CSCI 2270 - Data Structures and Algorithms

Instructor: Hoenigman / Jacobson
Recitation 9

Objective:

- Learn different sorting algorithm:
    - BubbleSort
    - InsertionSort
    - SelectionSort
    - MergeSort
    - QuickSort

- Compare complexity of algorithms.

Sorting Algorithms

When designing your algorithms, sorting them plays an important part. Basically, Sorting algorithms are algorithms that will arrange the data elements within a list or array in a particular order, usually by a numerical order.

Each sorting algorithm has particular strengths and are usually selected based on the efficiency it provides. When efficiency is discussed in this case, it means the algorithm's efficiency is based on the number of elements it has to sort as the size of the input grows or increases. These algorithms will work by comparing the data that needs to be sorted.

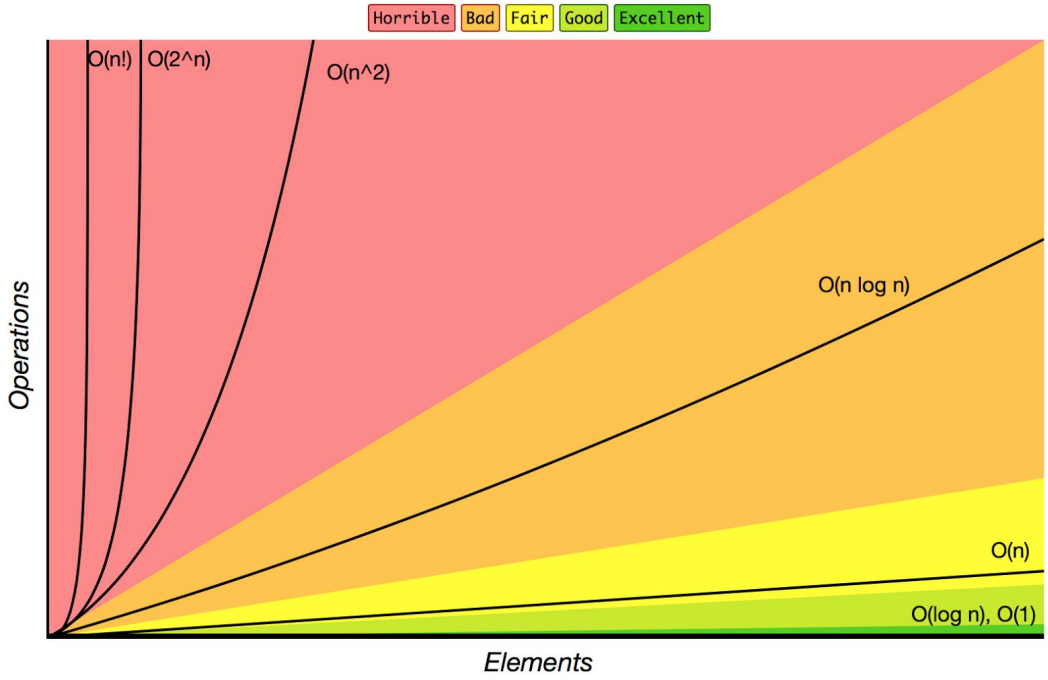
In-Place vs Not-In-Place

Some algorithms will do their comparisons by swapping the elements within the array and do not require any temporary extra space. One example of this type is Bubble sort and is called *in-place sorting.* Other algorithms, require a separate temporary space to be created for the sorting to take place. An example is Merge sort and this is called *not-in-place sorting.* As we continue, this will become more clear through other examples.

Comparisons of Sorting Algorithms[1]

Reference for Big - O Time Complexity Chart:

We have covered Big-O Notation and Time Complexities. This graph can be used for your reference.



Array Sorting Algorithm Comparisons:

These are a few of the time complexities or run times for a few of the sorting algorithms we discussed.

Algorithm	Time Complexity		
	Best	Average	Worst
Quicksort	Ω(n log(n))	θ(n log(n))	O(n^2)
Mergesort	Ω(n log(n))	θ(n log(n))	O(n log(n))
Heapsort	Ω(n log(n))	θ(n log(n))	O(n log(n))
Bubble Sort	Ω(n)	θ(n^2)	O(n^2)
Insertion Sort	Ω(n)	θ(n^2)	O(n^2)
Selection Sort	Ω(n^2)	θ(n^2)	O(n^2)

[1] http://bigocheatsheet.com

A Few Different Sorting Algorithms[2]

Previously we have seen BubbleSort and InsertionSort. We will look further into MergeSort for this recitation. Though there are many others, here are a few of the more common Sorting Algorithms used. Brief explanation of each to follow.

