SORTING METHODS

Sorting is ubiquitous in computer science. Almost every application needs to use some kind of sorting, for example

- binary search needs a sorted array
- ranking students involves sorting
- ranking search engine pages
- compiling a dictionary

⋮

often we are required to sort a huge number of records (of the order of millions or more). Two immediate examples are

- sorting the list of JEE candidates (~ 5,00,000) and the list of AIEEE candidates (~ 11,50,000)
- rank ordering search page results by search engines

Many a time we are required to do this in a real time to offer a fast response to queries.

SORTING PROBLEM

We are given n records (each record containing several well defined fields such as name, date of birth, etc.) and we are required to produce these records in ascending order or descending order of a designated "key" (such as name, date of birth, marks secured, rank, etc.).

In general, a key could be

- a string of characters
- a number
- a pattern of bits
- ⋮

Suppose $A[0], A[1], \ldots, A[n-1]$ represents the arrays of key values. A sorting algorithm "in situ" ("in place") produces an array $A[0], A[1], \ldots, A[n-1]$ which is such that

$$A[0] \leq A[1] \leq \ldots \leq A[n-1]$$

for ascending order and

$$A[0] \geq A[1] \geq \ldots \geq A[n-1]$$

for descending orders.

We study sorting methods for arrays of key values where the two fundamental operations are:

- comparison of key values, say, A[i] and A[j]
- swapping of keys, that is exchanging A[i] and A[j]

An important issue is that of computational complexity which for a sorting method can be expressed as the number of comparisons/exchanges in terms of "n", the number of elements. Here is a summary of the Worstcase complexity (WCC) and average case complexity (ACC) of the sorting methods that we study:

SORTING METHOD	WCC	ACC
Bubble Sort	$O(n^2)$	$O(n^2)$
Selection Sort	$O(n^2)$	$O(n^2)$
Insertion Sort	$O(n^2)$	$O(n^2)$
Shell Sort	$O(n\sqrt{n})$	$O(n\sqrt{n})$
Heap Sort	$O(n \log n)$	$O(n \log n)$
Quick Sort	$O(n^2)$	$O(n \log n)$
Merge Sort	$O(n \log n)$	$O(n \log n)$

Lower Bound on Complexity

It has been shown that any comparison-based sorting method will have a worst case complexity of at least $O(n \log n)$. In fact, it has been shown that even the average case complexity of ~~complexity~~ comparison based sorting algorithms has a lower bound of $O(n \log n)$.

From now on, we make the following assumptions.

- The input to the sorting algorithm is an array of integers:

  $A[0], A[1], \ldots, A[n-1]$

- Unless otherwise stated, we are required to arrange the above elements in ascending order:

  $A[0] \leq A[1] \leq \ldots \leq A[n-1]$

- There could be repititions of values, however we will be mostly working with an array of distinct elements.