Project 3: Number Placement Puzzle

CS 3343: Analysis of Algorithms Summer 2024

Instructor: Dr. Mohammad Imran Chowdhury

Total: 100 Points

Due: 07/09/2024 11:59 PM

In this project, I invite you to solve the ``Number Placement Puzzle" using the **transform and conquer algorithmic technique**, and Python programming discussed in class.

Problem Description:

You are given a set of distinct numbers. And you are given inequalities between them. Your goal is to place all the numbers into the inequalities in a way that satisfies all the comparisons. e.g., arrange the puzzle numbers so that the puzzle statement is true.

Transform and Conquer Algorithmic Approach:

If you sort the numbers first, then a much more efficient algorithm is possible. That is the **transformation**.

The algorithm also makes use of the ``**two-pointer"** technique which is used in quite a few algorithms and is very useful. We keep track of where we are in an array or list by having a low pointer and a high pointer (they are not really pointers in the programming sense as such – just variables containing the indices of the highest and lowest positions in the array at the current stage of the algorithm). You may have seen this technique before, for example in the famous **Binary Search algorithm**.

The basic logic of the transform-and-conquer solution is this:

- Sort the puzzle numbers first
- Iterate through the inequality signs
- Use the largest remaining puzzle number if the sign is >, and the smallest remaining if <, and delete the puzzle number from the list each time.
- The deleting works fine if solving the puzzle on paper/whiteboard. In a program, it is probably better to implement this using the "two-pointer" method, as described above.

Note: since we have stipulated that the puzzle numbers must be distinct, the puzzle will always have a solution. Phew!

Simulation:

Let's consider the step-by-step simulation below to see how the transform and conquer algorithmic approach works here to solve the following number placement puzzle.

