Sorting

The art of sorting is that of arranging objects according to some total ordering, that is, each object is comparable to each other. Examples include:

- Ascending
- Descending
- Lexicographical
- Size
- Color

Sorting allows us to search easier and group like things.

BogoSort(A)
while A is not sorted
   Shuffle A

What is the BC runtime? The worst case?
\[ \Theta(n) \] unbounded

Assuming a uniform distribution on the outcome of shuffle, the average case runtime is \( \Omega(n!) \). This is a lower bound because we've hidden a lot of details in the above algorithm. That said, it is \( \Omega(n!) \) because there are \( n! \) permutations of A, so after \( n! \) shuffles, we expect to get the right permutation exactly once.
SS(A)

\[ \begin{align*}
  i &= 0 \\
  \text{While } i < |A| \\
  \text{min} &= i \\
  j &= i + 1 \\
  \text{while } j < |A| \\
  \text{if } A[\text{min}] > A[j] \\
  \text{min} &= j \\
  j &= j + 1 \\
  \text{Swap } A[\text{min}] \text{ and } A[j]
\end{align*} \]

SS(A) sorts A

LIO(A, i) := The first i elements of A are sorted and the remaining elements are all larger.

\[ \begin{align*}
  \text{LIO Init} \\
  \text{Before the loop, since } i = 0, \text{ LIO(A, 0) is vacuously true.}
\end{align*} \]

\[ \begin{align*}
  \text{LIO Maintenance} \\
  \text{Assume LIO(A, i).} \\
  \text{Let } \text{min} = i, \quad j = i + 1
\end{align*} \]

\[ \begin{align*}
  \text{LII Init} \\
  \text{Before the loop, we know LIO(A, i) and } A[\text{min}] = A[j] = \min_{k \in [j]} (A[k])
\end{align*} \]

\[ \begin{align*}
  \text{LII Maintenance} \\
  \text{Assume LII}(A, i, \text{min}, j).
  \text{If } A[\text{min}] > A[j], \text{ then } \text{min}' = j, \text{ hence } A[\text{min}'] = \min_{k \in [j]} (A[k]).
  \text{If } A[\text{min}] \leq A[j], \text{ then } \text{min}' = \text{min}, \text{ hence } A[\text{min}'] = \min_{k \in [j]} (A[k]).
  \text{Lastly, } j' = j + 1, \text{ so in either case, } A[\text{min}'] = \min_{k \in [j']} (A[k]).
  \text{Thus } \text{LII}(A, i, \text{min}', j').
\end{align*} \]
LII Termination
Since |A| = \(i\) + 1, and \(j\) is incremented by 1 each iteration, the loop must break when \(i = |A|\). This yields LII(\(A, i, \min, |A|\)).

As such, \(A[\min] = \min_{i \leq j < |A|} (A[k])\).

We swap \(A[\min]\) and \(A[i]\), so the first \(i + 1\) elements of \(A\) are now sorted because \(A[\min]\) is larger than the first \(i\) sorted elements of \(A\). Moreover, the rest of \(A\) is larger than the first \(i + 1\) elements since \(A[\min]\) was minimal.

Lastly, \(i' = i + 1\), hence LIO(\(A', i'\)).

LIO Termination
As before, the loop breaks when \(i = |A|\).
Since LIO(\(A, |A|\)), all of \(A\) is sorted.

BS(\(A\))
do
    swapped = false
    For \(i = 1\) to \(|A| - 1\)
        If \(A[i-1] > A[i]\)
            Swap \(A[i-1]\) and \(A[i]\)
            swapped = true
    while swapped

BC: \(O(n)\)    WC: \(O(n^2)\)
AC: \(O(n^2)\)  you have to do half of all swaps on average and there are \(\frac{\binom{n+1}{2}}{2}\) possible swaps

Example:

3 5 1 7 2 4 6 0
3 1 5 2 4 6 0 7
1 3 2 4 5 0 6 7
1 2 3 4 0 5 6 7
**IS(A)**

For $i = 1$ to $|A|-1$

Let $j = i$

while $j > 0$ and $A[j-1] > A[j]$

Swap $A[j-1]$ and $A[j]$

$j--$

**Major Advantages**

- Very efficient on small inputs (QS/MS we sort as a BS)
- Very efficient on mostly ordered inputs
- Can sort as it receives new inputs
- Stable

A stable sort is a sorting algorithm that does not change the input ordering of equal keys.

**Ex.** Sort the following by $x$ and then $y$ coordinates.

$\begin{align*}
(1, 2) & \quad (0, 3) & \quad (1, 4) & \quad (2, 2) & \quad (3, 3) \\
& \downarrow \\
(0, 3) & \quad (1, 2) & \quad (1, 4) & \quad (2, 2) & \quad (3, 3) & \text{ x-sorted!} \\
& \downarrow \\
(1, 2) & \quad (0, 3) & \quad (1, 4) & \quad (2, 2) & \quad (3, 3) \\
& \downarrow \\
(1, 2) & \quad (2, 2) & \quad (0, 3) & \quad (1, 4) & \quad (3, 3) \\
& \downarrow \\
(1, 2) & \quad (2, 2) & \quad (0, 3) & \quad (3, 3) & \quad (1, 4) & \text{ y-sorted!} \\
\end{align*}$

Notice that where $y$ is equal, the $x$ component remains sorted.
All of these sorting algorithms have been comparison-based sorts.

There are other types.

\[ RS(A) \]

For the least to most significant digit of elements of \( A \)

For each \( a \in A \) with digit \( d(a,i) \)

Put \( a \) into bucket \( d(a,i) \) at the end

ensures a stable sort so that when we finish the MSB, all of \( A \) is sorted.

The runtime is always \( \Theta(nw) \) where \( w \) is the number of digits of things in \( A \). This is pseudo-polynomial time (runtime in terms of the numeric value of the input). This can be better or worse depending on the application.

The question we want to ask before we go further is what is the best we can do? What is the fastest sort theoretically possible?

It is uncommon that we can answer this sort of question, but with comparison-based sorts, we can!