Analysis of Algorithms

Big O notation: Asymptotic Upper bound $\lim_{x \to \infty} \left| \frac{f(x)}{g(x)} \right| < \infty$ (Here limit is "limit superior")

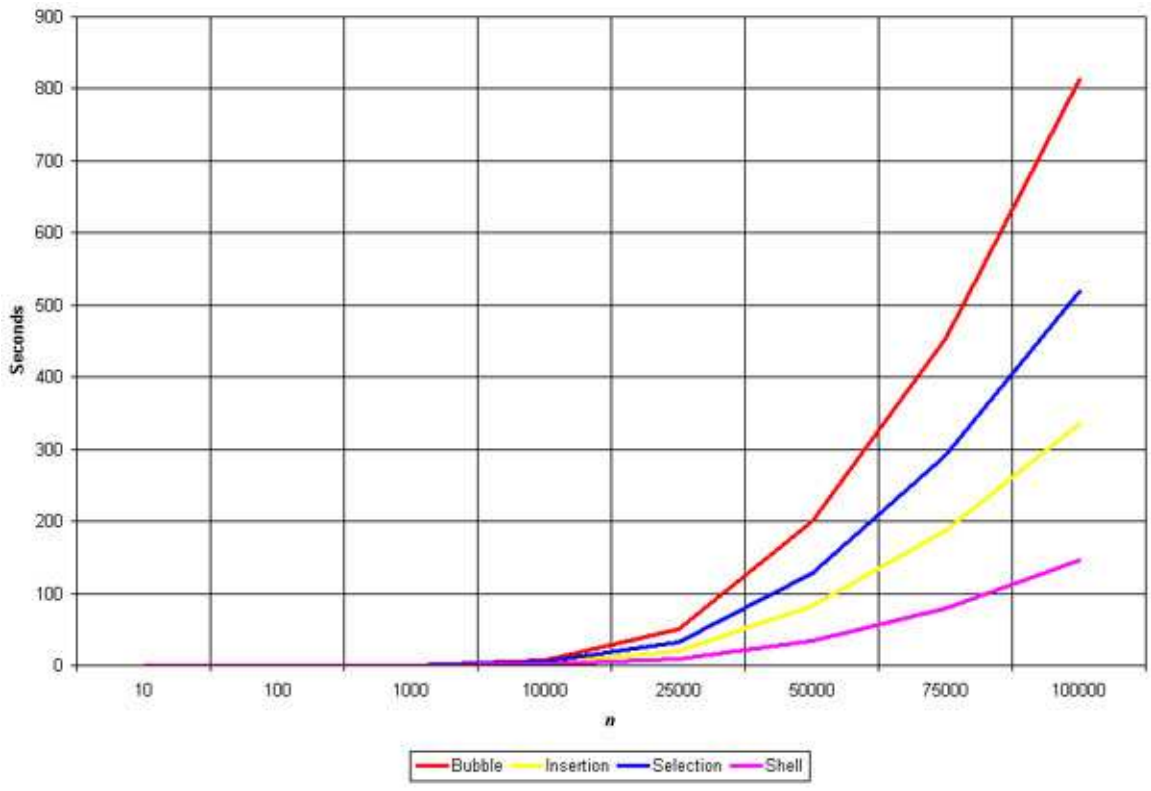
Small o notation: Asymptotically Negligible $\lim_{x \to \infty} \left| \frac{f(x)}{g(x)} \right| = 0$

