Algorithms and Data Structures 2

Sorting Algorithms

Robert Horvick

www.pluralsight.com

Outline

- **Overview of Sorting**
  - Measuring Performance
- **Sorting Algorithms**
  - Bubble Sort
  - Insertion Sort
  - Merge Sort
  - Selection Sort
  - Quick Sort
- **Visualizing Sorting Performance**
  - Sample Source Code and Demo

Sorting Overview

- **Arranging data in a collection based on a comparison algorithm**
  - E.g., Any object with a notion of greater-than/less-than/equality
- **Two general families of sorting algorithms**
  - Linear Sorting
  - Divide and Conquer
- **Linear algorithms treat the problem of sorting as a single large operation.**
- **Divide and Conquer algorithms partition the data to be sorted into smaller sets that can be independently sorted.**

Measuring Performance

- **Comparisons**
  - When two values of the input array are compared for relative equality
    - Equal to, Greater then, Less then
- **Swaps**
  - When two values stored in the input array are swapped
  - E.g., [1, 0] => [0, 1]
- **Performance Considerations**
  - Comparisons and Swaps both have a cost
  - Reducing either or both can improve performance
  - The cost of both operations depends on many factors.

Bubble Sort

- **Simplest sorting algorithm**
- **On Each Pass**
  - Compare each array item to it's right neighbor
  - If the right neighbor is smaller then Swap right and left
  - Repeat for the remaining array items
- **Perform subsequent passes until no swaps are performed**

Pass 1

Pass 2

Pass 3

Bubble Sort Performance

- **Worst case performance: $O(n^2)$**
  - Not appropriate for large unsorted data sets.
- **Average case performance: $O(n^2)$**
  - Not appropriate for large unsorted data sets.
- **Best case performance: O(n)**
  - Very good best case performance and can efficiently sort small and nearly sorted data sets
- **Space required: O(n)**
  - Bubble sort operates directly on the input array meaning it is a candidate algorithm when minimizing space is paramount.

Insertion Sort

- **Sorts each item in the array as they are encountered**
- **As the current item works from left to right**
  - Everything left of the item is known to be sorted
  - Everything to the right is unsorted
- **The current item is "inserted" into place within the sorted section**

Insertion Sort Performance

- **Worst case performance: $O(n^2)$**
  - Not appropriate for large unsorted data sets.
- **Average case performance: $O(n^2)$**
  - Not appropriate for large unsorted data sets.
- **Best case performance: O(n)**
  - Very good best case performance and can efficiently sort small and nearly sorted data sets
- **Space required: O(n)**
  - Insertion sort operates directly on the input array meaning it is a candidate algorithm when minimizing space is paramount.

Merge Sort

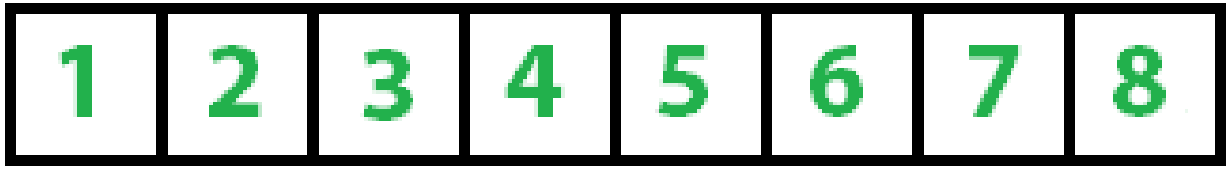
- The array is recursively split in half recursively
- When the array is in groups of 1, it is reconstructed in sort order
- Each reconstructed array is merged with the other half

Merge Sort Performance

- **Worst case performance: O(n log n)**
  - Appropriate for large data sets
  - Data splitting means that the algorithm can be made parallel.
- **Average case performance: O(n log n)**
  - Appropriate for large data sets
- **Best case performance: O(n log n)**
  - Appropriate for large data sets
- **Space required: O(n)**
  - Merge can be, but is often not, performed in-place. These extra allocations increase the memory footprint required to sort data.

Selection Sort

- **Sorts the data by finding the smallest item and swapping it into the array in the first unsorted location.**
- **Algorithm:**
  - Enumerate the array from the first unsorted item to the end
  - Identify the smallest item
  - Swap the smallest item with the first unsorted item

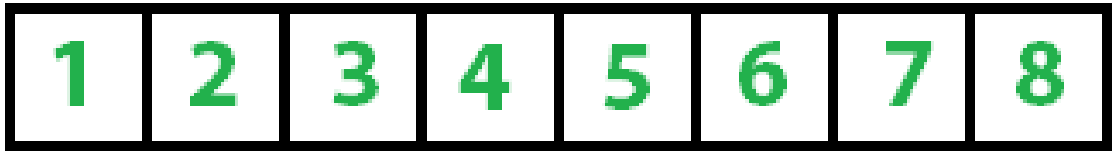


Selection Sort Performance

- **Worst case performance:$O(n^2)$**
  - Not appropriate for large unsorted data sets.
- **Average case performance:$O(n^2)$**
  - Not appropriate for large unsorted data sets.
  - Typically performs better than bubble but worse than insertion sort
- **Best case performance:$O(n^2)$**
  - Not appropriate for large unsorted data sets.
- **Space required: O(n)**
  - Selection sort operates directly on the input array meaning it is a candidate algorithm when minimizing space is paramount.

