Course Title:

Design & Analysis of Algorithms

Course Code: CSE533

Assignment Title:

CIA 3.1 - Algorithm Game Design

Sort Quest!

Submitted by:

Joel Tito (2362515)

Date of Submission:

6th September 2025

Introduction

The DAA Sorting Game is an interactive web application that can be used to know fundamental sorting algorithms through gameplay. Developed using Django, the game setup - where players sort arrays by following the specific rules of classic algorithms: Bubble Sort, Selection Sort, and Insertion Sort.

The game helps to bridge the gap between theoretical understanding and practical application of sorting algorithms. By enforcing algorithm-specific move rules and providing instant feedback, users develop a deeper intuition for how each sorting technique operates. The game features multiple difficulty levels, a scoring system, hints, and the ability to save and resume progress.

In addition to gameplay, the application includes a dedicated learning section with detailed explanations, pseudocode, step-by-step examples, and visual animations for each algorithm. This holistic approach ensures that users not only play but also understand the underlying logic and efficiency of each sorting method.

Algorithm Working

Bubble Sort

Bubble Sort is a simple comparison-based algorithm that repeatedly steps through the array, compares adjacent elements, and swaps them if they are in the wrong order. With each pass, the largest unsorted element "bubbles up" to its correct position at the end of the array.

Game Implementation:

Players can only swap adjacent elements.

A swap is valid only if the left element is greater than the right element.

The game enforces repeated passes until the array is sorted.

Initial: [5, 3, 8, 4, 2]

Pass 1: [3, 5, 4, 2, 8]

Pass 2: [3, 4, 2, 5, 8]

Pass 3: [3, 2, 4, 5, 8]

Pass 4: [2, 3, 4, 5, 8] (Sorted)

Pseudocode:

```
for i from 0 to n-1:
    for j from 0 to n-i-2:
        if arr[j] > arr[j+1]:
            swap arr[j], arr[j+1]
```

Validation Logic Used:

```
if algorithm == 'bubble':
    return abs(move_from - move_to) == 1 and arr[move_from] > arr[move_to]
```

Selection Sort

Selection Sort divides the array into a sorted and unsorted part. It repeatedly selects the smallest element from the unsorted portion and swaps it with the first unsorted element, gradually expanding the sorted section from left to right.

Game Implementation:

Players must select the first unsorted element and the minimum element in the unsorted portion.

A move is valid only if the selected elements match these criteria.

After each valid swap, the sorted portion grows by one.

Initial: [5, 3, 8, 4, 2]

Step 1: Find min in [5, 3, 8, 4, 2] → 2, swap with 5 → [2, 3, 8, 4, 5]

Step 2: Find min in [3, 8, 4, 5] → 3, swap with 3 → [2, 3, 8, 4, 5]

Step 3: Find min in [8, 4, 5] → 4, swap with 8 → [2, 3, 4, 8, 5]

Step 4: Find min in [8, 5] → 5, swap with 8 → [2, 3, 4, 5, 8]

Step 5: [2, 3, 4, 5, 8] (Sorted)

Pseudocode:

```
for i from 0 to n-1:
    min_idx = i
    for j from i+1 to n-1:
        if arr[j] < arr[min_idx]:
            min_idx = j
    swap arr[i], arr[min_idx]
```

Validation Logic Used:

```
elif algorithm == 'selection':
    if step >= len(arr):
        return False
    if move_from != step:
        return False
    min_idx = step
    for i in range(step + 1, len(arr)):
        if arr[i] < arr[min_idx]:
            min_idx = i
    return move_to == min_idx and move_from != move_to
```

Insertion Sort

Insertion Sort builds the sorted array one element at a time. It takes each new element and inserts it into its correct position within the already sorted portion, shifting elements as needed.

Game Implementation:

Players select the next unsorted element and its correct position in the sorted part.

The game validates the move and shifts elements accordingly.

The sorted portion expands with each valid insertion.