1. BubbleSort
2. InsertionSort
3. SelectionSort
4. MergeSort
5. QuickSort

Bubble Sort Algorithm[3]

Bubble sort is considered a more simple comparison-based sorting algorithm. It works by repeatedly stepping through the array that needs to be sorted and compares each pair of adjacent items. Then, it will swap them if they are not in the correct order. It will start at the front of the array and this process will continue until no further swaps are necessary and thus, the list has been sorted. This algorithm isn't used for larger data sets.

Steps to Bubble Sort Algorithm:

1. Start with comparing the first two elements to check which one is greater.
2. If the left element is greater than the right element, swap
3. Continue comparing each element to the next until the end of the array

Visualization:

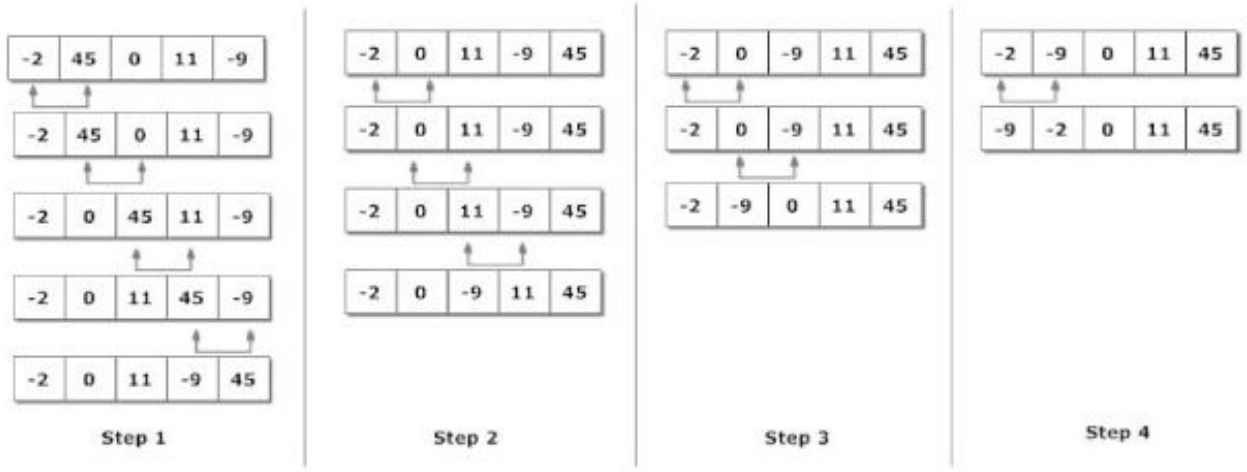


Figure: Working of Bubble sort algorithm

[2] https://www.tutorialspoint.com/data_structures_algorithms/bubble_sort_algorithm.htm
[3] https://www.programiz.com/algorithm/bubble-sort

