COMP26120: Introducing Complexity Analysis (2018/19)

Lucas Cordeiro

lucas.cordeiro@manchester.ac.uk
Introducing Complexity Analysis

- Textbook:
  - *Algorithm Design and Applications*, Goodrich, Michael T. and Roberto Tamassia (chapter 1)
  - *Introduction to Algorithms*, Cormen, Leiserson, Rivest, Stein (chapters 2 and 3)
Motivating Example (COMP11212)

• What does this code fragment represent? What is its complexity?

... 
for(i = 0; i < N-1; i++) {
    for(j = 0; j < N-1; j++) {
        if (a[j] > a[j+1]) {
            t = a[j];
            a[j] = a[j+1];
            a[j+1] = t;
        }
    }
}
...

- This is bubble sort
- There are two loops
- Both loops make \( n-1 \) iterations so we have \((n-1)^2\) iterations
- The complexity is \( O(n^2) \)

Perform worst case analysis and ignore constants
Intended Learning Outcomes

- Define asymptotic notation, functions, and running times
- Analyse the running time used by an algorithm via asymptotic analysis
- Provide examples of asymptotic analysis using the insertion sorting algorithm
Asymptotic Performance

• In this course, we care most about *asymptotic performance*
  - We focus on the **infinite set of large** $n$ ignoring small values of $n$
    - The best choice for all, but very small inputs
  - How does the algorithm behave as the problem size gets very large?
    - Running time
    - Memory/storage requirements
    - Bandwidth/power requirements/logic gates/etc.
Asymptotic Notation

• By now you should have an **intuitive feel** for asymptotic (big-O) notation:
  - *What does* O(n) *running time mean?*  O(n^2)?  O(lg n)?
  - *How does asymptotic running time relate to asymptotic memory usage?*

• Our first task is to **define this notation more formally and completely**
Search Problem
(Arbitrary Sequence)

Input
• sequence of numbers \((a_1, \ldots a_n)\)
• search for a specific number \((q)\)

Output
• index or NIL

\[ a_1, a_2, a_3, \ldots, a_n; \; q \]

\[ 2 \; 5 \; 4 \; 10 \; 7; \; 5 \]

\[ 2 \; 5 \; 4 \; 10 \; 7; \; 9 \]

\[ j \rightarrow 2 \]

\[ NIL \]
Linear Search

```plaintext
j=1
while j<=length(A) and A[j]!=q
do j++
if j<=length(A) then return j
else return NIL
```

• Worst case: \( f(n)=n \), average case: \( n/2 \)

• Can we do better using this approach?
  ▪ this is a lower bound for the search problem in an arbitrary sequence
A Search Problem (Sorted Sequence)

**Input**
- sequence of numbers \((a_1 \leq a_2, \ldots, a_{n-1} \leq a_n)\)
- search for a specific number \((q)\)

\[
a_1, a_2, a_3, \ldots, a_n; \quad q
\]

\[
2 \quad 4 \quad 5 \quad 7 \quad 10; \quad 10
\]

\[
2 \quad 4 \quad 5 \quad 7 \quad 10; \quad 8
\]

**Output**
- index or NIL

\[
j
\]

\[
5
\]

\[
NIL
\]

Did the sorted sequence help in the search?
Binary Search

• Assume that the array is sorted and then perform successive divisions

left=1
right=length(A)
do
  j=(left+right)/2
  if A[j]==q then return j
  else if A[j]>q then right=j-1
  else left=j+1
while left<=right
return NIL
Analysis of the Binary Search

• How many times the loop is executed?
  ▪ At each interaction the number of positions $n$ is cut in half
  ▪ How many times do we cut in half $n$ to reach 1?
    ○ $\lg_2 n$

\[
\begin{array}{c}
3 & 4 & 6 & 8 & 9 & 10 & 12 & 15 \\
\end{array}
\]

\[
\lg_2 n = x \iff n = 2^x
\]

\[
\lg_2 8 = 3
\]
Analysis of Algorithms (COMP15111)

• Analysis is performed with respect to a computational model

• We will usually use a generic uniprocessor random-access machine (RAM)
  ▪ All memory equally expensive to access
  ▪ Instructions executed one after another (no concurrent operations)
  ▪ All reasonable instructions take unit time
    ◦ Except, of course, function calls
  ▪ Constant word size
    ◦ Unless we are explicitly manipulating bits
Input Size

• **Time** and **space** complexity
  - This is generally a function of the input size
    - E.g., sorting, multiplication
  - How we characterize input size depends:
    - Sorting: number of input items
    - Multiplication: total number of bits
    - Graph algorithms: number of nodes & edges
    - Etc
Running Time