Initial: [5, 3, 8, 4, 2]

Step 1: Insert 3 before 5 → [3, 5, 8, 4, 2]

Step 2: 8 stays → [3, 5, 8, 4, 2]

Step 3: Insert 4 before 8 → [3, 5, 4, 8, 2] → [3, 4, 5, 8, 2]

Step 4: Insert 2 before 8 → [3, 4, 5, 2, 8] → [3, 4, 2, 5, 8] → [3, 2, 4, 5, 8] → [2, 3, 4, 5, 8] (Sorted)

Pseudocode:

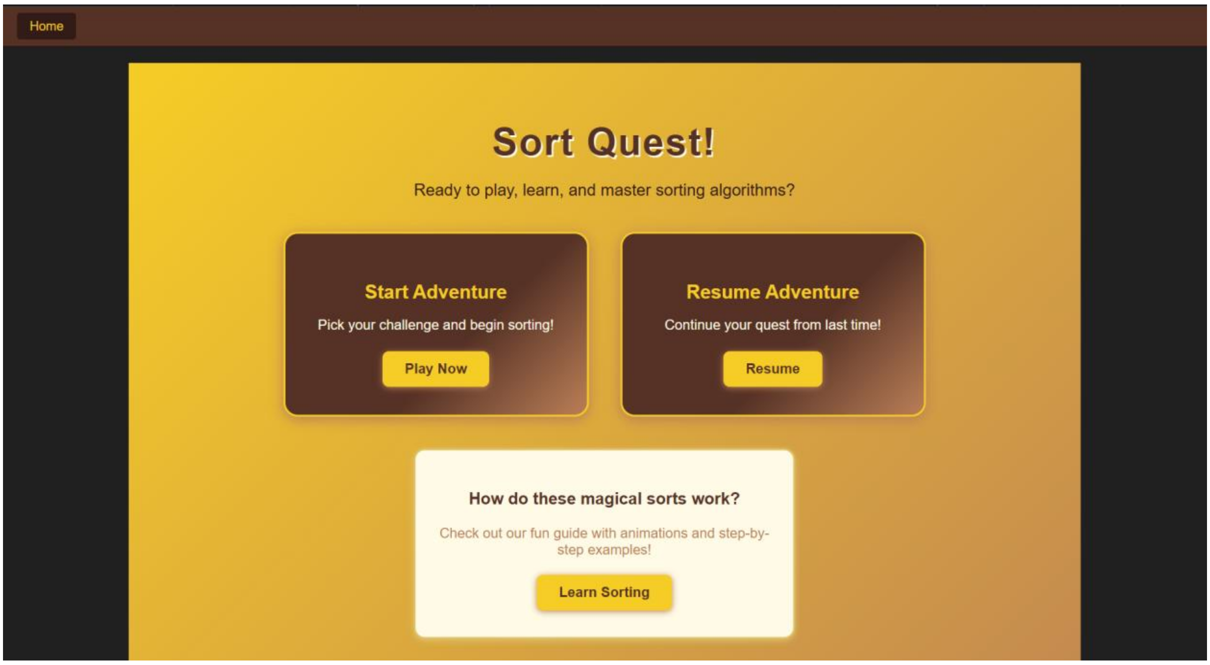
```
for i from 1 to n-1:
    key = arr[i]
    j = i-1
    while j >= 0 and arr[j] > key:
        arr[j+1] = arr[j]
        j = j-1
    arr[j+1] = key
```

Validation Logic Used:

```
elif algorithm == 'insertion':
    if step >= len(arr) - 1:
        return False
    if move_from != step + 1:
        return False
    element = arr[move_from]
    correct_pos = 0
    for i in range(step + 1):
        if arr[i] <= element:
            correct_pos = i + 1
    return move_to == correct_pos
```

Screenshots:

Home Page:



Sort Quest!

Ready to play, learn, and master sorting algorithms?

