**Insertion Sort**

```java
for (int i = 1; i < a.length; i++)
    { // insert a[i] into a[0:i-1]
        int t = a[i];
        int j;
        for (j = i - 1; j >= 0 && t < a[j]; j--)
            a[j + 1] = a[j];
        a[j + 1] = t;
    }
```

**Complexity**

△ *Space/Memory*

△ *Time*
- Count a particular operation
- Count number of steps
- Asymptotic complexity

**Comparison Count**

```java
for (int i = 1; i < a.length; i++)
    { // insert a[i] into a[0:i-1]
        int t = a[i];
        int j;
        for (j = i - 1; j >= 0 && t < a[j]; j--)
            a[j + 1] = a[j];
        a[j + 1] = t;
    }
```

**Comparison Count**

△ *Pick an instance characteristic ... n, n = a.length for insertion sort*

△ *Determine count as a function of this instance characteristic.*
Comparison Count
for (j = i - 1; j >= 0 && t < a[j]; j--)
a[j + 1] = a[j];

How many comparisons are made?

number of compares depends on a[]s and t as well as on i

Comparison Count

- Worst-case count = maximum count
- Best-case count = minimum count
- Average count

Worst-Case Comparison Count
for (j = i - 1; j >= 0 && t < a[j]; j--)
a[j + 1] = a[j];

a = [1, 2, 3, 4] and t = 0 => 4 compares
a = [1,2,3,....,i] and t = 0 => i compares
Worst-Case Comparison Count

for (int i = 1; i < n; i++)
  for (j = i - 1; j >= 0 && t < a[j]; j--)
    a[j + 1] = a[j];

total compares = 1 + 2 + 3 + … + (n-1)
    = (n-1)n/2

Step Count

A step is an amount of computing that does not depend on the instance characteristic n

10 adds, 100 subtracts, 1000 multiplies can all be counted as a single step

n adds cannot be counted as 1 step

Step Count

s/e
for (int i = 1; i < a.length; i++)
  { // insert a[i] into a[0:i-1]
    int t = a[i];
    int j;
    for (j = i - 1; j >= 0 && t < a[j]; j--)
      a[j + 1] = a[j];
    a[j + 1] = t;
  }

s/e isn’t always 0 or 1

x = MyMath.sum(a, n);

where n is the instance characteristic has a s/e count of n
Step Count
for (int i = 1; i < a.length; i++)
{// insert a[i] into a[0:i-1]
    int t = a[i];
    int j;
    for (j = i - 1; j >= 0 && t < a[j]; j--)
        a[j + 1] = a[j];
    a[j + 1] = t;
}

Step Count
for (int i = 1; i < a.length; i++)
{ // insert a[i] into a[0:i-1]
    int t = a[i];
    int j;
    for (j = i - 1; j >= 0 && t < a[j]; j--)
        a[j + 1] = a[j];
    a[j + 1] = t;
}

Asymptotic Complexity of Insertion Sort
\(O(n^2)\)
\(\Delta\) What does this mean?

Complexity of Insertion Sort
\(\Delta\) Time or number of operations does not exceed \(c \cdot n^2\) on any input of size \(n\) (\(n\) suitably large).
\(\Delta\) Actually, the worst-case time is Theta\((n^2)\) and the best-case is Theta\((n)\)
\(\Delta\) So, the worst-case time is expected to quadruple each time \(n\) is doubled.
Complexity of Insertion Sort

▲ Is $O(n^2)$ too much time?
▲ Is the algorithm practical?

Practical Complexities

$10^9$ instructions/second

<table>
<thead>
<tr>
<th>$n$</th>
<th>$n^2$</th>
<th>$n \log n$</th>
<th>$n^3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>1mic</td>
<td>10mic</td>
<td>1milli</td>
</tr>
<tr>
<td>10000</td>
<td>10mic</td>
<td>130mic</td>
<td>100milli</td>
</tr>
<tr>
<td>$10^6$</td>
<td>1milli</td>
<td>20milli</td>
<td>17min</td>
</tr>
</tbody>
</table>

Impractical Complexities

$10^9$ instructions/second

<table>
<thead>
<tr>
<th>$n$</th>
<th>$n^2$</th>
<th>$n^4$</th>
<th>$2^n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>17min</td>
<td>3.2 x $10^7$ years</td>
<td>3.2 x $10^{137}$ years</td>
</tr>
<tr>
<td>10000</td>
<td>116 days</td>
<td>???</td>
<td>???</td>
</tr>
<tr>
<td>$10^6$</td>
<td>3 x $10^{10}$ years</td>
<td>?????</td>
<td>?????</td>
</tr>
</tbody>
</table>

Faster Computer Vs Better Algorithm

Algorithmic improvement more useful than hardware improvement.

E.g. $2^n$ to $n^3$