Complexity Analysis

Complexity

- Space
  - The amount of memory space needed to run the program.
- Time
  - The amount of computational time needed to run the program

We use insertion sort as an example

Pick an instance characteristic … n

n = a.length (the number of elements to be sorted)

Space Complexity for 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;
}
```

Fixed part:
independent of n
ex: instruction space
Variables: i, j,,t

Variable part:
size dependent on n
ex: a[]

Space requirement=
Fixed + Variable

Focus on variable part:
a[] → n

Time Complexity

- 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;
}
```

- Determine the number of comparison count as a function of n

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[], t and 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,…,n] and t = 0 => n 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

In Class Exercise: Best 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];
```

- a = [1, 2, 3, 4] and t = 5 => 1 compares
- a = [1,2,3,…,n] and t = n+1 =>1 compares
- Compute the total number of comparison

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 per execution (s/e)

	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 per execution

s/e isn't always 0 or 1

x = sum(a, n);

where n is the instance characteristic and
sum adds a[0:n-1] has a s/e count of n
(a[0]+a[1]+a[2]+…+a[n-1])

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	

Worst case analysis

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)

	s/e	**frequency**	**total steps**
for (int i = 1; i < a.length; i++)	**1**	**n**	**n**
{ // insert a[i] into a[0:i-1]	**0**	**n-1**	**0**
int t = a[i];	**1**	**n-1**	**n-1**
int j;	**0**	**n-1**	**0**
for (j = i - 1; j >= 0 && t < a[j];j--)	**1**	**(n-1)(n+2)/2**	
a[j + 1] = a[j];	**1**	**n(n-1)/2**	
a[j + 1] = t;	**1**	**n-1**	**n-1**
}	**0**	**n-1**	**0**

Total：n^2+3n-3

In Class Exercise:

Determine the <u>**frequency counts**</u> for all statements in the following program segment

