Sorting a Sequence of Numbers

A well-studied problem in CS.

Formal Definition of the sorting problem

**Input:** a sequence of n numbers \( a_1, a_2, \ldots, a_n \).

**Output:** A permutation \( a_1', a_2', \ldots, a_n' \) of the input sequence s.t., \( a_1' \leq a_2' \leq \ldots \leq a_n' \).

**Example:** \( (2, 1, 5, 3, 10, 8) \rightarrow (1, 2, 3, 5, 8, 10) \)

The actual input sequence is the instance of the problem.

**Insertion Sort**

A simple and easy algorithm to sort a sequence of numbers.

\[ 0, 5, 2, 4, 6, 13 \]
Idea: Traverse the elements from left to right, stop as soon as you see two adjacent elements \( a_i, a_{i+1} \) s.t. \( a_i > a_{i+1} \)

\[ 2 = 0, 5, 2, 4, 6, 1, 3 \]

We now need to "push" \( a_{i+1} \) to the left, until it is larger than the element to its left.

\[ 2 = 0, 2, 5, \underline{4}, 6, 1, 3 \]

\( \rightarrow \text{sorted} \rightarrow \)

\[ 2 = 0, 2, 4, 5, \underline{6}, 1, 3 \]

\( \rightarrow \text{unsorted} \rightarrow \)

6 is ok, no need to switch elements.
u. 0 2 4 5 6 1
\[\text{\textit{sorted}}\rightarrow\]
\[\text{push 1 to the left}\]
\[\text{\downarrow}\]

5. 0, 1, 2, 4, 5, 6 1
\[\text{\textit{sorted}}\rightarrow\]
\[\text{push 3 to the left}\]
\[\text{\downarrow}\]

0, 1, 2, 3, 4, 5, 6.
\[\text{\textit{sorted}}\rightarrow\]

Done!

Main property of algorithm (Invariant)

At the \((i+1)\)th iteration, all elements at positions 1, 2, \ldots, i are sorted.

\[\Rightarrow\text{After the n-th iteration the entire sequence is sorted.}\]
Number of comparisons:

How many comparisons does the algorithm perform at the $i$th step?
- $i$ comparisons at the worst case.
- Achieved by an instance given in a decreasing order

\[ 6 \quad 5 \quad 4 \quad 3 \quad 2 \quad 1 \]

Total number of comparisons:

\[
\sum_{i=1}^{n} (i-1) = n(n-1) \frac{n^2}{2} = \left( \frac{n^2}{2} \right) - \frac{n}{2}.
\]

Number of comparisons:

$O(n^2)$. 

major term
Pseudo-code for Insertion-Sort:

Assume the sequence of numbers is given in an array $A[i, \ldots, n]$.

**Insertion Sort (A)**

1. for $i = 2$ to $\text{length}(A)$
2.   do $\text{key} = A[i]$, $\text{insert } A[i] \text{ into } A[1, \ldots, i-1]$
3.   $s = i-1$
4. while $s > 0$ and $A[s-1] > \text{key}$
6.   $s = s - 1$
7. $A[s+1] = \text{key}$

In Lines 5-6, we move the elements one position to the right.

In Line 7, we place $\text{key}$ at its "correct" position.
Merge Sort Algorithm

Divide & Conquer Approach

Divide the problem into several subproblems.

Conquer the subproblems by solving them recursively.

Combine the solutions of the subproblems to form the solution of the original problem.

Merge Sort:

Divide: the n-element sequence to be sorted into two subsequences of size \( \frac{n}{2} \) each.

Conquer: Sort the two subsequences recursively.

Combine: Merge the two subsequences to form the whole sorted sequence.
When do we stop dividing?
- When there is only one element: \( n=1 \). In this case, sequence is trivially sorted.

**How to perform Merge?**

**Input:** Two sorted sequence.

**Output:** A merged sorted sequence

**Demonstration:**

\[ 3, 4, 6, 9, 13, 17 \]
\[ 1, 2, 5, 10, 11, 12 \]
\[ \text{null} \]

\[ 1, 2, 3, 4, 5, 6, 9, 10, 11, 12, 13, 17 \]
Idea:
- Place two pointers, at the leftmost positions.
- At each step compare the two elements referred by the pointers.
- Report the smaller element of the two. Then move its pointer one position to the right.
- If one sequence ends, we report all elements in the other sequence one by one.

Number of comparisons:

\[ \frac{n}{2} + \frac{n}{2} - 1 = n - 1 = O(n) \]

As we compare only adjacent elements.
Pseudo-code for Merge Sort

All elements are given in an array $A[1, \ldots, n]$.

$\text{MergeSort} \left(A, p, r\right) - \text{A mergesort procedure that sorts all elements in positions } p, p+1, \ldots, r.$

Assume we have a Merge procedure:

$\text{Merge} \left(A, p, q, r\right) - \text{A procedure that merges } A[p], \ldots, q \text{ and } A[q+1, \ldots, r] \text{ into a single sequence placed in } A[p, \ldots, r].$

[We leave the Merge pseudo-code as an exercise]
Merge Sort (A, p, r)

1. if p < r
   1.1 otherwise base case
2. then
3. \( q = \lfloor \frac{(p+r)}{2} \rfloor \)
4. MergeSort(A, p, q)
5. MergeSort(A, q+1, r)
6. Merge(A, p, q, r)

Simulation:

\[ A = \langle 5, 2, 4, 6, 1, 3, 2, 6 \rangle \]

\[ \begin{array}{cccccccc}
5 & 2 & 4 & 6 & 1 & 3 & 2 & 6 \\
\text{third level split} & \text{second level split} & \text{first level split} & \text{second level split} \\
\end{array} \]
We now present the **Merge** step.

The divide step creates a **Tree**, the leaves represent the base case (singletons).

We climb up the tree by calling **Merge**.

```
1, 2, 2, 3, 4, 5, 6, 6
```

```
1, 2, 3, 6
```

```
1, 2, 3, 6
```

```
1, 3, 2, 6
```

```
5, 2, 4, 6
```

```
5, 2, 4, 6
```

**Divide**: goes from top to **Bottom**.

**Merge** (combine): goes from **bottom** to top.
Number of comparisons

\[ T(n) = \text{number of comparisons on an } \]
\[ n\text{-element sequence.} \]

\( n = 1 \): \[ T(n) = 0 \text{ (no comparisons are made)} \]

\( n = 2 \): \[ T(n) = 1. \]

\( n > 2 \): \text{We recursively solve two subproblems of size } \frac{n}{2} \text{ each. Then we have at most } \]
\[ n \text{ comparisons in Merge:} \]

\[ T(n) \leq 2T\left(\frac{n}{2}\right) + n. \]

\[ \Rightarrow \text{We got a recursive equation, which we need to solve.} \]

\text{solution:} \[ T(n) \leq n \log_2 n \]
Solution: Recursion trees

At the root of the recursion we spend \( \leq n \) comparisons:

```
\[
\begin{array}{c}
\text{n} \\
\text{\text{R}_1, \text{R}_2} \\
\text{n/2, n/2}
\end{array}
\]
```

"Open up" the next levels:

```
\[
\begin{array}{c}
\text{n} \\
\text{n/2} \\
\text{n/4, n/4, n/4, n/4} \\
\text{n/8, n/8, n/8, n/8, n/8, n/8, n/8, n/8}
\end{array}
\]
```

root (level-0)
level -1
level -2
level -3
level -4