- Number of **primitive steps** that are executed
  - Except for time of executing a function call most statements roughly require the same amount of time
    - \( f = (g + h) - (i + j) \)
    - \( y = m \times x + b \)
    - \( c = 5 / 9 \times (t - 32) \)
    - \( z = f(x) + g(y) \)

- We can be more exact if need be

```assembly
add t0, g, h # temp t0 = g + h
add t1, i, j # temp t1 = i + j
sub f, t0, t1 # f = t0 - t1
```
Analysis

• **Worst case**
  - Provides an upper bound on running time
  - An absolute guarantee

• **Average case**
  - Provides the expected running time
  - Very useful, but treat with care: what is “average”?  
    - Random (equally likely) inputs
    - Real-life inputs
An Example: Insertion Sort

```
InsertionSort(A, n) {
    for i = 2 to n {
        key = A[i]
        j = i - 1;
        while (j > 0) and (A[j] > key) {
            j = j - 1
        }
        A[j+1] = key
    }
}
```
An Example: Insertion Sort

```
30 10 40 20
1 2 3 4
```

\[
i = \emptyset \quad j = \emptyset \quad key = \emptyset \\
A[j] = \emptyset \quad A[j+1] = \emptyset
\]

```
InsertionSort(A, n) {
    for i = 2 to n {
        key = A[i]
        j = i - 1;
        while (j > 0) and (A[j] > key) {
            j = j - 1
        }
        A[j+1] = key
    }
}
```
An Example: Insertion Sort

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>10</td>
<td>40</td>
<td>20</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

i = 2  j = 1  key = 10

InsertionSort(A, n) {
    for i = 2 to n {
        key = A[i]
        j = i - 1;
        while (j > 0) and (A[j] > key) {
            j = j - 1
        }
        A[j+1] = key
    }
}
An Example: Insertion Sort

InsertionSort(A, n) {
    for i = 2 to n {
        key = A[i]
        j = i - 1;
        while (j > 0) and (A[j] > key) {
            j = j - 1
        }
        A[j+1] = key
    }
}
An Example: Insertion Sort

```
<table>
<thead>
<tr>
<th>30</th>
<th>30</th>
<th>40</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>
```

InsertionSort(A, n) {
    for i = 2 to n {
        key = A[i]
        j = i - 1;
        while (j > 0) and (A[j] > key) {
            j = j - 1
        }
        A[j+1] = key
    }
}

i = 2   j = 1   key = 10
An Example: Insertion Sort

<table>
<thead>
<tr>
<th>30</th>
<th>30</th>
<th>40</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

InsertionSort(A, n) {
    for i = 2 to n {
        key = A[i]
        j = i - 1;
        while (j > 0) and (A[j] > key) {
            j = j - 1
        }
        A[j+1] = key
    }
}

i = 2    j = 0    key = 10
An Example: Insertion Sort

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>30</td>
<td>40</td>
<td>20</td>
</tr>
</tbody>
</table>

1 2 3 4

InsertionSort(A, n) {
    for i = 2 to n {
        key = A[i]
        j = i - 1;
        while (j > 0) and (A[j] > key) {
            j = j - 1
        }
        A[j+1] = key
    }
}
An Example: Insertion Sort

```
InsertionSort(A, n) {
    for i = 2 to n {
        key = A[i]
        j = i - 1;
        while (j > 0) and (A[j] > key) {
            j = j - 1
        }
        A[j+1] = key
    }
}
```
An Example: Insertion Sort

<table>
<thead>
<tr>
<th>10</th>
<th>30</th>
<th>40</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

InsertionSort(A, n) {
    for i = 2 to n {
        key = A[i]
        j = i - 1;
        while (j > 0) and (A[j] > key) {
            j = j - 1
        }
        A[j+1] = key
    }
}
An Example: Insertion Sort

<table>
<thead>
<tr>
<th>10</th>
<th>30</th>
<th>40</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

InsertionSort(A, n) {
    for i = 2 to n {
        key = A[i]
        j = i - 1;
        while (j > 0) and (A[j] > key) {
            j = j - 1
        }
        A[j+1] = key
    }
}

i = 3    j = 0    key = 40
An Example: Insertion Sort

<table>
<thead>
<tr>
<th></th>
<th>10</th>
<th>30</th>
<th>40</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

```
InsertionSort(A, n) {
    for i = 2 to n {
        key = A[i]
        j = i - 1;
        while (j > 0) and (A[j] > key) {
            j = j - 1
        }
        A[j+1] = key
    }
}
```

i = 3  j = 0  key = 40
An Example: Insertion Sort

```
InsertionSort(A, n) {
    for i = 2 to n {
        key = A[i]
        j = i - 1;
        while (j > 0) and (A[j] > key) {
            j = j - 1
        }
        A[j+1] = key
    }
}
```