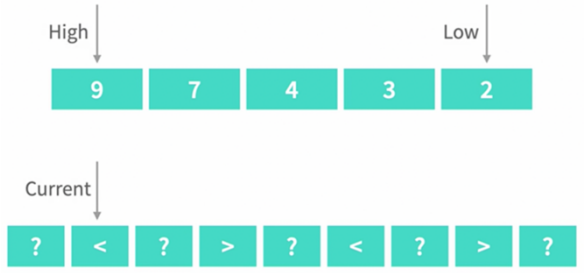

Fig. 1

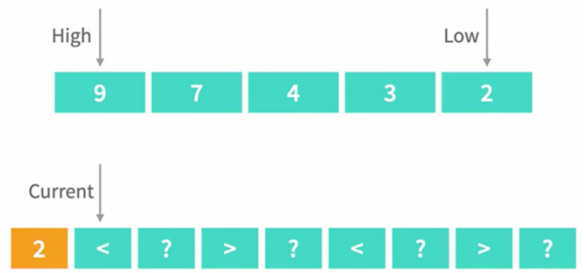

Fig. 2

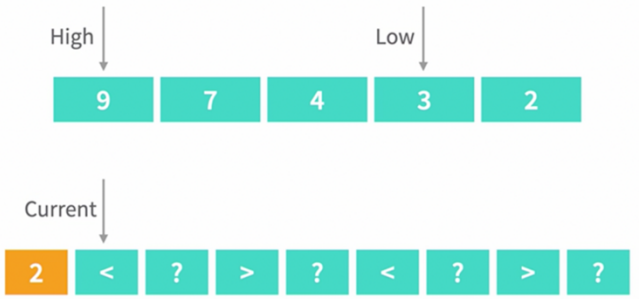

Fig. 3

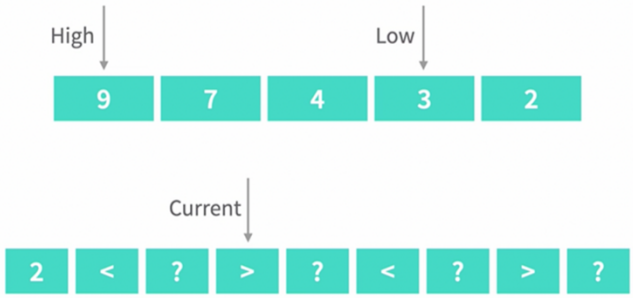

Fig. 4

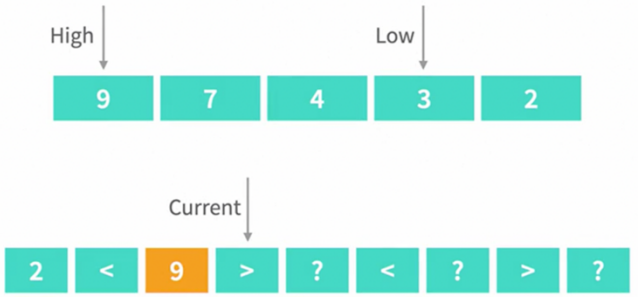

Fig. 5

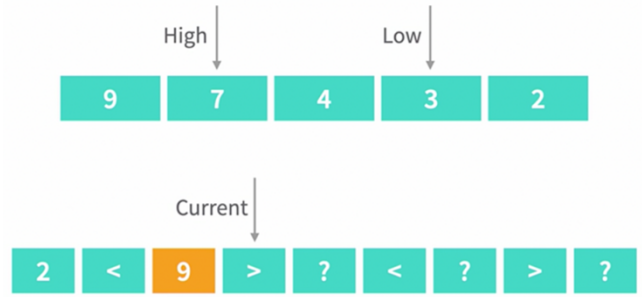

Fig. 6

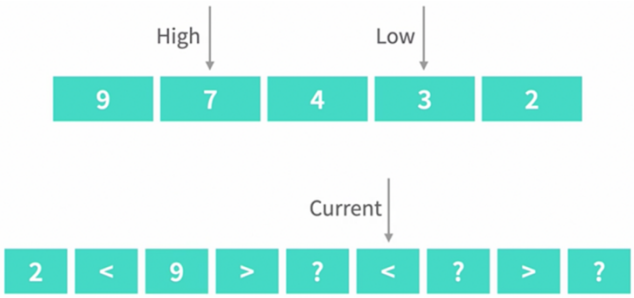

Fig. 7

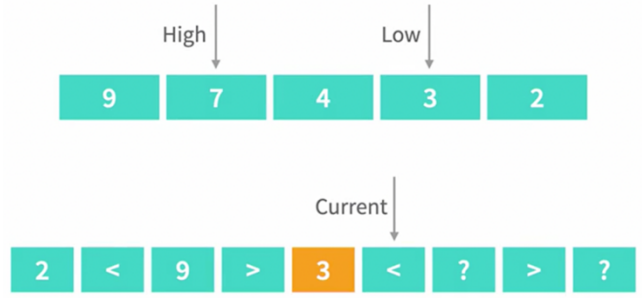

Fig. 8

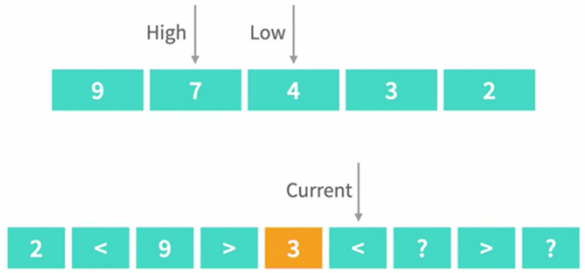

Fig. 9

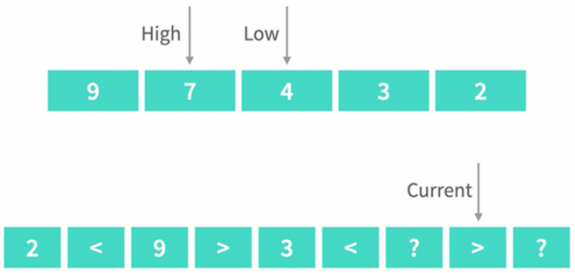

Fig. 10

Fig. 11

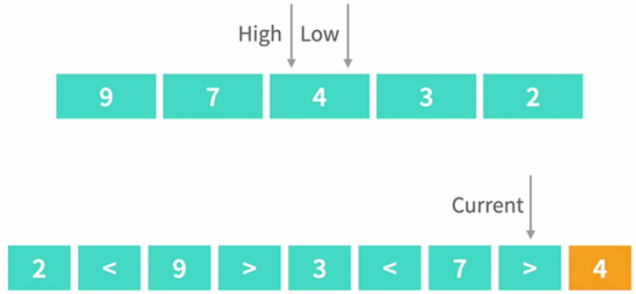

Fig. 12

Source Code:

You need to start work with the following source code also available through the project folder found on the CANVAS course website i.e., the starter file **number_placement_algorithm.py**, and complete the **TO-DO portion** to solve the ``Number Placement Puzzle" using the **transform and conquer algorithmic technique**, and python programming discussed in class.

```
import random

PUZZLE_SIZE = 10

# Create random puzzle
# random.sample ensures no duplicates
puzzle_nums = random.sample(range(100), PUZZLE_SIZE)
puzzle_symbols = []

# Randomly assign inequalities
for i in range(PUZZLE_SIZE - 1):
    puzzle_symbols.append(">" if random.random() < .5 else "<")

print("Random puzzle numbers: ", puzzle_nums)
print("Randomly assign inequalities: ", puzzle_symbols)
```

```
# Sort puzzle numbers first
# Use largest remaining if greater than, smallest remaining if less than
sorted_puzzle_nums = sorted(puzzle_nums, reverse=True)

# Vars for the "two pointer" method
high = 0
low = PUZZLE_SIZE - 1

# store solution values
solution_values = []

# TO-DO #
# Your code goes here
# Iterate through the inequalities and apply solution algorithm to populate solution  values
# You may need to add the last value outside the loop
pass

# Convert solution  values to list of strings
solution_values = list(map(str, solution_values))

# Create a list to store the final solution representation
final_solution_representation = [None] * (len(solution_values) + len(puzzle_symbols))

# Create solution representation using nifty slicing with step parameter
final_solution_representation[::2] = solution_values
final_solution_representation[1::2] = puzzle_symbols

# Display final solution
# The join() method takes all items in an iterable and joins them into one string
print("Final solution: ", " ".join(final_solution_representation))

# Evaluate whether solution is correct.
print(eval(" ".join(final_solution_representation)))
```

Sample I/O:

Run 1:

Random puzzle numbers: [1, 83, 14, 93, 0, 96, 13, 24, 70, 40]
Randomly assign inequalities: ['<', '>', '>', '>', '>', '<', '>', '<', '>']
Final solution: 0 < 96 > 93 > 83 > 70 > 1 < 40 > 13 < 24 > 14
True

Run 2:

Random puzzle numbers: [18, 52, 99, 8, 97, 73, 26, 74, 29, 36]
Randomly assign inequalities: ['>', '<', '<', '>', '>', '>', '<', '<', '<']
Final solution: 99 > 8 < 18 < 97 > 74 > 73 > 26 < 29 < 36 < 52

True

Run 3:
Random puzzle numbers: [86, 97, 40, 78, 46, 60, 69, 75, 58, 55]
Randomly assign inequalities: ['>', '<', '>', '>', '>', '<', '<', '<', '>']
Final solution: 97 > 40 < 86 > 78 > 75 > 46 < 55 < 58 < 69 > 60
True

The submission grading rubric is as follows (points out of 100 total):

Project element	**Points**
Code readability such as the usage of comments in code	10
Proper coding implementation	70
Screenshots of the program output	10
Program's sample Input/Output format matching as per project writeup	10

Submission Instructions: Create a compressed file (.zip or .tar.gz files are accepted) with all your source files such as .py files. Within this compressed .zip folder, you should provide some screenshots of your running program's output as proof. Generally speaking, to complete the number placement puzzle, you just need one .py file. But it's better to submit everything as a compressed file. Submit the compressed file to Canvas.

Late submission policy: As described in the syllabus, any late submission will the penalized with 10% off after each 24 hours late. For example, an assignment worth 100 points turned in 2 days late will receive a 20-point penalty. Assignments turned in 5 or more days after the due date will receive a grade of 0.