Monsoon 2025 Graduate Algorithms: Mid-sem *Max: 50 points*
Date: **22 September** *Duration: 120 mins*

10% if you write **"I don't know"** for any (sub)question; with a cap of total 3 marks. A faster algorithm will get more credit but only if it is correct. A slower algorithm that is correct may get more marks than a faster incorrect algorithm. None of the recursive algorithms can use non-constant space and non-readonly global variables and data structures.

Q1 [3+2=5 points] (a) Design a recursive algorithm MyAlgo(A[1... n]) with the following API: Under the condition that $A[1 \dots n-1]$ is sorted, MyAlgo should return the number of "inversions" among all pairs of array elements in A, and further, after MyAlgo(A) returns, A[1 ... n] must be sorted. There is no API requirement if $A[1 \dots n-1]$ is not sorted when MyAlgo(A[1...n]) is called. The only operation on A that MyAlgo is allowed is swapping of elements. Add explanations to the steps of the algorithm.

Two array elements A[i] and A[j] are said to be inverted if $A[i] > A[j]$ but $i < j$; for example, $[0, 1, 13, 8]$ has 1 inversion: $(A[3], A[4])$ and $[10, 7, 9, 12, 8]$ has 5 inversions at these pairs of indices: (1,2), (1,3), (1,5), (3,5), (4,5). For simplicity, you can assume that A has distinct elements.
(b) Analyse the number of swaps done by MyAlgo on an n-sized array by writing a recurrence and solving it using either the substitution method or induction.

Q2 [3 + 2 + 2 = 7 points] This question is about quicksort variations that employ different strategies for choosing their pivots.
(a) Write the pseudocode of MomSort which is a version of QuickSort along with lines of MomSelect. Specifically, MomSort first computes the MoM (median of medians of groups of 5) and then uses it as the pivot. MomSort should take as input an array and after it returns, the array must be sorted in-place. You can ignore floors and ceilings in the pseudocode. You need to write the code of MomSelect, if you are calling it. You do not need to write the code for partition or computing the median of constant-sized arrays.
(b) Write down the recurrence for the time-complexity of MomSort. You do not have to solve it. If you are calling MomSelect, you can directly use its complexity.
(c) Let MedianQSort be another version of Quicksort(A[1...n]) that uses the element returned by MomSelect$(A, |A|/2)$ as its pivot. Write down the recurrence for the time-complexity of MedianQSort and then find a solution to the recurrence.

Q3 [2 + 1 + 8 + 4 = 15 points] Consider an array $A[1 \dots n]$ with distinct elements. This question is about subarrays of A. A subarray is a contiguous portion of A; e.g., $[6, 1, 9]$ has these non-empty subarrays: $[6], [1], [9], [6, 1], [1, 9], [6, 1, 9]$. The task is to compute the sum of the minimum elements of every subarray of A (minimum element of the empty subarray is 0). For example, it should return 19 for A (0 is the minumum element of the empty subarray, 1 is the minimum element of the subarrays $[6, 1]$, $[6, 1, 9]$, $[1]$, $[1, 9]$, then 6 is the minimum element of the subarray $[6]$, and finally 9 is the minimum element of $[9]$; thus, the algorithm should return $0 + 1 + 1 + 1 + 1 + 6 + 9 = 19$).

You may freely use a subroutine $MinArray(B)$ that takes as input an array and returns the index of the minimum element in B; the complexity of $MinArray$ is $O(n)$ on an n-sized array.

(a) One approach towards designing an algorithm is to generate all subarrays of A, identify the minimum element in each, and compute their sums. What would be the complexity of this approach, assuming that you can enumerate all subarrays in O(1) time per subarray. Explain the derivation of complexity.
Hint: First, write down an expression for the number of non-empty subarrays of A.
(b) Suppose the minimum element of A appears at $A[i]$. Write down an expression for the number of non-empty subarrays of A whose minimum element is $A[i]$. Explain.
(c) Write down a polynomial-time divide and conquer algorithm to return the sum of the minimum elements of every subarray of A. Space usage should be constant, i.e., the space required for local variables, parameters, and return values should be $O(1)$.
Hint: Use (b).
(d) Write down a recurrence for the time-complexity of your algorithm in (c). A solution is not required but you will get 5 BONUS points for solving it (less than 3 BONUS points will not be added).