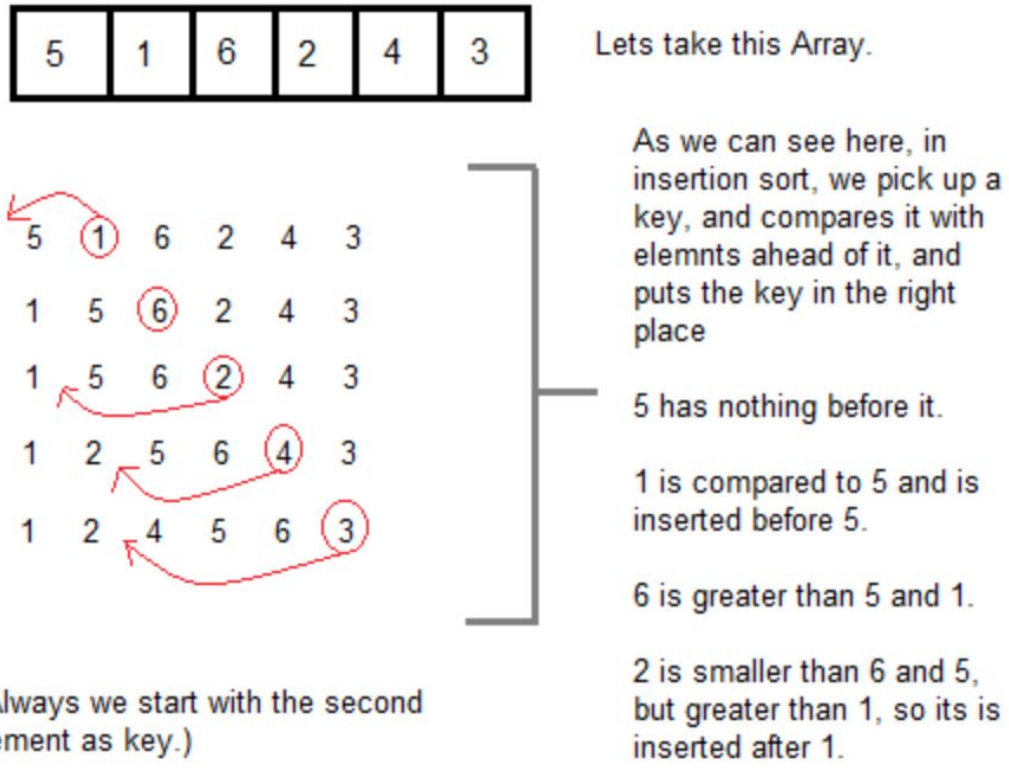
Insertion Sort Algorithm[4]

[5]Insertion sort is another simple sorting algorithm that we have learned. It will build the final sorted array one item at a time starting at the front. It will compare each iteration element and swap when it is out of order. This behavior continues through its sub-list, which is a smaller sorted list of elements at the front of the array.

Steps to Insertion Sort Algorithm:

1. Check if the element is the only list and return
2. Pick the next element and compare it will all the elements in the sorted sublist
3. If the value on the left is greater than the value on the right, then shift all the elements in the sorted sublist
4. Continue comparing each element to the next until the end of the array.

Visualization:



[4]http://www.studytonight.com/data-structures/insertion-sorting
[5] http://interactivepython.org/courselib/static/pythonds/SortSearch/TheInsertionSort.html

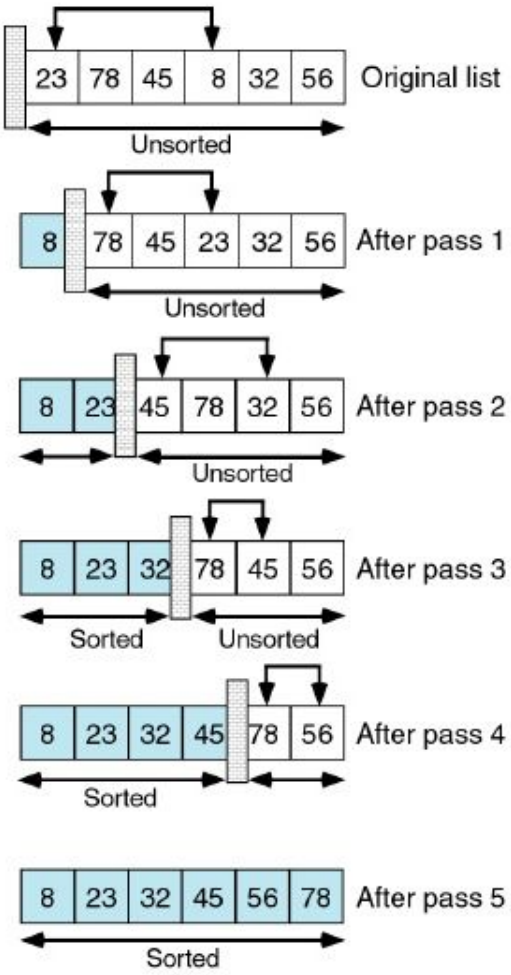
Selection Sort Algorithm[6]

Selection sort is the last of the simple sorting algorithms covered today. It will sort by finding the smallest element in the unsorted array and swap it with the leftmost element of that array. The array is then divided into two conceptual parts. A sorted part (leftmost end) and the unsorted part (everything else). The element that was swapped to the leftmost position becomes the sorted array. This process will continue until the entire array is sorted.

