York University
LE/EECS 3101 B, Fall 2022
Assignment 2

Due Date: October 16th, at 23:59pm

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. Please refer to the course webpage for guidelines on academic integrity.

Problem 1 [Bonus] Max Value SubMatrix [4+3+3 = 10 marks]

Let A be an n by n matrix of integers. A "cross" of A is a formed by a contiguous set of items in row i and a contiguous set of items in a column j of A that both contain $A[i, j]$. The "value" of a cross is the sum of the numbers in the cross. In the example below, items highlighted in red form a cross of value -15 with $i = 2$ and $j = 4$, and items in blue form another cross of value 16 with $i \in \{6, 7, 8\}$ and $j = 7$. Note that, with this definition, a set of items in a single row or column or even a single item form a cross.

Present a *divide-and-conquer* algorithm that returns the maximum value of the crosses in A. Do your best to make your algorithm as fast as possible.

-1	18	-4	3	2	7	11	5
-12	1	14	-6	1	8	-7	-5
-3	28	54	-13	2	17	6	35
2	12	14	-16	1	8	-17	-12
-12	8	-3	3	3	9	13	5
3	28	44	-23	2	17	6	45
13	3	13	-6	1	18	-7	-5
2	12	14	26	1	8	17	-12

(a) Describe your algorithm in English or pseudocode.

(b) What is the running time of your algorithm in the *worst case*? Write the recursive formula for the running time and justify your answer. You may assume n is a power of 2.

Problem 2 Matrix Multiplication [4+4 = 8 marks]

Justin is a smart programmer who has managed to correctly implement both matrix multiplication algorithms we saw in the class. Due to a performance issue in the code, however, the addition of two matrices of size $n \times n$ takes $\Theta(n^{2.5})$ instead of $\Theta(n^2)$.

(a) Write down the recursive formula for the running time of the simple divide-and-conquer algorithm of Slide 24 in Justin's implementation. State the asymptotic time complexity (using Θ notation) and indicate whether the running time changes because of the bug.

(b) Write down the recursive formula for the running time of the Strassen algorithm of Slide 24 in Justin's implementation. State the asymptotic time complexity (using Θ notation) and indicate whether the running time changes because of the bug.

Problem 3 Heap Operations [3+3+4 = 10 marks]

(a) Consider a heap stored in an array $a = [25, 17, 23, 14, 8, 21, 12, 7]$. Write the updated array when we apply `extractMax` operation.

(b) Consider the same heap $a = [25, 17, 23, 14, 8, 21, 12, 7]$. Write the updated array when we apply the operation `insert(24)`.

(c) Specify the best-case running time of `extractMax()` in a heap formed by n distinct items. If your answer is $\Theta(f(n))$, you must provide an example, with large n, for which extractMax() takes $\Theta(f(n))$.

Problem 4 Reheap (Heapify) Complexity [2+3+3=8 marks]

Recall that the *height* of a node in a binary tree is the number of edges (links) from the node to the deepest leaf in its subtree (e.g., the height of leaves is 0).

To answer the following question, consider a complete binary tree T of size n in which the last level contains all possible nodes from left to right.

(a) Specify the number of nodes at height h in T for any $h \geq 0$ (no proof is needed).

(b) Specify the asymptotic running time of Bubble-Down for a node of height h as a function of h. Then show the total time complexity to Bubble-Down all nodes at level h of T when we apply the Heapify operation on T. Justify your answer in one or two sentences.

(c) Use your answer in part to prove that the time complexity of the Heapify operation is $O(n)$. You need to sum the total work over all levels. Show your work.
Hint: $\sum_{x=0}^{\infty} \frac{x}{2^x} = \Theta(1)$.

Problem 5 Quick-Select [7 marks]

When doing Quick-Select and Quick-Select, it is desired to have a *good* pivot which is almost in the middle of the sorted array. When doing the average-case analysis of Quick-Select, we considered a *good* and a *bad* case; the good case happened when the pivot was among the half middle items of the sorted array, i.e., we had $n/4 \leq i < 3n/4$ (i is the index of pivot in the partitioned array). In our analysis, we provided an upper bound for the time complexity of the algorithm in the *good* case and showed that $T(n) \leq T(3n/4) + cn$ in these cases for some constant c. Since the good case happened with probability $1/2$, we could prove that the algorithm runs in linear time on average (see the recursion slide 12 of lectures on selections).

Change the definition of the good case and assume the good case happens when we have $n/3 \leq i < 2n/3$. Provide an upper bound for $T(n)$ and use that to show that Quick-Select runs in $O(n)$.
Hint: start by calculating the probability of good case and bad case happening.

Problem 6 Median-of-Three Algorithm [4+4=8 marks]

Consider a variant of Median-of-Five algorithm in which, instead of partitioning input into $n/5$ blocks of size 5, we partition the input into $n/3$ blocks of size 3.

a) Follow the same steps as in the slides to derive a recursive formula for the time complexity $T(n)$ of this algorithm.

b) Try to solve the recursion by guessing that $T(n) \in O(n)$. Follow the same steps as in the slides and indicate whether we can state $T(n) \in O(n)$.