Example: For $3n + 4$ Big O is n where as Small o is n^2

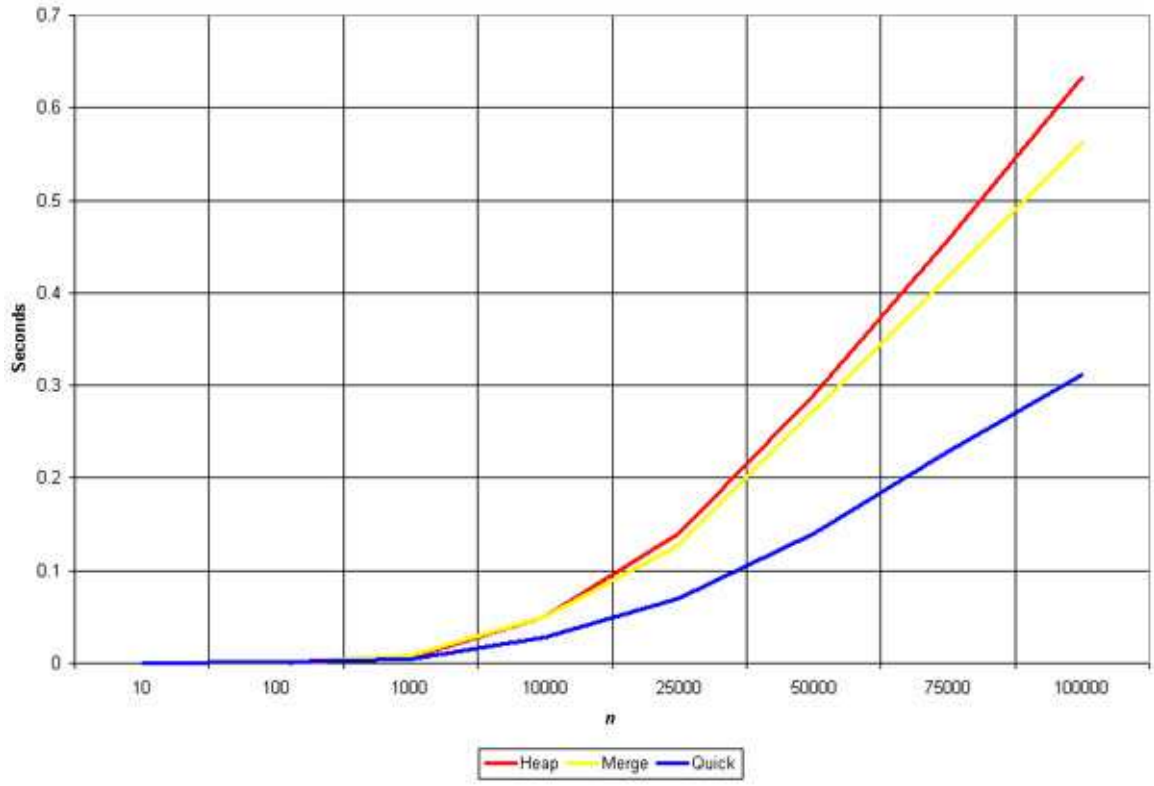
Sorting Algorithms:

There are two classes of Sorting Algorithms:

- $O(n^2)$:
- Bubble Sort
- Insertion Sort
- Selection Sort
- Shell Sort
- $O(n \log n)$
- Heap Sort
- Merge Sort
- Quick Sort



(Efficiency for **O** (n^2) **Sorts-** http://linux.wku.edu/~lamonml/algor/sort/sort.html)



(Efficiency for **O**$(n\log n)$ **Sorts-** http://linux.wku.edu/~lamonml/algor/sort/sort.html)

Comparing different Sorting Algorithms:

Bubble Sort:

Under best-case conditions (the list is already sorted), the bubble sort can approach a constant O(n) level of complexity. General-case is an abysmal $O(n^2)$

While the insertion, selection, and shell sorts also have $O(n^2)$ complexities, they are siginificantly more efficient thanbubble sort.

Insertion Sort:

Like bubble sort, the insertion sort has a complexity of $O(n^2)$. Although it has the same complexity, the insertion sort is a little over twice as efficient as the bubble sort.

Selection Sort:

It yields a 60% performance improvement over the bubble sort, but the insertion sort is over twice as fast as the bubble sort and is just as easy to implement as the selection sort. In short, there really isn't any reason to use the selection sort - use the insertion sort instead.

ShellSort:

The shell sort is by far the fastest of the n^2 class of sorting algorithms. It is more than 5 times faster than the bubble sort and a little over twice as fast as the insertion sort, its closest competitor.

HeapSort:

It is the slowest of the $O(n\log n)$ sorting algorithms but unlike merge and quick sort it does not require massive recursion or multiple arrays to work.

Merge Sort:

The merge sort is slightly faster than the heap sort for larger sets, but it requires twice the memory of the heap sort because of the second array.

Quick Sort:

The quick sort is an in-place, divide-and-conquer, massively recusrsive sot. It can be said as the faster version of the merge sort. The efficiency of the algorithm is majorly impacted by which element is chosen as the pivot point. The worst-case efficienvy of the quick sort is $o(n^2)$ when the list is sorted and left most element is chosen as the pivot. As long as the pivot point is chosen randomly, the quick sort has an algorithmic complexity of $O(n\log n)$.