

SORT

Sorting algorithms take a group of elements and return them in a specific order.

I. Language Agnostic Overview

Sorting is a problem that is well suited to computers. Sorting steps can be laid out explicitly, and the task is highly repetitive. A significant number of sorting algorithms have been developed, and there are substantial differences between them. The majority of sorting algorithms are comparison based, which limits their fastest worst-case sorting time to $O(n \cdot \log(n))$ where n is the number of elements to be sorted.

All the sorting algorithms presented here were taken from the data structures textbook “Data Structures and Algorithm Analysis in Java” by Mark Allen Weiss.

Ia. Algorithm Overview

- **Insertion Sort** – Perhaps the simplest of all sorting algorithms, insertion sort starts with the first element, and then compares the second to it. Once those two are in order, the algorithm continues, finding the correct place for every element.
 - Algorithms in Java, 3rd Edition, Robert Sedgewick, pgs 285-287
- **Shell Sort** – This sorting method sorts elements (using insertion sort) separated by k elements – a k sized group is referred to as a gap – and then decreases k and repeats the process until $k=1$, after which the elements are sorted.
 - Algorithms in Java, 3rd Edition, Robert Sedgewick, pgs 300-306
- **Heap Sort** – Once the elements are in a heap data structure, the top element is removed. This is either the largest or smallest element (depending on whether a min-heap or max-heap is used) and the heap structure is preserved – so the process can be repeated.
 - Algorithms in Java, 3rd Edition, Robert Sedgewick, pgs 389-393
- **Merge Sort** – The elements are split into halves recursively until they are single. Then, they are merged with each other, and when they merge, they are sorted. This produces sorted arrays, the number of which divides in half and size multiplies by two with every iteration.
 - Algorithms in Java, 3rd Edition, Robert Sedgewick, pgs 353-356
- **Quick Sort** – An element is chosen as pivot, and then all elements larger than it go to one side and smaller elements to the other side. A pivot is chosen from each half, and the two sides are split again. This process continues recursively.
 - Algorithms in Java, 3rd Edition, Robert Sedgewick, pgs 315-320

II. Java Implementation

insertionSort

- | |
|--|
| a. This algorithm has two loops, one nested inside the other. The outer loop iterates over every |
|--|

element in the array, and the second iterates from the current element backwards until it reaches the beginning of the array. The inner loop moves every element up one and moves the selected element down until the next element is smaller or equal – this means the element is in the correct spot.

b. The precondition is that the array to be sorted isn't null. The postcondition is that the array is sorted.

shellsort

a. This algorithm has three loops, nested inside one another. The outer loop iterates over the gaps – starting with half of the length of the array, then one fourth of the length of the array, etc.. The middle loop iterates over each gap – two for the first iteration of the outer loop, four for the second, etc.. Finally the inner loop iterates over each element in the given gap, and sorts them using an insertion sort.

b. The precondition is that the array to be sorted isn't null. The postcondition is that the array is sorted.

heapsort

a. Heapsort has two for loops, though they are sequentially ordered rather than nested. The first loop converts the array into a heap, and the second loop sorts the heap. The algorithm uses two helper methods, `percDown` and `swapReferences`. `percDown` moves elements down the heap – this is used in the initial heap construction as well as restructuring after an element is removed for sorting. `swapReferences` simply swaps two elements of an array.

b. The precondition is that the array to be sorted isn't null. The postcondition is that the array is sorted.

mergeSort

a. This algorithm can be thought of as occurring in two phases – splitting and merging. Both are abstracted from the initiating method (below) for clarity. The splitting phase simply divides the given array in half, and then recursively calls itself on both halves. This continues until the arrays contain only one element, this is when the merging begins. Two elements are compared and merged into a two element array – then two of these arrays are merged into a sorted four element array. The algorithm continues like this until one array with all the elements in sorted order is created.

b. The precondition is that the array to be sorted isn't null. The postcondition is that the array is sorted.

quicksort

a. This implementation is again split into several helper methods. In addition to `swapReferences`

(which simply switches two array elements), a method named median3 is called on the first, middle and last values. It finds the median of the three numbers and returns it – this is used to find the pivot. Once a pivot is found, the array is partitioned, split, and sorted again. Once the arrays reach a pre-set cutoff value, they are sorted with insertion sort.

b. The precondition is that the array to be sorted isn't null. The postcondition is that the array is sorted.

Metrics:

Lines of code for implementation: 130

Methods for implementation: 15

Lines of code for invariants: 29

Methods for invariants: 3

References:

Wikipedia's entry on sorting algorithms: http://en.wikipedia.org/wiki/Sorting_algorithm

Algorithms in Java, 3rd Edition by Robert Sedgewick. Chapters 6,7,8,9. Pages 263-417.

- 6.4 “Insertion Sort.” pgs 285-287
- 6.8 “Shell Sort.” pgs 300-306
- 9.4 “Heap Sort.” pgs 389-393
- 8.3 “Top-Down Mergesort.” pgs 353-356
- 7.1 “Quicksort – The basic algorithm.” pgs 315-320