Honesty statement:

I HAVE ADHERED TO THE COLLABORATION POLICY FOR THIS CLASS. IN OTHER WORDS, EVERYTHING WRITTEN DOWN IN THIS SUBMISSION IS MY OWN WORK. FOR PROBLEMS WHERE I RECEIVED ANY HELP, I HAVE CITED THE SOURCE, AND/OR NAMED THE COLLABORATOR.

Signed and Acknowledged, Johny B Good

1. (Regular) We discussed the invariant for SELECTIONSORT in class. You can find the pseudocode and invariants for INSERTIONSORT and MERGESORT in Chapter 2 of your text (Cormen, et al.) [CLRS]. Write down precise invariants for BUBBLESORT (see p40 of text) and MERGE (not MERGESORT) (see p31 of text).

Solution 1: Invariant for MERGE can be found on p32 of [CLRS]. It states:

At the start of each iteration of the for-loop of lines 12 through 17, the subarray $A[p..(k-1)]$ contains the $k-p$ smallest elements of $L[1..(n_1+1)]$ and $R[1..(n_2+1)]$ in sorted order. Moreover, $L[i]$ and $R[j]$ are the smallest elements of their arrays that have not been copied back into $A$.

The proof (not asked for in the problem) is in the text p32-34.

Invariant for BUBBLESORT (pseudocode on p40 of [CLRS]) is:

At the start of each iteration of the for-loop for $i$ (lines 1 through 4), the subarray $A[1..(i-1)]$ contains the $i-1$ smallest elements of $A$ in sorted order.

Proof is not required for this problem. Proof requires you to show Initialization at start of iteration $i = 1$ (trivial!) and Maintenance, i.e., at start of arbitrary iteration $i$. A more detailed invariant is possible by adding something about the for-loop for $j$ on lines 2-4.

Always start a fresh problem on a fresh page. Since you do not need to print out your solutions, we will not be killing any trees in the process.
3. (Regular) Most sorting algorithms work correctly even if all items are not unique, i.e.,
there are repeats. A sorting algorithm is called stable if numbers with the same value
appear in the output array in the same order as they do in the input array. Which
of the algorithms SELECTIONSORT, INSERTIONSORT and BUBBLESORT are stable? If
they are not stable give a small example with at most 4 items to prove your answer.

Solution 3: Consider an input array \( A \) with 3 items \([2_1, 2_2, 1]\) in that order. The array
has two copies of the number 2. In order to distinguish them we label them as \( 2_1 \) and
\( 2_2 \), where \( 2_1 \) appears before \( 2_2 \) in the input array. If you apply algorithm XXXSORT
(refer to pseudocode on page XX of [CLRS]) then because of \( \cdots \) and line \( \text{mm-nn} \) in
pseudocode, the output would be \( \cdots \) proving that XXXSORT is not a stable sorting
algorithm.
X. (Regular) Design an algorithm to …

Solution X:

Basic Idea: The basic idea behind my solution is the following observation. We know that … Thus … However, … In order to take care of this problem, we do the following … The following pseudocode for FictionFlight provides the details of the idea.

Pseudocode:

Algorithm 1 FictionFlight

1: Assume bird starts at \((r_R, c_R)\) of grid
2: \(s \leftarrow \) length of command seq \(S\)
3: \(D[0, 0, r_R, c_R] = \text{true}\)
4: \((\text{prev}R, \text{prev}C) = (r_R, c_R)\)
5: for \(t = 1\) to \(s\) do
6: \(D[0, t, \text{move}(s[t], \text{prev}R, \text{prev}C)] = \text{true}\)
7: \((\text{prev}R, \text{prev}C) = \text{move}(s[t], \text{prev}R, \text{prev}C)\) \(\triangleright\) This trick is explained above
8: end for
9: for \(e = 1\) to \(\text{numRows} \times \text{numCols}\) do
10: for \(t = 1\) to \(s\) do
11: for \(r = 1\) to \(\text{numRows}\) do
12: for \(c = 1\) to \(\text{numCols}\) do
13: \(D[e, t, r, c] = D[e - 1, t - 1, r, c] \text{ or } D[e - 1, t - 1, \text{NGBR}(r, c)]\)
14: end for
15: end for
16: end for
17: if \((D[e, s, r_E, c_E) = \text{true})\) then
18: Return \(e\)
19: end if
20: end for

Correctness: This is not a formal proof of correctness. However, the algorithm must be correct because it uses the \(XX\) algorithm from the book (page \(ZZ\)) and algorithm \(\text{move}\) (discussed in Lec \(W\) of class) with a minor modification, and hence must be correct.

Time Complexity Analysis: The pseudocode suggests that the time complexity can be written down using the following recurrence:

\[
T(n) = 2T(n/2) + O(n)
\]

Solving this recurrence using case 2 of the Master Theorem gives us

\[
T(n) = \Theta(n \log n)
\]