Start Adventure

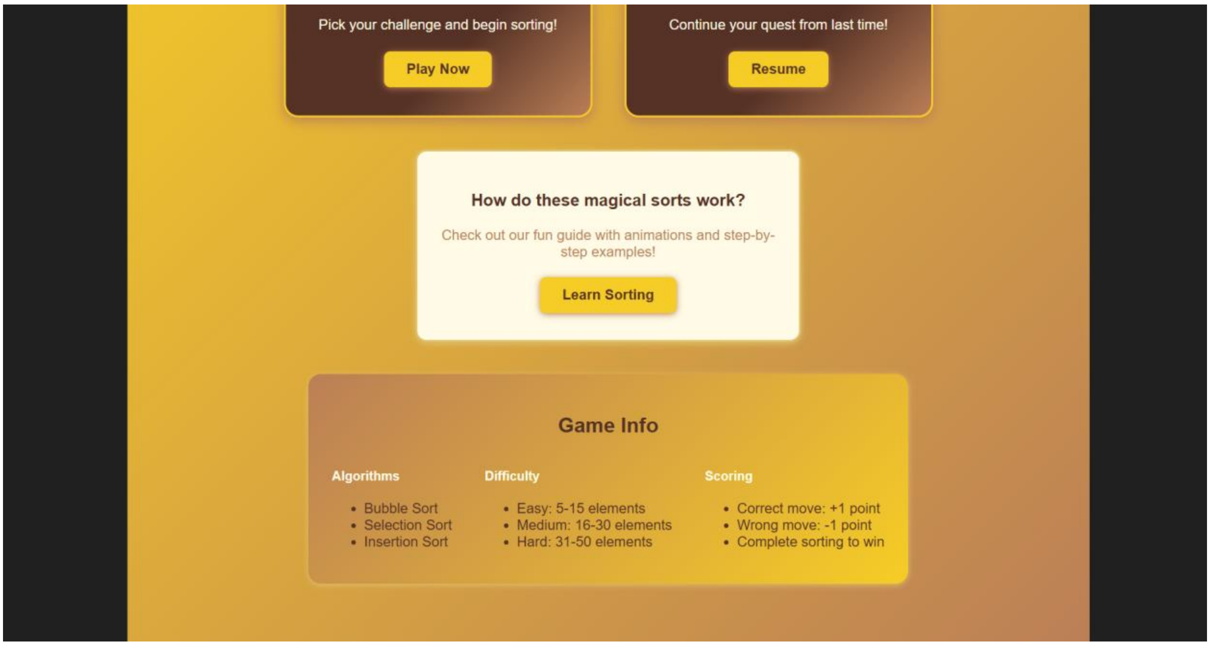
Pick your challenge and begin sorting!

Resume Adventure

Continue your quest from last time!

How do these magical sorts work?

Check out our fun guide with animations and step-by-step examples!


Pick your challenge and begin sorting!

Continue your quest from last time!

How do these magical sorts work?

Check out our fun guide with animations and step-by-step examples!