<table>
<thead>
<tr>
<th>i = 3</th>
<th>j = 2</th>
<th>key = 40</th>
</tr>
</thead>
</table>

10 30 40 20

1 2 3 4
An Example: Insertion Sort

```
InsertionSort(A, n) {
    for i = 2 to n {
        key = A[i]
        j = i - 1;
        while (j > 0) and (A[j] > key) {
            j = j - 1
        }
        A[j+1] = key
    }
}
```

<table>
<thead>
<tr>
<th>i = 3  j = 2  key = 40</th>
</tr>
</thead>
<tbody>
<tr>
<td>A[j] = 30</td>
</tr>
<tr>
<td>A[j+1] = 40</td>
</tr>
</tbody>
</table>
An Example: Insertion Sort

```plaintext
InsertionSort(A, n) {
    for i = 2 to n {
        key = A[i]
        j = i - 1;
        while (j > 0) and (A[j] > key) {
            j = j - 1
        }
        A[j+1] = key
    }
}
```

```
10  30  40  20
1  2  3  4
```

```
i = 3  j = 2  key = 40
```
An Example: Insertion Sort

<table>
<thead>
<tr>
<th>10</th>
<th>30</th>
<th>40</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

**InsertionSort**(A, n) {
  for i = 2 to n {
    key = A[i]
    j = i - 1;
    while (j > 0) and (A[j] > key) {
      j = j - 1
    }
    A[j+1] = key
  }
}

i = 4   j = 2   key = 40
An Example: Insertion Sort

<table>
<thead>
<tr>
<th>10</th>
<th>30</th>
<th>40</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

InsertionSort(A, n) {
    for i = 2 to n {
        key = A[i]
        j = i - 1;
        while (j > 0) and (A[j] > key) {
            j = j - 1
        }
        A[j+1] = key
    }
}
An Example: Insertion Sort

InsertionSort(A, n) {
    for i = 2 to n {
        key = A[i]
        j = i - 1;
        while (j > 0) and (A[j] > key) {
            j = j - 1
        }
        A[j+1] = key
    }
}

<table>
<thead>
<tr>
<th>10</th>
<th>30</th>
<th>40</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

i = 4  j = 2  key = 20

1  2  3  4  5  6  7  8  9  10
**An Example: Insertion Sort**

```
InsertionSort(A, n) {
    for i = 2 to n {
        key = A[i]
        j = i - 1;
        while (j > 0) and (A[j] > key) {
            j = j - 1
        }
        A[j+1] = key
    }
}
```

- **i = 4**  **j = 3**  **key = 20**

**Array:**

<table>
<thead>
<tr>
<th></th>
<th>10</th>
<th>30</th>
<th>40</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>
An Example: Insertion Sort

InsertionSort(A, n) {
    for i = 2 to n {
        key = A[i]
        j = i - 1;
        while (j > 0) and (A[j] > key) {
            j = j - 1
        }
        A[j+1] = key
    }
}

<table>
<thead>
<tr>
<th></th>
<th>10</th>
<th>30</th>
<th>40</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>
i = 4  j = 3  key = 20
An Example: Insertion Sort

<table>
<thead>
<tr>
<th>10</th>
<th>30</th>
<th>40</th>
<th>40</th>
</tr>
</thead>
</table>

1 2 3 4

InsertionSort(A, n) {
    for i = 2 to n {
        key = A[i]
        j = i - 1;
        while (j > 0) and (A[j] > key) {
            j = j - 1
        }
        A[j+1] = key
    }
}

i = 4  j = 3  key = 20

An Example: Insertion Sort
An Example: Insertion Sort

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>30</td>
<td>40</td>
<td>40</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

InsertionSort(A, n) {
    for i = 2 to n {
        key = A[i]
        j = i - 1;
        while (j > 0) and (A[j] > key) {
            j = j - 1
        }
        A[j+1] = key
    }
}

An Example: Insertion Sort

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>i = 4</td>
<td>j = 3</td>
<td>key = 20</td>
<td></td>
</tr>
</tbody>
</table>
An Example: Insertion Sort

\[
\begin{array}{cccc}
10 & 30 & 40 & 40 \\
1 & 2 & 3 & 4
\end{array}
\]

\[
i = 4 \quad j = 3 \quad \text{key} = 20 \\
\]

\[
\text{InsertionSort}(A, n) \{ \\
\quad \text{for } i = 2 \text{ to } n \{ \\
\quad \quad \text{key} = A[i] \\
\quad \quad j = i - 1; \\
\quad \quad \text{while } (j > 0) \text{ and } (A[j] > \text{key}) \{ \\
\quad \quad \quad A[j+1] = A[j] \\
\quad \quad \quad j = j - 1 \\
\quad \quad \} \\
\quad \quad A[j+1] = \text{key} \\
\quad \}\}
\]
An Example: Insertion Sort

```
InsertionSort(A, n) {
    for i = 2 to n {
        key = A[i]
        j = i - 1;
        while (j > 0) and (A[j] > key) {
            j = j - 1
        }
        A[j+1] = key
    }
}
```

