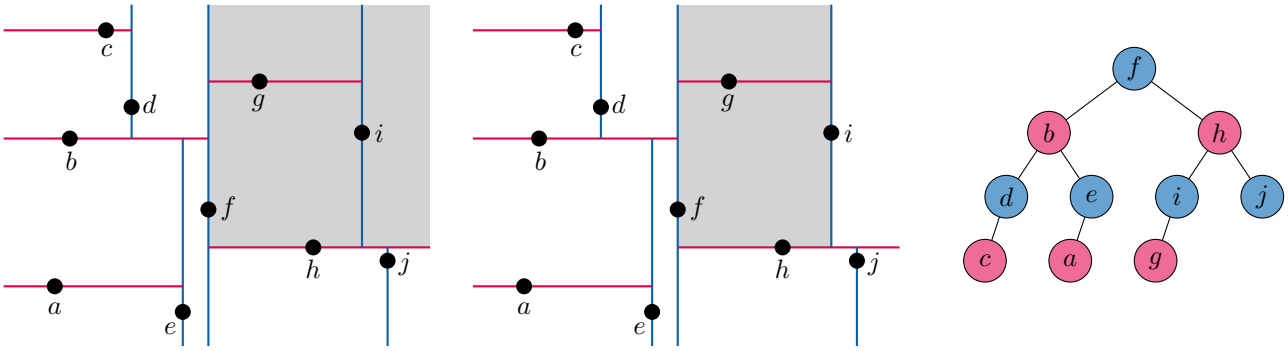

Figure 3: The cell corresponding to i (left) and the cell corresponding to g (middle).

(c) Give a divide-and-conquer algorithm that, given a rectangle R and a split tree on a set of points S, finds all points in S that lie within R. Try to avoid checking a branch of the tree whenever possible. This implies that your algorithm should only check nodes whose cell lies at least partly inside R.

If there are k points inside R, reporting all of them will take at least $\Omega(k)$ time. The key to analyzing the efficiency of the query algorithm is to bound the number of nodes inspected whose point lies outside R. Note that this is at most the number of cells intersected by the *boundary* of R. The next part is the key to showing that the number of such cells is $O(\sqrt{n})$, giving a total query time of $O(\sqrt{n} + k)$.

(d) Show that a vertical line intersects at most $O(\sqrt{n})$ cells. (*Hint:* Give a recurrence, considering two levels of recursion at once.)

(e) (10%) Complete the split tree implementation in the accompanying zip file. Submit your source code to the submission server (not cuLearn).

(f) **Bonus:** Modify your implementation of the split tree to reduce the time required to count the points inside R to just $O(\sqrt{n})$, removing the dependency on the number of points inside R.