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

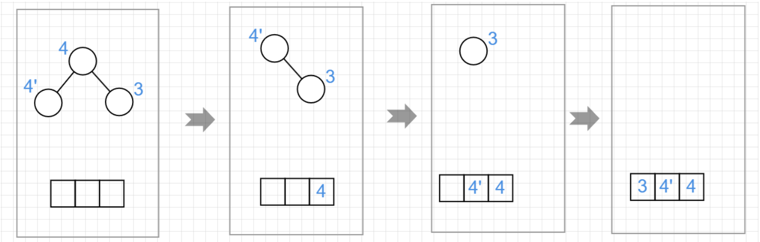
All problems are written problems; submit your solutions electronically **only via Crowdmark**. You are welcome to discuss the general idea of the problems with other students. However, you must write your answers individually and mention your peers (with whom you discussed the problems) in your solution.

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.

Answer: Heap sort is not stable. For example, let $A = [4, 4', 3]$. After extracting 4, the node $4'$ becomes the new root of the heap, and the sorted array will be $[3, 4', 4]$.



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

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

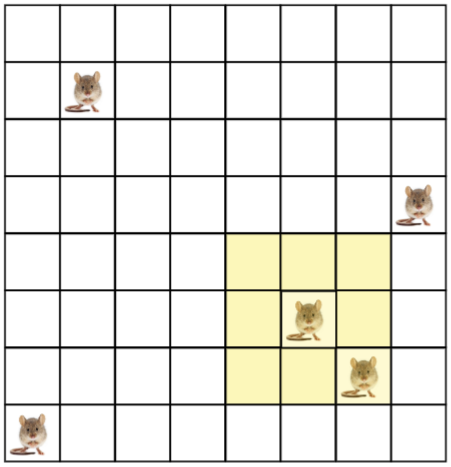
Answer:

0624	→ sort by the right-most digit	0180	→ sort by the second digit	0001	→ sort by the third digit	0001	→ sort by the forth digit	0001
0238		0001		1006		1006		0164
0180		0624		0624		0164		0180
0384		0384		0238		0180		0184
1006		0164		0446		0184		0238
0446		0184		0164		0238		0384
0164		1006		0180		0384		0446
0001		0446		0384		0446		0624
0184		0238		0184		0624		1006

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 mice 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.



Answer: Given that the mice are indistinguishable, there are n^2 possible positions, out of which 5 include mice. In other words, there are $\binom{n^2}{5}$ possibilities for the location of the 5 mice, that is, there are $\binom{n^2}{5}$ possible inputs, each associated with a leaf of the decision tree.

Next, we check the number of possible outputs for each query. Each query answer indicates that there are $i \in \{0, 1, 2, 3, 4, 5\}$ mice in the query box. That is, there are at most 6 possible outcomes for each query. In other each internal node of the decision tree has at most 6 children.

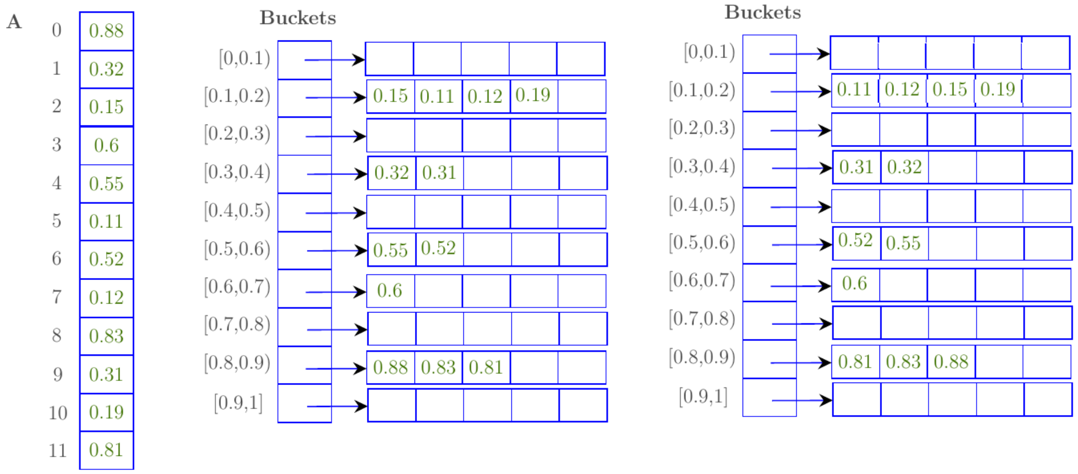
Therefore, each algorithm becomes a decision tree with $\binom{n^2}{5}$ leaves and at most 6 branch at each internal node. The height of that would specify the worst-case running time of the algorithm. A tree with $\binom{n^2}{5}$ leaves and 6 branch per internal node has a height of at least $\log_6 \binom{n^2}{5}$ (realized when the tree is as balance as possible). Therefore, the worst-case running-time of any algorithm is at least $\log_6 \binom{n^2}{5}$.