```
for(i=1;i<=n;i++)
    for(j=1;j<=i;j++)
       for(k=1;k<=j;k++)
          x++;
```

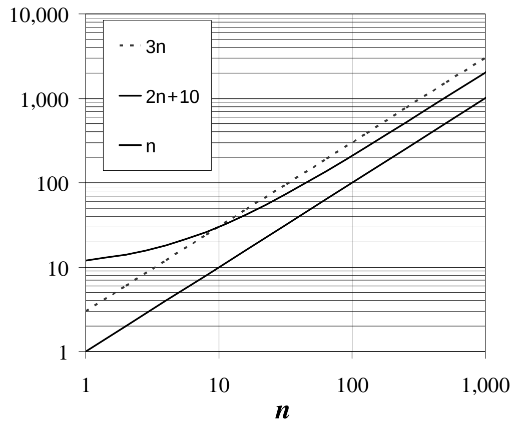

Asymptotic Complexity of Insertion Sort

- (n-1)(n+3)→**O(n²)**
- What does this mean?

Big-Oh Notation

- Given functions $f(n)$ and $g(n)$, we say that $f(n)$ is $O(g(n))$ if there are positive constants c and n_0 such that

  $f(n) \le cg(n)$ for $n \ge n_0$
- Example: $2n + 10$ is $O(n)$
  - $2n + 10 \le cn$
  - $(c - 2)\, n \ge 10$
  - $n \ge 10/(c - 2)$
  - Pick $c = 3$ and $n_0 = 10$

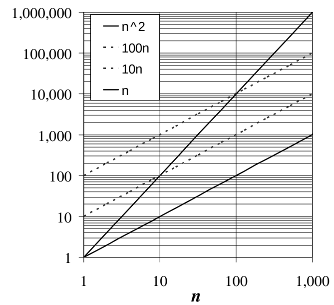


Big-Oh Example

Example: the function n^2 is not $O(n)$

- $n^2 \le cn$
- $n \le c$

the above inequality cannot be satisfied since c must be a constant



Big-Oh and Growth Rate

- The big-Oh notation gives an upper bound on the growth rate of a function
- The statement "$f(n)$ is $O(g(n))$" means that the growth rate of $f(n)$ is no more than the growth rate of $g(n)$
- We can use the big-Oh notation to rank functions according to their growth rate

	$f(n)$ is $O(g(n))$	$g(n)$ is $O(f(n))$
$g(n)$ grows more	Yes	No
$f(n)$ grows more	No	Yes
Same growth	Yes	Yes

Complexity of Insertion Sort

- Time or number of operations does not exceed **c.n²** on any input of size **n** (**n** suitably large).
- Actually, the worst-case time is $\Theta(\mathbf{n^2})$ and the best-case is $\Theta(\mathbf{n})$
- So, the worst-case time is expected to quadruple each time **n** is doubled

The definition of Θ (n) will be discussed finally.

Complexity of Insertion Sort

- Is **O(n²)** too much time?
- Is the algorithm practical?

Practical Complexities

10^9 instructions/second

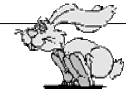
1000	1mic	10mic	1milli	1sec
10000	10mic	130mic	100milli	17min
10^6	1milli	20milli	17min	32years

Impractical Complexities

109 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 v.s Better algorithm



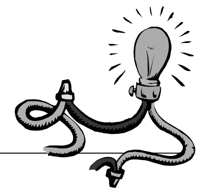
Algorithmic improvement more useful than hardware improvement.

E.g. 2^n to n^3

Relatives of Big-Oh

- **big-Omega**
  - f(n) is $\Omega(g(n))$ if there is a constant $c > 0$ and an integer constant $n_0 \geq 1$ such that $f(n) \geq c \bullet g(n)$ for $n \geq n_0$
- **big-Theta**
  - f(n) is $\Theta(g(n))$ if there are constants $c' > 0$ and $c'' > 0$ and an integer constant $n_0 \geq 1$ such that $c' \bullet g(n) \leq f(n) \leq c'' \bullet g(n)$ for $n \geq n_0$
- **little-oh**
  - f(n) is $o(g(n))$ if, for any constant $c > 0$, there is an integer constant $n_0 \geq 0$ such that $f(n) \leq c \bullet g(n)$ for $n \geq n_0$
- **little-omega**
  - f(n) is $\omega(g(n))$ if, for any constant $c > 0$, there is an integer constant $n_0 \geq 0$ such that $f(n) \geq c \bullet g(n)$ for $n \geq n_0$

Intuition for Asymptotic Notation



- **Big-Oh**
  - f(n) is O(g(n)) if f(n) is asymptotically **less than or equal** to g(n)
- **big-Omega**
  - f(n) is $\Omega(g(n))$ if f(n) is asymptotically **greater than or equal** to g(n)
- **big-Theta**
  - f(n) is $\Theta(g(n))$ if f(n) is asymptotically **equal** to g(n)
- **little-oh**
  - f(n) is o(g(n)) if f(n) is asymptotically **strictly less** than g(n)
- **little-omega**
  - f(n) is $\omega(g(n))$ if f(n) is asymptotically **strictly greater** than g(n)

Example

$$f(n) = 2n^2 + n + 4$$
$$g(n) = n^2$$
$$f(n) = \theta(g(n))$$

$$c' = 1, c'' = 7$$
$$1 * g(n) <= f(n) <= 7 * g(n), \ for\ n >= 1$$
$$1*1^2 <= 2*1^2 + 1 + 4 <= 7*1^2$$

Example

$$f(n) = a_m n^m + a_{m-1} n^{m-1} + \cdots + a_1 n + a_0$$
$$f(n) = O(n^m)$$

$$\begin{aligned} f(n) &= a_m n^m + a_{m-1} n^{m-1} + \cdots a_1 n + a_0 \\ &= \sum_{i=0}^{m} a_i n^i \\ &\le \sum_{i=0}^{m} |a_i| n^i \\ &\le n^m \cdot \sum_{i=0}^{m} |a_i| n^{i-m} \\ &\le n^m \cdot \sum_{i=0}^{m} |a_i| \end{aligned}$$

Homework

Determine the frequency counts for all statements and analysis the complexity for the program segment

```
for(int i=0;i<n;i++)
  { // n is number of elements stored in array
    for (int j=0;j<n-i;j++)
        {
         if(array[j]>array[j+1])
           Swap(array[j],array[j+1]);
        }
 }
```