Steps to Selection Sort Algorithm:

1. Set the min location to array[0]
2. Search for the minimum element in the remainder of the array
3. Swap the value at min location with the minimal element in the array
4. Increment min the next element (array[1])
5. Continue until the list is sorted.

Visualization:



[6] https://www.slideshare.net/J.T.A.JONES/sorting-linked-lists

Merge Sort Algorithm

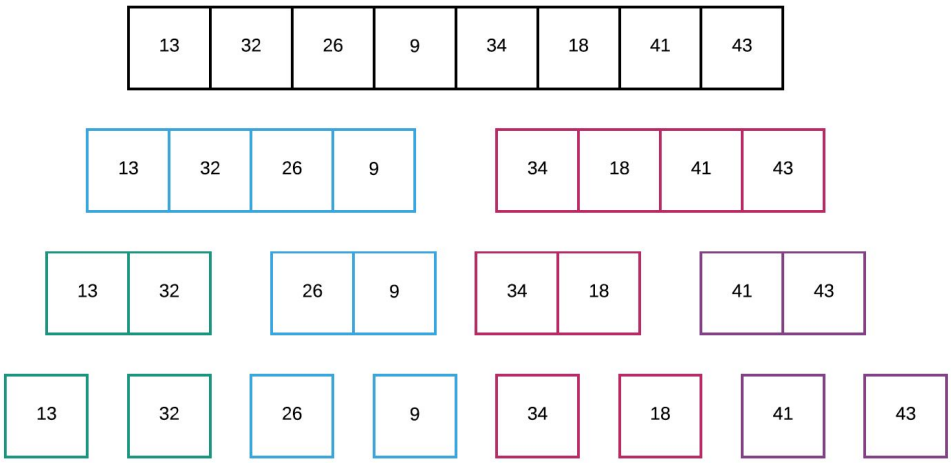
Merge sort is a more advanced sorting algorithm that is based on a divide and conquer method. It will divide the array into equal halves (until the elements can't be divided) and then start to merge them back together, while sorting each of the halves. We will be showing an example of this in class.

Steps to Merge Sort Algorithm:

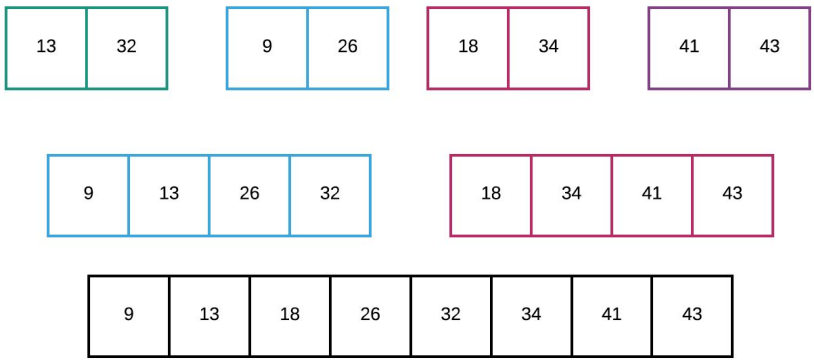
1. If the array only has one element, return as sorted
2. Otherwise, divide the array into two halves.
3. Continue this process until it can't be divided any more times.
4. Merge the smaller lists into new list, which should be sorted.
5. Continue this process until it is completely merged and sorted.

Visualization:

To start, we take an unsorted array and divide the array until we're at single elements.



Now, as they are merged into another array, they are compared and swapped if out of order. As we can see, 13 and 32 remain, but 9 and 26 are swapped when each of these are compared.