Game Info

Algorithms

- Bubble Sort
- Selection Sort
- Insertion Sort

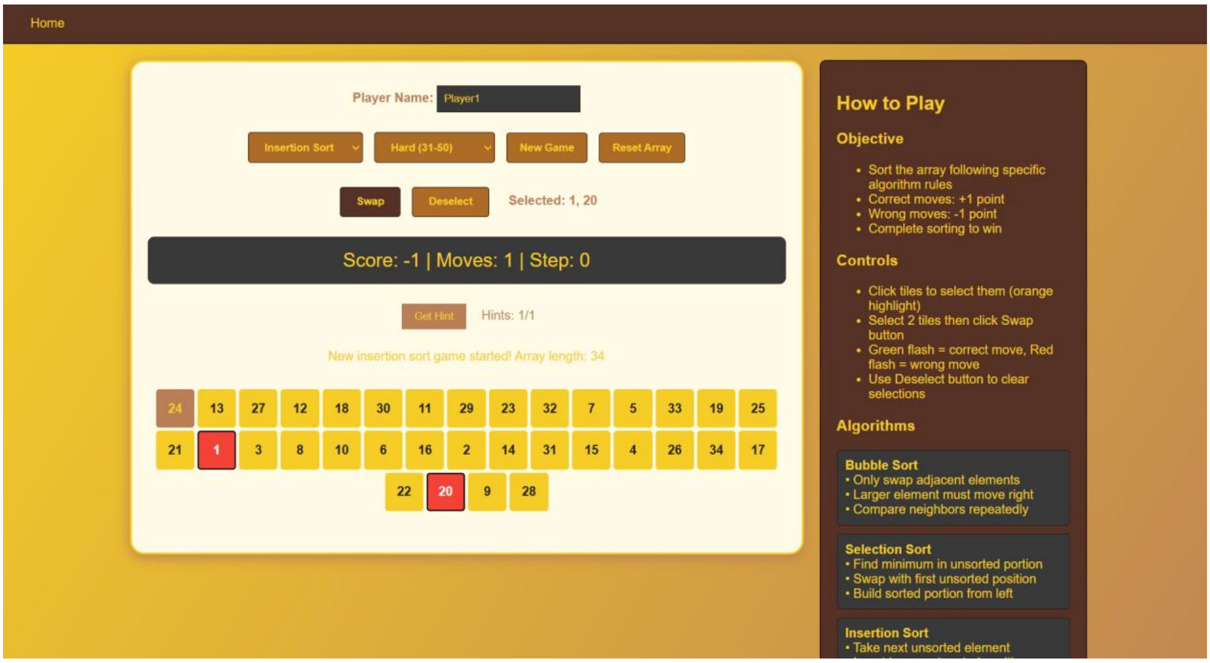
Difficulty

- Easy: 5-15 elements
- Medium: 16-30 elements
- Hard: 31-50 elements

Scoring

- Correct move: +1 point
- Wrong move: -1 point
- Complete sorting to win

New Game Page:



Player Name: Player1

Insertion Sort | Hard (31-50)

Selected: 1, 20

Score: -1 | Moves: 1 | Step: 0

Hints: 1/1

New insertion sort game started! Array length: 34

24 13 27 12 18 30 11 29 23 32 7 5 33 19 25
21 1 3 8 10 6 16 2 14 31 15 4 26 34 17
22 20 9 28

How to Play

Objective

- Sort the array following specific algorithm rules
- Correct moves: +1 point
- Wrong moves: -1 point
- Complete sorting to win

Controls

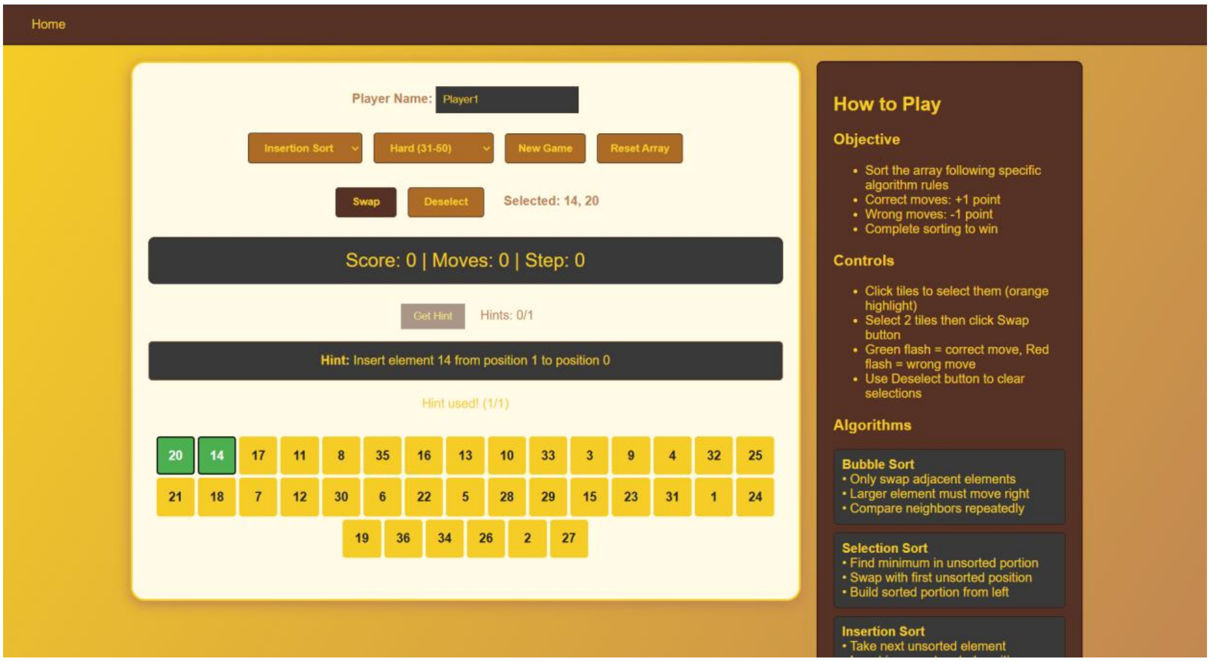
- Click tiles to select them (orange highlight)
- Select 2 tiles then click Swap button
- Green flash = correct move, Red flash = wrong move
- Use Deselect button to clear selections

