**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**

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

**Comparison Count**

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

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

---

How many comparisons are made?
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

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 isn’t always 0 or 1
x = MyMath.sum(a, n);
where n is the instance characteristic has a s/e count of n
Asymptotic Complexity of Insertion Sort

- $O(n^2)
- What does this mean?

Complexity of Insertion Sort

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

Complexity of Insertion Sort

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

Practical Complexities

$10^9$ instructions/second

<table>
<thead>
<tr>
<th>$n$</th>
<th>$n$</th>
<th>$n\log n$</th>
<th>$n^2$</th>
<th>$n^3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>1 mic</td>
<td>10 mic</td>
<td>1 milli</td>
<td>1 sec</td>
</tr>
<tr>
<td>10000</td>
<td>10 mic</td>
<td>100 mic</td>
<td>100 milli</td>
<td>17 min</td>
</tr>
<tr>
<td>100000</td>
<td>100 mic</td>
<td>1 milli</td>
<td>100 milli</td>
<td>32 years</td>
</tr>
</tbody>
</table>

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 ($j = i - 1; j >= 0 && t < a[j]; j--$
 $a[j + 1] = a[j]$; 1
 $a[j + 1] = t$;

Step Count

for (int i = 1; i < a.length; i++)
{ 2i + 3}

step count for
for (int i = 1; i < a.length; i++)
is n

step count for body of for loop is
$2(1+2+3+...+n-1) + 3(n-1)$

= $(n-1)n + 3(n-1)$

= $(n-1)(n+3)$
Impractical Complexities

$10^9$ instructions/second

<table>
<thead>
<tr>
<th>$n$</th>
<th>$n^2$</th>
<th>$n^{10}$</th>
<th>$2^n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>17 min</td>
<td>$3.2 \times 10^1$ years</td>
<td>$3.2 \times 10^{13}$ years</td>
</tr>
<tr>
<td>10000</td>
<td>116 days</td>
<td>???</td>
<td>???</td>
</tr>
<tr>
<td>$10^9$</td>
<td>$3 \times 10^6$ 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 $n^3$