Quick Sort

- **Divide and Conquer algorithm**
- **Pick a pivot value and partition the array**
- **Put all values before the pivot to the left and above to the right**
  - The pivot point is now sorted – everything right is larger, everything left is smaller.
- **Perform pivot and partition algorithm on the left and right partitions**
- **Repeat until sorted**



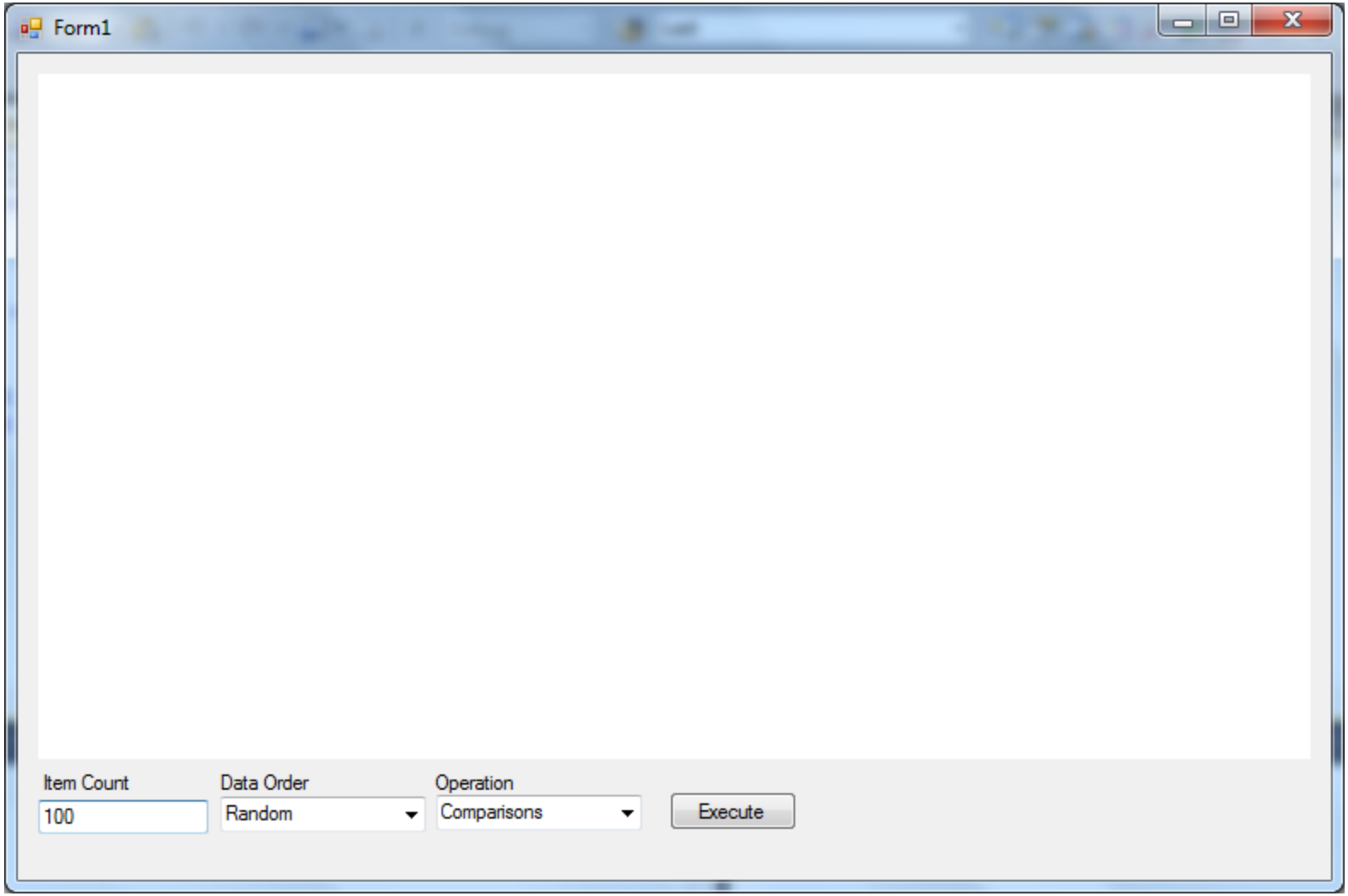
Quick Sort Performance

- **Worst case performance:$O(n^2)$**
  - Not appropriate for large pathologically sorted (inverse sorted) data sets.
- **Average case performance:O(n log n)**
  - Appropriate for large data sets
- **Best case performance:O(n log n)**
  - Very good best case performance and can efficiently sort small and nearly sorted data sets
- **Space required: O(n)**
  - As a recursive algorithm the array space as well as the stack space must be considered. There exist optimizations to reduce space usage further.

