Insertion Sort

```
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

```
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?

Comparison Count

```
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

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

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

Step Count

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

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
$2(1+2+3+\ldots+n-1) + 3(n-1)$
$= (n-1)n + 3(n-1)$
$= (n-1)(n+3)$

Asymptotic Complexity of Insertion Sort

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

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

Complexity of Insertion Sort

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

Practical Complexities

10^9 instructions/second

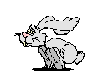
n	*n*	*nlogn*	*n^2*	*n^3*
1000	1mic	10mic	1milli	1sec
10000	10mic	130mic	100milli	17min
10^6	1milli	20milli	17min	32years

Impractical Complexities

10^9 instructions/second

n	*n^4*	*n^{10}*	*2^n*
1000	17min	3.2×10^{13} years	3.2×10^{283} years
10000	116 days	???	???
10^6	3×10^7 years	??????	??????

Faster Computer Vs Better Algorithm



Algorithmic improvement more useful than hardware improvement.

E.g. 2^n to n^3