York University
LE/EECS 3101 A Fall 2022
Assignment 3

Due Date: November 9th, at 23:59

A dreamer is one who can only find his way by moonlight, and his punishment is that he sees the dawn before the rest of the world ...

Oscar Wilde

All problems are written problems; submit your solutions electronically **only via Crowdmark**. You are welcome to discuss the general idea of the problems with other students. However, you must write your answers individually and mention your peers (with whom you discussed the problems) in your solution.

Problem 1 Sorting [3+3=6 marks]

An steady sorting algorithm is one in which the relative order of all identical elements (or keys) is the same in the output as it was in the input. For example, a steady sorting of array $[1, 3, 6, 1', 4, 6', 4']$ gives $[1, 1', 3, 4, 4', 6, 6']$.

a) Indicate whether heap sort is stable or not. You need to justify your answer via a counter-example or a proof.

b) In Radix sort, we iteratively and digit by digit (often starting from the least significant digit, moving towards most significant digit). In the i'th iteration, a steady sorting method A is used to sort items based on their i'th digit; given that the A is steady, at the end of the i'th iteration, the numbers will be sorted with respect to their rightmost i digit. Here is an example:

123		581		123		045
435		123		435		123
396		763		045		246
45		394		246		257
257	→ sort by the rightmost digit	435	→ sort by the second digit	257	→ sort by the third digit	394
394		045		763		396
246		396		581		435
581		246		394		581
763		257		396		763

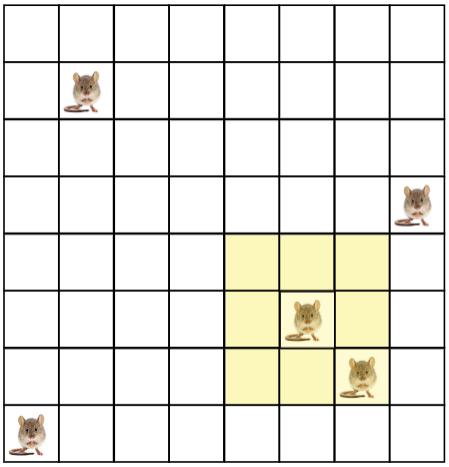
Note that if a linear-time stable sorting algorithm is used as A, the running time of Radix Sort will be $\Theta(n \times b)$.
Apply Radix Sort to sort the following input. Show your work.

$$[624, 238, 180, 384, 1006, 446, 164, 1, 184]$$

Problem 2 Decision Trees [5 marks]

We need to locate 5 naughty mice who are hiding underneath a $n \times n$ grid. Suppose each cell has space for one mouse. You cannot see the mice, and your only tool for finding them is to use a *query* the grid, where each query is a square (of an arbitrary size), and the answer is the number of rats within the square. In the figure below, the grid size is 8×8 (we have $n = 8$), and the answer for the highlighted query square is 2.

Use a decision tree approach to give a precise (not big-Omega) lower bound for the number of queries needed to locate the position of all 5 mice. Justify your answer in a few sentences.



Problem 3 Linear-time Sorting [3+4 = 7 marks]

a) Apply Bucket-Sort on the following input. Show your work in a similar way that we did in the class. Assume we use 10 buckets.

$$A = [0.88, 0.32, 0.15, 0.6, 0.55, 0.11, 0.52, 0.12, 0.83, 0.31, 0.19, 0.81]$$

b) We saw in the class that Bucket Sort is useful when the input is uniformly distributed. Consider the following input which is NOT uniformly sorted: $n/\log n$ items are uniformly distributed in $[0, 0.8]$, $\log^2 n$ items are uniformly distributed in $(0.8, 0.9]$ and the remaining $n - n/\log n - \log^2 n$ items are uniformly distributed in $(0.9, 1]$.
Repeat the analysis from the class to describe the expected running time of Bucket-Sort with parameter $k = n/20$ for the above input. You may assume that we a comparison-based sorting algorithm with running time $\Theta(n \log n)$ to sort items within each bucket.

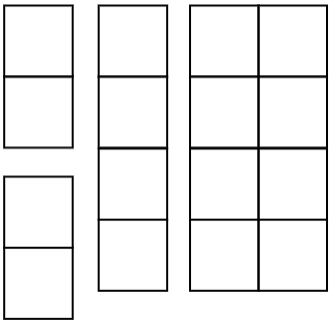
Problem 4 Dynamic Programming I [5 marks]

Consider a variant of the rod cutting problem in which, in addition to length n of the rod, you are given an integer $k \geq 1$ that specifies the maximum number of cuts you are allowed to make. For example, if $k = 1$, you are allowed to make 0 or 1 cuts and not more. As before, the values of subrods are stored in an array p. Let $L[i, q]$ be the maximum value you can get for cutting a rod of length i when you are allowed to make q cuts (we have $i \leq n$ and $q \leq k$). Write down the recursive formula for $L[i, q]$.

Problem 5 Dynamic Programming II [5+3=8 marks]

Given a woorden board of size $n \times n$ for some integer $n > 1$, we want to cut the board to maximize the profit. For each pair (i, j), the value (profit) of a board of dimension $i \times j$ is stored in $p[i, j]$, where p is a two-dimensional matrix and $i, j \in \{1, 2, \ldots n\}$. For instance, if $n = 4$, the matrix p can be as follows:

1	3	4	8
3	6	7	9
4	7	10	13
8	9	14	15



For example, the value of a subboard of dimensions 2×3 in the above example is 7. The cuts take place vertically or horizontally. The total profit of an algorithm is the sum of subboard it outputs. In the above example, a board of length 4×4 can be first cut vertically at index 1 to output a subboard of size 4×1 and a subboard of size 4×3. The subboard of size 4×1 may be further partitioned horizontally to generate to subboards of size 2×1, and the subboard of size 4×3 may be cut vertically into two subboards of size 4×1 and 4×2 (see the figure). The profit of this particular cutting is therefore $2 \times 3 + 8 + 9 = 23$. Note that there are many other possible cuttings, for example, one may choose not to cut at all and get a profit of 15.

Let $L[i, j]$ be the maximum profit that one can achieve from cutting a board of size $i \times j$.

a) Write down the value of $L[i, j]$ recursively.

b) Write a dynamic programming pseudocode to compute $L[i, j]$, for $i \in [1, n]$ in a bottom-up approach.