Algorithms

Bubble Sort
• Only swap adjacent elements
• Larger element must move right
• Compare neighbors repeatedly

Selection Sort
• Find minimum in unsorted portion
• Swap with first unsorted position
• Build sorted portion from left

Insertion Sort
• Take next unsorted element



Player Name: Player1

Insertion Sort | Hard (31-50)

Selected: 14, 20

Score: 0 | Moves: 0 | Step: 0

Hints: 0/1

Hint: Insert element 14 from position 1 to position 0

Hint used! (1/1)

20 14 17 11 8 35 16 13 10 33 3 9 4 32 25
21 18 7 12 30 6 22 5 28 29 15 23 31 1 24
19 36 34 26 2 27

How to Play

Objective

- Sort the array following specific algorithm rules
- Correct moves: +1 point
- Wrong moves: -1 point
- Complete sorting to win

Controls

- Click tiles to select them (orange highlight)
- Select 2 tiles then click Swap button
- Green flash = correct move, Red flash = wrong move
- Use Deselect button to clear selections

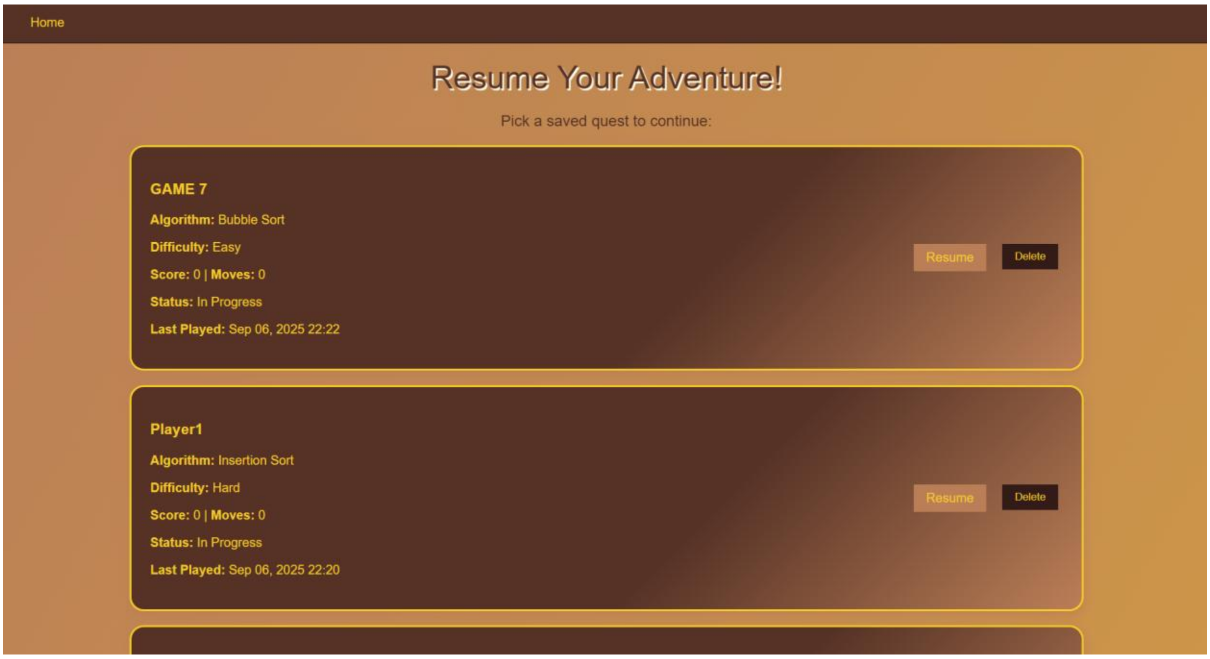
Algorithms

Bubble Sort
• Only swap adjacent elements
• Larger element must move right
• Compare neighbors repeatedly

Selection Sort
• Find minimum in unsorted portion
• Swap with first unsorted position
• Build sorted portion from left

Insertion Sort
• Take next unsorted element

Resume Page:



Resume Your Adventure!

