1. **Problem Description**: We have a stack of n textbooks that need to be sorted, such that the frontmatter of each textbook is facing up. For example, initially, we may have the following stack (m=7):

1	0
4	1
5	1
7	0
3	1
2	1
6	0

where the first number in each row shows the desired order of the textbook, and the second number shows whether the cover is frontmatter of the textbook is facing up (1) or facing down (0).

The goal is to sort the textbooks so that they form the following stack:

1	1
2	1
3	1
4	1
5	1
6	1
7	1

However, to sort the stack, we are only allowed to select one of the textbooks, lift that textbook and all textbooks on top of it,[1] and flip them altogether. For example, assume that we select textbook 7 in the following stack:

1	0
4	1
5	1
7	0
3	1
2	1
6	0

Flipping that textbook and everything above it will give us the following stack of textbooks:

7	1
5	0
4	0
1	1
3	1
2	1
6	0

[1]The top textbook has no textbook on top of it, so in case it is selected, it will be the only textbook that flips.

We wish to start from an arbitrary stack, and do a finite number of flips and sort the textbooks.

2. The goal of this Lab is that you write a program that takes an initial stack of textbooks of size n, and sorts them using A* Search. A* uses a priority queue as its fringe. In order to implement the priority queue efficiently, you are recommended to use a binary heap. The cost function $g(n)$ is the number of flips so far. The heuristic function that you will use is the number of pairs in the stack that satisfy one the following conditions:

   (a) If the pair of books are not adjacent in the ordered stack, regardless of being face up or face down.

   (b) If the pair has a book facing up and one facing down.

   (c) If the pair is wrongly ordered, but with correct orientations (both facing up)

   (d) If the pair is correctly ordered, but with wrong orientations (both facing down)

   For example, in

1	1
2	1
3	1
4	0
5	0
7	1
6	1
8	1
9	1

   (6,8) is of the first type, (3,4) is of the second type,(opposite orientations), (7,6) is of the third type (correct orientations, wrong order), and (4,5) is of the fourth type (wrong orientations, correct order). There is also the pair (5,7) which falls in both the first and the second category. This pair is of course only counted once to maintain admissibility. (so, a pair is counted if it is of first, second, third, or fourth type, and this is LOGICAL OR). So, $h = 5$.

   We have provided a skeleton code with extensive comments in the README.md file that you should read. Please use the skeleton code and submit your code to the GitHub repository that will be provided to you.

   Here is how your code will be graded:

   - General soundness of code: 60 pts.
   - Passing multiple test cases: 40 pts.

3. (20 pts **Extra Credit A* with Dynamic Weighting**): One way of dealing with large state spaces is to use weighted A* algorithm, which uses:

$$f(n) = g(n) + wh(n)$$

Weighted A* uses the heuristic to find the goal faster, although it compromises optimality. If $w = 0$, it becomes Uniform Cost Search. If $w = 1$, it is the classic A* algorithm. The larger the w, the more it resembles greedy search. An idea to benefit from both the heuristic and the cost function is to use dynamic weighting, which uses a large weight at the beginning and uses greedy search to take us "somewhere" first, and then "fine-tuning " using A*.

The weighted cost function is:

$$f(n) = g(n) + w(n)h(n)$$
$$w(n) = 1 + \epsilon - \frac{\epsilon d(n)}{N}$$

$d(n)$ is the depth of the current search and N is an upper bound on the search depth $(N \geq d(n), \forall n)$. Implement the A* search with dynamic weight using $\epsilon = 1$ and $N = 2m$, where m is the size of the stack of textbooks. Create a table and record the average number of flips needed for ordering over all permutations of textbooks when $m \in \{1, 2, \dots, 8\}$ when you use A* and dynamically weighted A*.

m	A*	Dynamically weighted A*
1		
2		
3		
4		
5		
6		
7		
8		

Also, make a similar table about the average number of nodes visited. Provide your analysis based on those two tables. Submit a PDF file of both of the tables along with your code.

Note: You must submit Extra Credit along with your main homework. If you submit Extra Credit Late, your whole homework will be late.