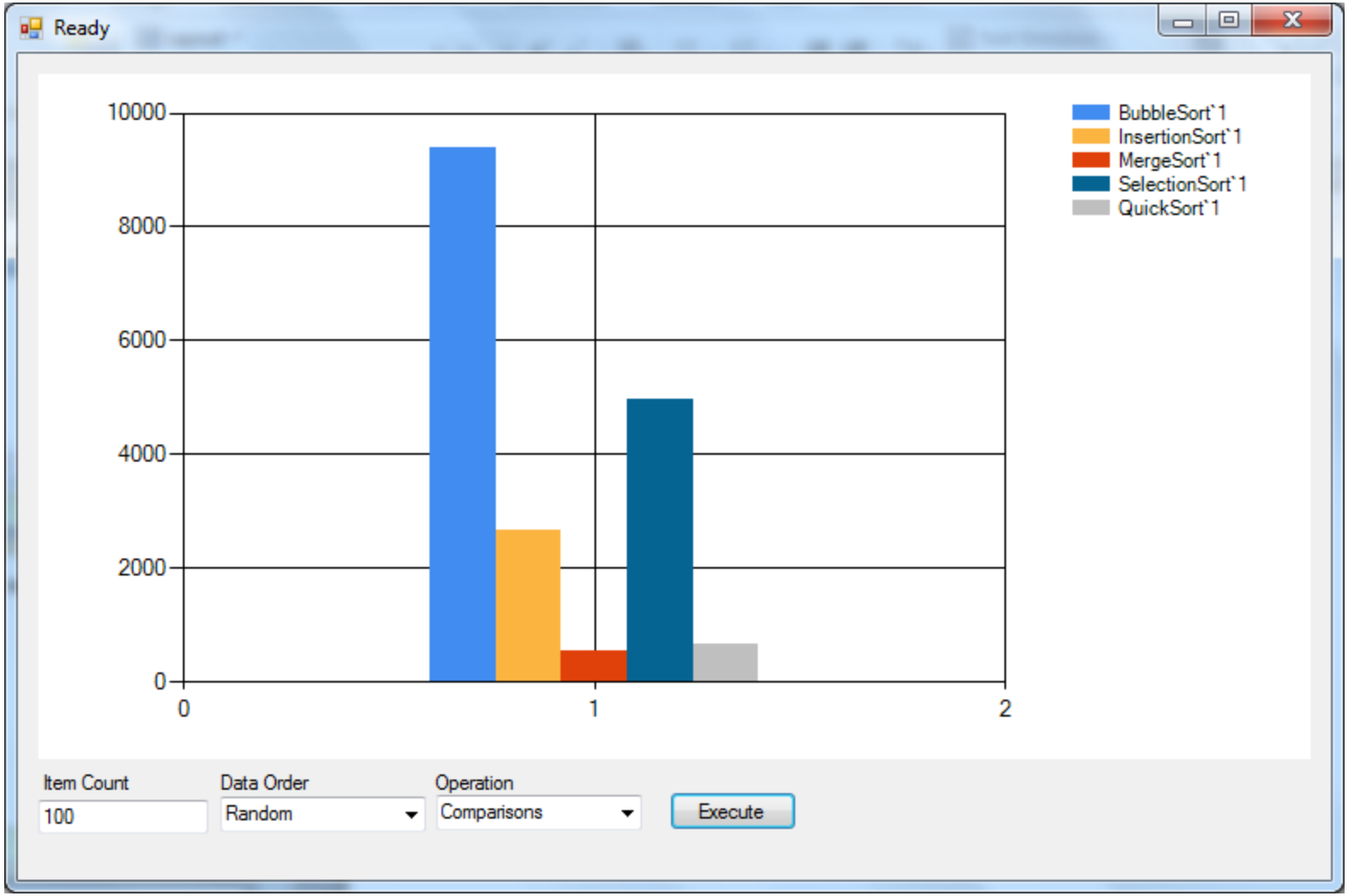
Visualizing Sorting Performance

- **Graphically comparing the cost of sorting various data sets**
- **Variable Item Counts**
  - Ability to set the item count to any range from 0 to billions.
- **Variable data sorting**
  - Pre-Sorted
  - Reversed Sorted
  - Random Data
- **Measurement type**
  - Comparisons
  - Swaps (or merge sort assignments)

Visualizing Sorting Performance



Visualizing Sorting Performance



Summary

- **Overview of Sorting**
  - Measuring Performance
- **Sorting Algorithms**
  - Bubble Sort
  - Insertion Sort
  - Selection Sort
  - Merge Sort
  - Quick Sort
- **Visualized Sorting Performance**
  - Sample Source Code and Demo

References

- **Wikipedia.org Sorting Articles**
  - http://en.wikipedia.org/wiki/Bubble_sort
  - http://en.wikipedia.org/wiki/Insertion_sort
  - http://en.wikipedia.org/wiki/Merge_sort
  - http://en.wikipedia.org/wiki/Selection_sort
  - http://en.wikipedia.org/wiki/Quicksort
- **MSDN Charting API (Sample Application)**
  - System.Windows.Forms.DataVisualization.Charting.Chart
  - http://msdn.microsoft.com/en-us/library/dd489065.aspx