Pick a saved quest to continue:

GAME 7

Algorithm: Bubble Sort

Difficulty: Easy

Score: 0 | **Moves:** 0

Status: In Progress

Last Played: Sep 06, 2025 22:22

Player1

Algorithm: Insertion Sort

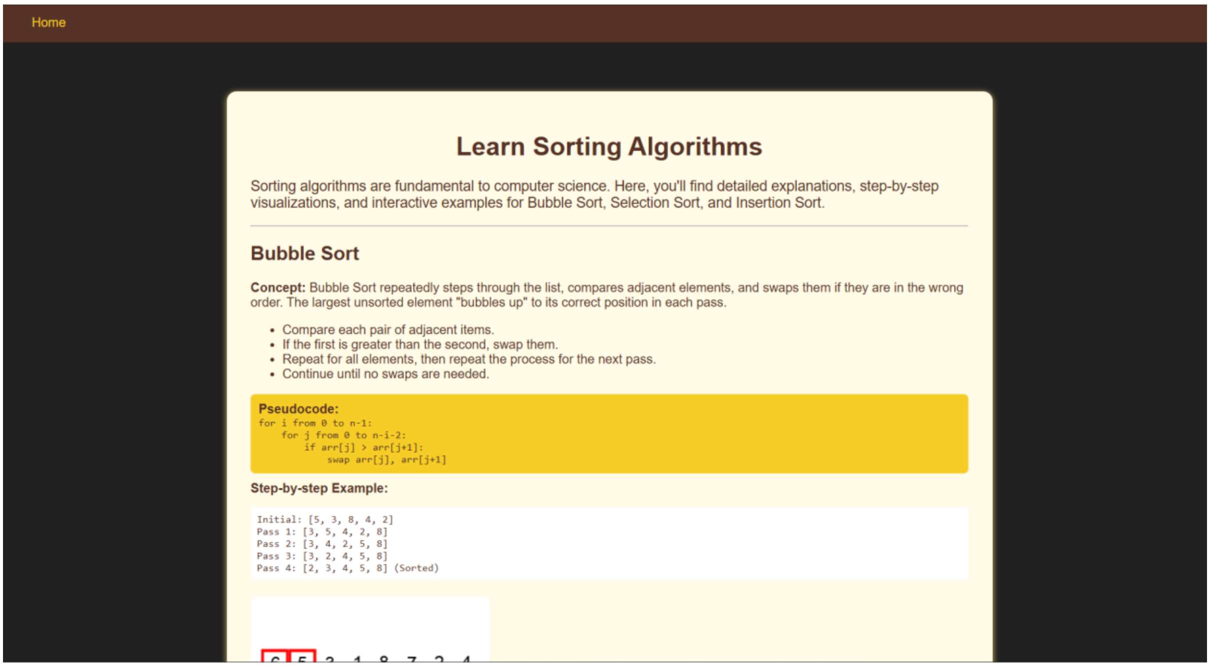
Difficulty: Hard

Score: 0 | **Moves:** 0

Status: In Progress

Last Played: Sep 06, 2025 22:20

Learning Page:



Learn Sorting Algorithms

Sorting algorithms are fundamental to computer science. Here, you'll find detailed explanations, step-by-step visualizations, and interactive examples for Bubble Sort, Selection Sort, and Insertion Sort.

Bubble Sort

Concept: Bubble Sort repeatedly steps through the list, compares adjacent elements, and swaps them if they are in the wrong order. The largest unsorted element "bubbles up" to its correct position in each pass.