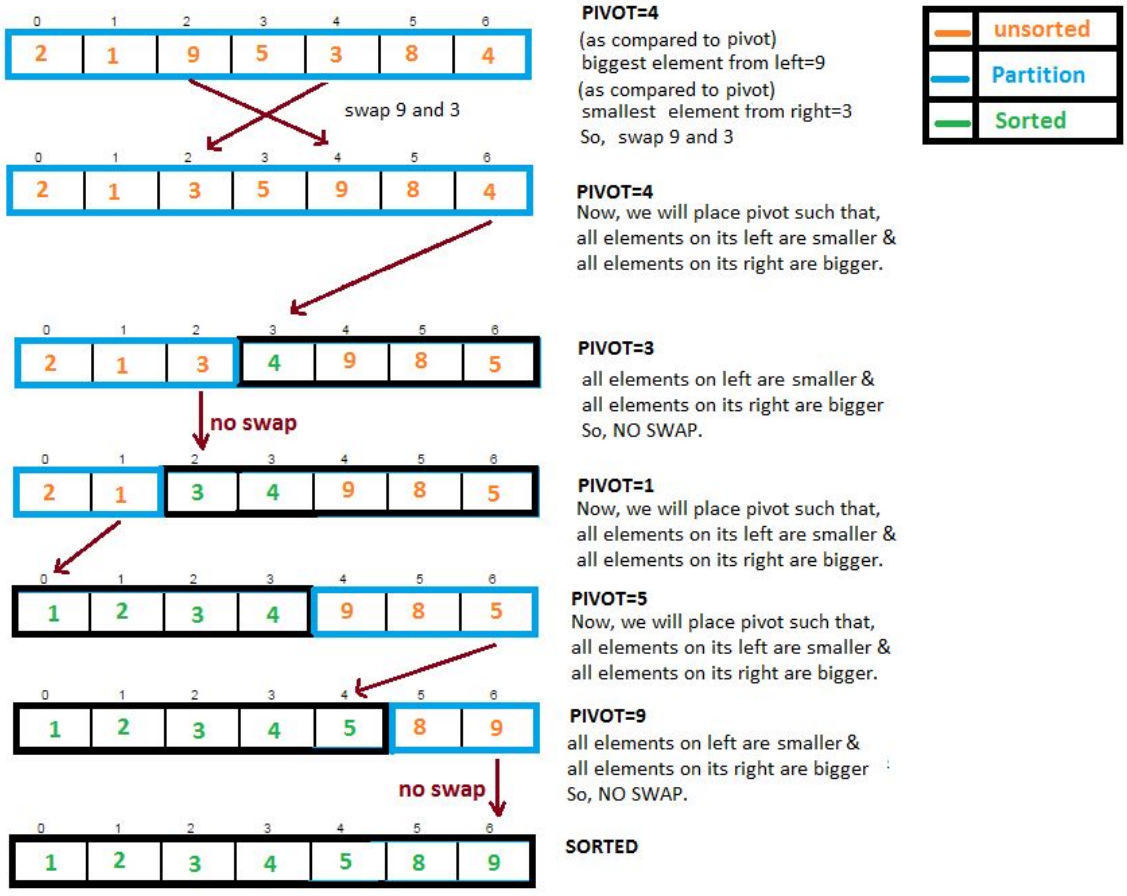
Quick Sort Algorithm[7]

Quick sort is another advanced sorting algorithm that is based on a partitioning method. It will partition the array into smaller arrays, one which will hold the values smaller than the specified partition value (called a pivot), and the other array holds the values that are greater than that value. It works in a recursive manner until it sorts the subarrays and the final array. A reference for an article on this sorting algorithm is in the above footnote.

QuickSort Pivot Algorithm

1. Choose the highest index value as pivot (right-most element)
2. Take two variables to point left and right of the list excluding pivot
3. Left points to the low index
4. Right points to the high
5. While value at left is less than pivot move right
6. While value at right is greater than pivot move left
7. If both step 5 and step 6 does not match swap left and right
8. If left ≥ right, the point where they met is new pivot

Visualization of QuickSort:



[7] https://www.tutorialspoint.com/data_structures_algorithms/quick_sort_algorithm.htm

In class work:

You will take the array: { 34, 45, 81, 24, 9, 2, 49, 75} and show each steps for Selection sort and Merge Sort. Show your TA your work for credit.

Recitation 9 Programming Assignment

There is a link on moodle that contains a file called Recitation9.cpp. Please download this file. You will add some modifications to the code provided. The goal of this exercise is for you to see which sorting algorithm is most efficient. You should come up with different tests, such as using different size arrays. You should randomly generate the large integer arrays (1000+) and have a counter in the code that will count swaps or loops; either will be fine, but be consistent.

NOTE: You have until Sunday at 5pm to complete the exercise. When submitting, you will need to zip files with a screenshot of your findings and place a comment at the top of the code describing what you did to get your findings and which algorithm you saw was the most efficient.