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.

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

Answer:



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.
Answer: As before, distributing items between buckets takes $\Theta(n)$.

- The first $0.8k = 0.8n/20 = n/25$ buckets receive $n/\log n$ of the items. The expected number of items in each bucket is thus $\frac{n/\log n}{n/25} = 25/\log n = o(1)$. That is, for large values of n, the expected number of items within each bucket converges to 0. Therefore, sorting each of these buckets takes $O(1)$.
- The second group of $0.1k = 0.1n/20 = n/200$ buckets receive $\log^2 n$ of the items. The expected number of items in each bucket is thus $\frac{\log^2 n}{n/200} = o(1)$. As in the previous group, for large values of n, the expected number of items within each bucket converges to 0. Therefore, sorting each of these buckets takes $O(1)$.
- The last group of $0.1k = 0.1n/20 = n/200$ buckets receive $n - n/\log n - \log^2 n$ of the items. The expected number of items in each bucket is thus $\frac{n - n/\log n - \log^2 n}{n/200} = \Theta(1)$. That is, for large values of n, the expected number of items within each bucket is a constant. Therefore, sorting each of these buckets takes $O(1)$.

To conclude, sorting each bucket takes $O(1)$. Therefore, the overall complexity of the algorithm for all $n/20$ buckets is $\Theta(n)$.

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 $L[i, q]$.

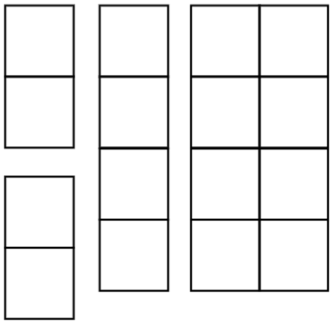
Answer: If $i = 0$, there is no rod and the profit will be 0. Similarly, if $q = 0$, we cannot make any more cut, and the profit for a rod of length i would be $p[i]$. Otherwise, we can make a cut of length $j \in \{1, \ldots, i\}$, collect value $p[j]$ plus whatever profit that can be achieved from cutting the remainder of the rod, i.e., $L[i-j, q-1]$. Note that the number of cuts q is decremented because we just used one cut.

$$L[i,q] = \begin{cases} 0 & \textbf{if} \quad i = 0, \\ p[i] & \textbf{if} \quad q = 0, \\ \max\limits_{j \in \{1,\ldots,i\}} (p[j] + L[i-j, q-1]) & \textbf{otherwise.} \end{cases}$$

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.
Answer:

If $i < 0$ or $j < 0$, there is no board left and the profit will be 0. Otherwise, we We have the following options for cutting a board of size $i \times j$:

- Do not cut the board. Then the profit will be simply $p[i, j]$.
- Cut the board vertically at index $k \in \{1, \ldots, i-1\}$. The result will be a sub-board of size $k \times j$ and another sub-board of size $(i - k) \times j$. The profit we collect from the best such vertical cut will be $\max_{k\in\{1,\ldots,i-1\}} (L[k, j] + L[i - k, j])$.
- Cut the board horizontally at index $x \in \{1, \ldots, j - 1\}$. The result will be a sub-board of size $i \times x$ and another sub-board of size $i \times (j - x)$. The profit we collect from the best such vertical cut will be $\max_{x\in\{1,\ldots,j-1\}} (L[i, x] + L[i, j - x])$.

Therefore, we can write

$$L[i, j] = \max\{p[i, j], \max_{k\in\{1,\ldots,i-1\}} (L[k, j] + L[i - k, j]), \max_{x\in\{1,\ldots,j-1\}} (L[i, x] + L[i, j - x])\}$$

.

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

```
FillDPTable($p, n$)
for $i = 1$ to $n$
    for $j = 1$ to $n$
        $option\_1 \leftarrow p[i, j]$
        $option\_2 \leftarrow -\infty$
        for $k = 1$ to $i - 1$
            if $L[k, j] + L[i - k, j] > option_2$
                $option_2 \leftarrow L[k, j] + L[i - k, j]$
        $option\_3 \leftarrow -\infty$
        for $x = 1$ to $j - 1$
            if $(L[i, x] + L[i, j - x] > option_3$
                $option_3 \leftarrow L[i, x] + L[i, j - x]$
        $L[i][j] \leftarrow \max\{option_1, option_2, option_3\}$
```