Explanation of Sorting Algorithm Visualizer in Streamlit

Varun Kumar

July 5, 2025

1. Importing Libraries

```
import streamlit as st
import time
import plotly.express as px
import heapq
```

Explanation:

- `streamlit` is used to create the web interface.
- `time` module helps measure performance.
- `plotly.express` is used for bar chart plotting.
- `heapq` is used for implementing heap sort.

2. Sorting Algorithm Implementations

Each sorting function returns a sorted copy of the input array.

Bubble Sort

```
def bubble_sort(arr):
    a = arr.copy()
    n = len(a)
    for i in range(n):
        for j in range(0, n - i - 1):
            if a[j] > a[j + 1]:
                a[j], a[j + 1] = a[j + 1], a[j]
    return a
```

Explanation: Compares adjacent elements and swaps if needed; slow $O(n^2)$.

Insertion Sort

```
def insertion_sort(arr):
    a = arr.copy()
    for i in range(1, len(a)):
        key = a[i]
        j = i - 1
        while j >= 0 and key < a[j]:
            a[j + 1] = a[j]
            j -= 1
        a[j + 1] = key
    return a
```

Explanation: Builds sorted array by inserting elements one at a time.

Selection Sort

```
def selection_sort(arr):
    a = arr.copy()
    for i in range(len(a)):
        min_idx = i
        for j in range(i + 1, len(a)):
            if a[j] < a[min_idx]:
                min_idx = j
        a[i], a[min_idx] = a[min_idx], a[i]
    return a
```

Explanation: Finds the minimum and places it at correct position iteratively.

Merge Sort

```
def merge_sort(arr):
    if len(arr) <= 1:
        return arr
    mid = len(arr) // 2
    L = merge_sort(arr[:mid])
    R = merge_sort(arr[mid:])
    return merge(L, R)

def merge(left, right):
    result = []
    i = j = 0
    while i < len(left) and j < len(right):
        if left[i] < right[j]:
            result.append(left[i])
            i += 1
        else:
            result.append(right[j])
            j += 1
    result.extend(left[i:])
    result.extend(right[j:])
    return result
```

Explanation: A divide-and-conquer algorithm that splits and merges sorted halves. Time complexity: $O(n \log n)$.

Quick Sort

```
def quick_sort(arr):
    if len(arr) <= 1:
        return arr
    pivot = arr[0]
    left = [x for x in arr[1:] if x < pivot]
    mid = [x for x in arr if x == pivot]
    right = [x for x in arr[1:] if x >= pivot]
    return quick_sort(left) + mid + quick_sort(right)
```

Explanation: Uses pivot element and partitions data recursively.

Heap Sort

```
def heap_sort(arr):
    h = []
    for value in arr:
        heapq.heappush(h, value)
    return [heapq.heappop(h) for _ in range(len(h))]
```

Explanation: Uses a min-heap to extract elements in sorted order.

3. Mapping Sorting Names to Functions

```
SORT_FUNCTIONS = {
    "Bubble Sort": bubble_sort,
    "Insertion Sort": insertion_sort,
    "Selection Sort": selection_sort,
    "Merge Sort": merge_sort,
    "Quick Sort": quick_sort,
    "Heap Sort": heap_sort,
}
```

Explanation: Dictionary to link names (dropdown) to actual sorting functions.

4. Streamlit UI

```
st.title("Sorting Algorithm Visualizer")
input_str = st.text_input("Enter array (comma-separated)", "5, 3, 8, 4, 2"
    )
algo = st.selectbox("Choose sorting algorithm", list(SORT_FUNCTIONS.keys()
    ) + ["All (Test Performance)"])
```

Explanation:

- Sets page title.
- Text input for the user to enter a list.
- Dropdown (selectbox) to choose an algorithm or benchmark all.

5. When `Sort` Button is Clicked

```
if st.button("Sort"):
    try:
        input_array = list(map(int, input_str.split(',')))
```

Explanation: On button click, input string is split and converted to a list of integers.

6. If "All (Test Performance)" is Selected

```
if algo == "All (Test Performance)":
    st.subheader("Performance Comparison")
    results = []
    for name, func in SORT_FUNCTIONS.items():
        start = time.perf_counter()
        _ = func(input_array)
        duration = time.perf_counter() - start
        results.append({"Algorithm": name, "Time (s)": duration})
```

Explanation:

- Loops over all sorting algorithms.
- Records execution time using `time.perf_counter()`.

Plotting Performance

```
    fig = px.bar(
        results,
        x="Algorithm",
        y="Time (s)",
        hover_data={"Time (s)": ":.6f"},
        text="Time (s)",
        title="Sorting Algorithm Time Comparison",
    )
    fig.update_traces(texttemplate="%{text:.6f}", textposition="outside")
    st.plotly_chart(fig, use_container_width=True)
```

Explanation: Uses Plotly to render a bar chart showing performance of each algorithm.

7. If Specific Algorithm is Selected

```
else:
    sort_func = SORT_FUNCTIONS[algo]
    start = time.perf_counter()
    sorted_array = sort_func(input_array)
    duration = time.perf_counter() - start

    st.success(f"Sorted Array: {sorted_array}")
    st.info(f"Time Taken: {duration:.6f} seconds")
```

Explanation:

- Applies the selected sorting function.
- Displays sorted array and time taken.

8. Error Handling

```
except Exception as e:
    st.error(f"Error: {e}")
```

Explanation: Catches errors like invalid input and shows an error message to the user.