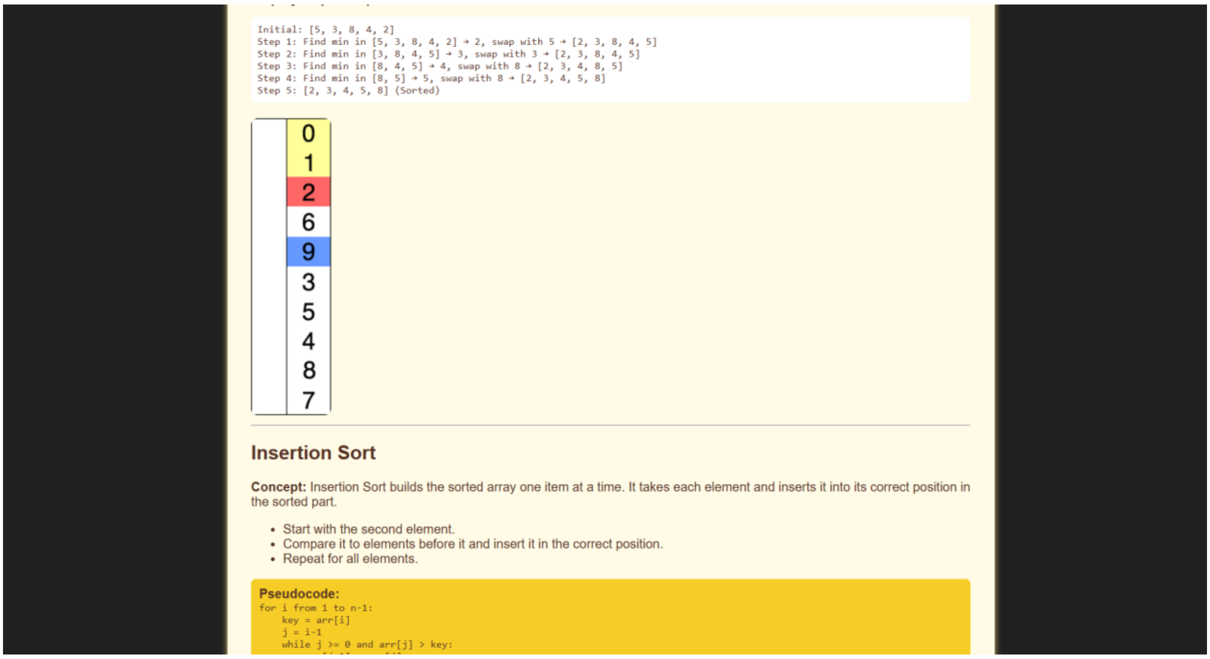
- Compare each pair of adjacent items.
- If the first is greater than the second, swap them.
- Repeat for all elements, then repeat the process for the next pass.
- Continue until no swaps are needed.

Pseudocode:

```
for i from 0 to n-1:
    for j from 0 to n-i-2:
        if arr[j] > arr[j+1]:
            swap arr[j], arr[j+1]
```

Step-by-step Example:

```
Initial: [5, 3, 8, 4, 2]
Pass 1: [3, 5, 4, 2, 8]
Pass 2: [3, 4, 2, 5, 8]
Pass 3: [3, 2, 4, 5, 8]
Pass 4: [2, 3, 4, 5, 8] (Sorted)
```



```
Initial: [5, 3, 8, 4, 2]
Step 1: Find min in [5, 3, 8, 4, 2] → 2, swap with 5 → [2, 3, 8, 4, 5]
Step 2: Find min in [3, 8, 4, 5] → 3, swap with 3 → [2, 3, 8, 4, 5]
Step 3: Find min in [8, 4, 5] → 4, swap with 8 → [2, 3, 4, 8, 5]
Step 4: Find min in [8, 5] → 5, swap with 8 → [2, 3, 4, 5, 8]
Step 5: [2, 3, 4, 5, 8] (Sorted)
```