Q4 [13 points] Your company has two offices, one in San Francisco (SF) and the other in New York (NY). Each week you decide whether you want to work in the San Francisco office or in the New York office. Depending on the week, your company makes more money by having you work at one office or the other. You are given a schedule of the profits you can earn at each office for the next n weeks. You'd obviously prefer to work each week in the location with higher profit, but there's a catch: Flying from one city to the other costs $1000. Your

task is to design a travel schedule for the next n weeks that yields the maximum total profit, assuming you start in Buffalo (so, you have to fly to either SF or NY no matter what you decide).

Example: Consider the following instance showing the schedule of the profits for the next three weeks (all amounts are in USD). If you always switch to the city that pays the most every week, you earn a profit of \$0 (fly to SFO, fly to NYC, fly to SFO); a better option is to fly to SFO and then remain there – this will earn a profit of \$1500.

San Franisco	1000	500	1000
New York	500	1000	500

Design a 6-stage dynamic programming algorithm to compute the maximum total profit you can earn. The input to your algorithm is a pair of arrays $NY[1 \dots n]$ and $SF[1 \dots n]$, containing the profits in each city for each week. Add explanations to the relevant steps.

Q5 [1+1+1+2+5=10 points] Below I have displayed the edit table between the strings A="112233" and B="312321". I am adding an additional copy in case you need to scribble anything on them.

i\j		0	1	2	3	4	5	6
	A[i] \ B[j]	-	3	1	2	3	2	1
0	-	0	1	2	3	4	5	6
1	1	1	1	1	2	3	4	5
2	1	2	2	1	2	3	4	4
3	2	3	3	2	1	2	3	4
4	2	4	4	3	2	2	2	3
5	3	5	4	4	3	2	3	3
6	3	6	5	5	4	3	3	4

i\j		0	1	2	3	4	5	6
	A[i] \ B[j]	-	3	1	2	3	2	1
0	-	0	1	2	3	4	5	6
1	1	1	1	1	2	3	4	5
2	1	2	2	1	2	3	4	4
3	2	3	3	2	1	2	3	4
4	2	4	4	3	2	2	2	3
5	3	5	4	4	3	2	3	3
6	3	6	5	5	4	3	3	4

I am also adding two empty tables in case you need them.

i\j		0	1	2	3	4	5	6
	A[i] \ B[j]	-	3	1	2	3	2	1
0	-							
1	1							
2	1							
3	2							
4	2							
5	3							
6	3							

i\j		0	1	2	3	4	5	6
	A[i] \ B[j]	-	3	1	2	3	2	1
0	-							
1	1							
2	1							
3	2							
4	2							
5	3							
6	3							

(a) Write down any optimal edit sequence from A to B that does not involve any insertion. Show the transformation of the strings after each step, along with the specific changes (no-op, insert ____, delete ____, etc.).

(b) Write down any optimal edit sequence from A to B whose last operation (assuming that A is transformed from its left end) is a delete.

(c) Write down the definition of $Half(5, 6)$ with respect to A and B.

(d) Compute the value of $Half(5, 6)$ and justify your answer. For this, you can either use the recursive formula for $Half()$ or its definition in (c), but you must show the steps of whatever method you used.

(e) Define a subproblem $N(i, j)$ where i, j are indices of A and B, respectively, as $N(i, j)$ = the number of optimal edit sequences from $A[1 \dots i]$ to $B[1 \dots j]$. Write down a recursive formulæ to compute $N(i, j)$. You can use the Edit-table values in your formulæ.