Pre-tutorial questions

Do you know the basic concepts of this week's lecture content? These questions are only to test yourself. They will not be explicitly discussed in the tutorial, and no solutions will be given to them.

1. Divide and Conquer
   (a) What is the general idea of a Divide-and-Conquer algorithm? Can you break up the general structure into three steps?
   (b) Why are recurrences usually needed to analyse the running time of a Divide-and-Conquer algorithm?
2. Algorithms
   (a) Describe the general idea of the algorithm for maximum contiguous subarray.
   (b) What are the three main steps in the algorithm for counting inversions?
   (c) In the merge step of the Closest-Pair algorithm,
      i. Why do we only need to consider points that are within the δ-strip to the left of the dividing line and the δ-strip to the right?
      ii. Suppose the points in these strips are $s_1, \ldots, s_k$, in ascending order of their y-coordinate. Why is it that we only need to check pairs (s_i, s_j) for $|i - j| \leq 7$?
3. Recurrences
   (a) State the master method (write down all three cases).
   (b) Try to explain the master method using a recursion tree.
4. Review Tutorial 11 and 12 from COMP2123 on solving recurrences using master method as well as designing divide and conquer algorithms.

Tutorial

Problem 1

Suppose we are given numbers a, n, where $n > 0$ is an integer. We wish to calculate the number a^n. What is the quickest way to do this? How many multiplication operations do we have to perform? Of course, we may compute 19^8 by calculating $19 \times 19 = 361$, then calculating $19^3 = 361 \times 19 = 6859$, then $19^4 = 6859 \times 19 = 130321$, and so on, until we get 19^8. This takes seven multiplications in total. Is this the quickest possible? Note that $8 = 2 \times 4$, so we can also write $19^8 = 19^4 \times 19^4$. If we compute 19^4 first, and then square it, we need only one more multiplication. The straightforward method would require four more multiplications: $19^8 = 19^4 \times 19 \times 19 \times 19 \times 19$. Similarly, $19^4 = 19^2 \times 19^2$. So if we calculate $19^2 = 361$ with one multiplication, $19^4 = 361^2 = 130321$ with one more, we get $19^8 = 130321^2 = 16983563041$ with the third multiplication. This cleverer method requires only three multiplications. The method above seems to work when the exponent n is even. What do we do when it is odd? Say, we would like to calculate 19^7. We may

write $7 = 6+1$, so $19^7 = 19^6 \times 19$, then $19^6 = 19^3 \times 19^3$, and finally $19^3 = 19^2 \times 19$. So $19^3 = 361 \times 19 = 6859$, $19^6 = 6859^2 = 47045881$, and $19^7 = 47045881 \times 19 = 893871739$. The straightforward method of calculation requires 6 multiplications, and we needed only 4 here. We can combine the ideas from the two examples above to get a procedure to calculate the power a^n for any pair a, n.

Algorithm 1 Power

```
1:  function POWER(a, n)
2:      if n = 1 then
3:          return a
4:      end if
5:      if n is even then
6:          b = POWER(a, n/2)
7:          return b^2
8:      else
9:          b = POWER(a, (n − 1)/2)
10:         return a × b^2
11:     end if
12: end function
```

Prove that the algorithm correctly computes a^n. Can you bound the number of multiplications for each n?

Problem 2

(Exercise 6, Chapter 5 from textbook Algorithm Design) Let T be a complete binary tree with n vertices. Each vertex v has a distinct weight $w(v)$. A vertex is a *local minima* if its weight is less than the weight of its neighbors. Give a divide-and-conquer algorithm that finds a local minima in time $O(\log n)$.

Problem 3

(Due to Jeff Erickson.) For this problem, a *subtree* of a binary tree means any connected subgraph. A binary tree is *complete* if every internal node has two children, and every leaf has exactly the same depth. Describe and analyze a divide and conquer algorithm to compute the *largest complete subtree* of a given binary tree. Your algorithm should return both the root and the depth of this subtree. See Figure 1 for an example.

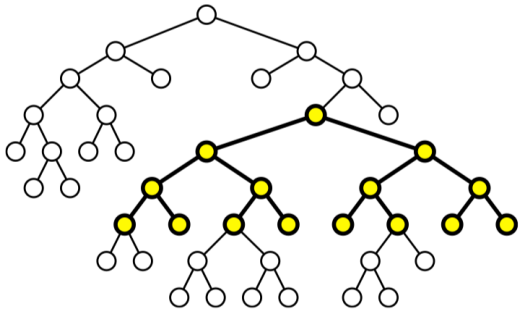

Figure 1: The largest complete subtree of this binary tree has depth 3.

Problem 4

(Due to Jeff Erickson.) Let $n \geq 2$ be a power of two. Suppose you are given a $n \times n$ checkerboard with one

(arbitrarily chosen) square removed. Describe and analyze an algorithm to compute a tiling of the board without gaps or overlaps by L-shaped tiles, each composed of 3 squares. Your input is the integer n and two integers representing the row and column of the missing square. The output is a list of the positions and orientations of $(n^2 - 1)/3$ tiles. Your algorithm should run in $O(n^2)$ time (note that this is linear in the size of the checkerboard). *Hint: prove that such a tiling always exists by induction on n.*

Problem 5

[COMP3927 only] Recall the streaming algorithm for majority. Let $k > 0$ be the value of the counter at the end of the stream. In the lecture, we saw an example where $k = 1$ and yet there is no absolute majority. For what values of k can we be sure that there is an absolute majority?

Problem 6

[COMP3927 only] Can you prove a lower bound on the approximation ratio of the "MST-approach" (doubling the MST) for the Travelling Salesman Problem?