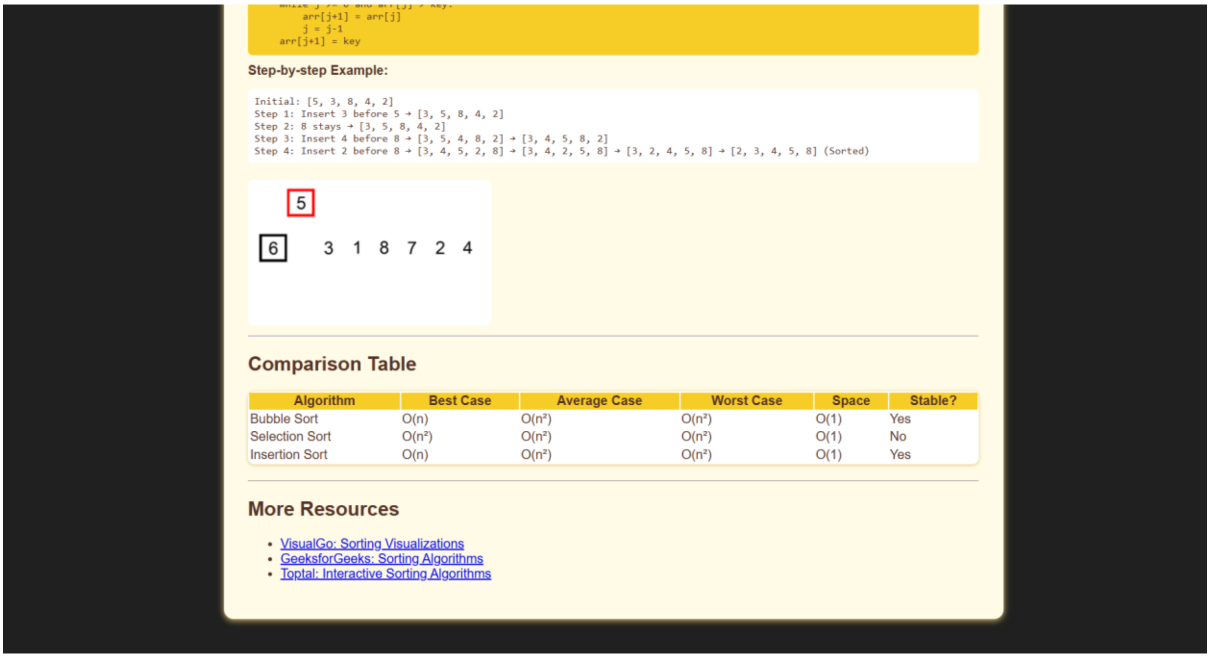
Insertion Sort

Concept: Insertion Sort builds the sorted array one item at a time. It takes each element and inserts it into its correct position in the sorted part.

- Start with the second element.
- Compare it to elements before it and insert it in the correct position.
- Repeat for all elements.

Pseudocode:

```
for i from 1 to n-1:
    key = arr[i]
    j = i-1
    while j >= 0 and arr[j] > key:
```



```
        arr[j+1] = arr[j]
        j = j-1
    arr[j+1] = key
```

Step-by-step Example:

```
Initial: [5, 3, 8, 4, 2]
Step 1: Insert 3 before 5 → [3, 5, 8, 4, 2]
Step 2: 8 stays → [3, 5, 8, 4, 2]
Step 3: Insert 4 before 8 → [3, 5, 4, 8, 2] → [3, 4, 5, 8, 2]
Step 4: Insert 2 before 8 → [3, 4, 5, 2, 8] → [3, 4, 2, 5, 8] → [3, 2, 4, 5, 8] → [2, 3, 4, 5, 8] (Sorted)
```

Comparison Table

Algorithm	Best Case	Average Case	Worst Case	Space	Stable?
Bubble Sort	O(n)	O(n²)	O(n²)	O(1)	Yes
Selection Sort	O(n²)	O(n²)	O(n²)	O(1)	No
Insertion Sort	O(n)	O(n²)	O(n²)	O(1)	Yes

More Resources

- VisualGo: Sorting Visualizations
- GeeksforGeeks: Sorting Algorithms
- Toptal: Interactive Sorting Algorithms

More Detailed Implementation and details is provided at:

https://github.com/JO-Techs/DAA_Game