Sorting Algorithm Research

Insertion Sort

Insertion sort is an incremental algorithm that starts with the second number of a list, and finishes at the last- sorting in place. If the current number is greater than the one before, then they are switched. This is repeated until the number is greater than the one before.

This algorithm has time complexity $O(n^2)$. This occurs in the worst case scenario, when the list is sorted in reverse order. In the best case scenario the list is already sorted, so there is only one comparison per loop. In this case the complexity is $O(n)$.

Insertion sort is most effective with smaller or mostly sorted lists, as large unsorted lists can cause large run times.

Merge Sort

Merge sort takes the divide and conquer approach. Dividing an array into two halves, it calls itself on both halves, and then merges the two sorted halves. The merging process adds all the numbers from two arrays to a new array in ascending order.

The time complexity can be expressed as the recurrence relation:

$$T(n) = 2T(\tfrac{n}{2}) + \Theta(n)$$

Solving this gives time complexity $\Theta(nlog(n))$. This applies to the best, worst, and average cases as the array is still recursively split and merging takes linear time.

Merge sort can be used to effectively sort linked lists quickly.

Selection Sort

Selection sort maintains a sorted and an unsorted portion of the input array, in place. Starting from the first element, the minimum of the list is moved to the beginning of the unsorted portion (currently all of the array). The first element is then considered the sorted portion of the array. This process is repeated moving up the array until the sorting is complete.

The time complexity is always $O(n^2)$ as there are two nested for loops. Additionally, numbers are swapped no more than $O(n)$ times, making selection sort useful when write operations are expensive.

Quick Sort

Using the divide and conquer methodology, quick sort picks a pivot. It then rearranges all smaller numbers to the left of the pivot, and larger to the right. The algorithm then recurses over the smaller, and then the larger half. Each time the algorithm runs, the pivot must be chosen consistently. The pivot can be chosen as: the first element, last element, median element, or a random element.

The time complexity can be represented by:

$$T(n) = T(k) + T(n-k-1) + \Theta(n)$$

With the best case when the middle element is picked as the pivot:

$$T(n) = 2T(\tfrac{n}{2}) + \Theta(n)$$

And worst case when the greatest/smallest element is picked as the pivot

$$T(n) = T(n-1) + \Theta(n)$$

Pancake Sort

Pancake sort is an in place sorting algorithm. It begins with the current size being the length of the input array. It reverses the section of the input array between zero and the index of the maximum element, to bring the maximum element to the start of the array. It then reverses the array from zero to the current size, to bring the largest element to the end of the array. Current size is decremented, and this is repeated until the array is sorted.

Pancake sort is a colloquial term for the mathematical problem of trying to sort a stack of pancakes, where a spatula can be inserted beneath any pancake, and flip the entire stack above. The algorithm has a time complexity of $O(n^2)$.