<table>
<thead>
<tr>
<th>10</th>
<th>30</th>
<th>40</th>
<th>40</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

i = 4  j = 2  key = 20
An Example: Insertion Sort

InsertionSort(A, n) {
    for i = 2 to n {
        key = A[i]
        j = i - 1;
        while (j > 0) and (A[j] > key) {
            j = j - 1
        }
        A[j+1] = key
    }
}

InsertionSort(A, n) {
    for i = 2 to n {
        key = A[i]
        j = i - 1;
        while (j > 0) and (A[j] > key) {
            j = j - 1
        }
        A[j+1] = key
    }
}

10 30 40 40
1 2 3 4

i = 4  j = 2  key = 20
An Example: Insertion Sort

$$\begin{array}{cccc}
10 & 30 & 30 & 40 \\
1 & 2 & 3 & 4
\end{array}$$

$$\begin{array}{cccc}
i = 4 & j = 2 & \text{key} = 20 \\
\end{array}$$

InsertionSort(A, n) {
    for i = 2 to n {
        key = A[i]
        j = i - 1;
        while (j > 0) and (A[j] > key) {
            j = j - 1
        }
        A[j+1] = key
    }
}
An Example: Insertion Sort

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>10</th>
<th>30</th>
<th>30</th>
<th>40</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

InsertionSort(A, n) {
    for i = 2 to n {
        key = A[i]
        j = i - 1;
        while (j > 0) and (A[j] > key) {
            j = j - 1
        }
        A[j+1] = key
    }
}

i = 4  j = 2  key = 20

An Example: Insertion Sort
An Example: Insertion Sort

InsertionSort(A, n) {
    for i = 2 to n {
        key = A[i]
        j = i - 1;
        while (j > 0) and (A[j] > key) {
            j = j - 1
        }
        A[j+1] = key
    }
}

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>30</td>
<td>30</td>
<td>40</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

i = 4  j = 1  key = 20

InsertionSort(A, n) {
    for i = 2 to n {
        key = A[i]
        j = i - 1;
        while (j > 0) and (A[j] > key) {
            j = j - 1
        }
        A[j+1] = key
    }
}
An Example: Insertion Sort

```
InsertionSort(A, n) {
    for i = 2 to n {
        key = A[i]
        j = i - 1;
        while (j > 0) and (A[j] > key) {
            j = j - 1
        }
        A[j+1] = key
    }
}
```

InsertionSort(A, n) {
    for i = 2 to n {
        key = A[i]
        j = i - 1;
        while (j > 0) and (A[j] > key) {
            j = j - 1
        }
        A[j+1] = key
    }
}

<table>
<thead>
<tr>
<th>10</th>
<th>30</th>
<th>30</th>
<th>40</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

i = 4  j = 1  key = 20
An Example: Insertion Sort

InsertionSort(A, n) {
    for i = 2 to n {
        key = A[i]
        j = i - 1;
        while (j > 0) and (A[j] > key) {
            j = j - 1
        }
        A[j+1] = key
    }
}

10 20 30 40

1 2 3 4

i = 4  j = 1  key = 20
An Example: Insertion Sort

InsertionSort(A, n) {
    for i = 2 to n {
        key = A[i]
        j = i - 1;
        while (j > 0) and (A[j] > key) {
            j = j - 1
        }
        A[j+1] = key
    }
}

Done!
InsertionSort(A, n) {
    for i = 2 to n {
        key = A[i]
        j = i - 1;
        while (j > 0) and (A[j] > key) {
            j = j - 1
        }
        A[j+1] = key
    }
}
Insertion Sort

```c
InsertionSort(A, n) {
    for i = 2 to n {
        key = A[i]
        j = i - 1;
        while (j > 0) and (A[j] > key) {
            j = j - 1
        }
        A[j+1] = key
    }
}
```

How many times will this loop execute?
Insertion Sort

InsertionSort(A, n) {
    for i = 2 to n {
        key = A[i]
        j = i - 1;
        while (j > 0) and (A[j] > key) {
            j = j - 1
        }
        A[j+1] = key
    }
}

What is the post-condition for this loop?
Insertion Sort

InsertionSort(A, n) {
    for i = 2 to n {
        key = A[i]
        j = i - 1;
        while (j > 0) and (A[j] > key) {
            j = j - 1
        }
        A[j+1] = key
    }
}

Invariant: A[1..i-1] consists of the elements originally in A[1..i-1], but in sorted order
Insertion Sort

InsertionSort(A, n) {
    for i = 2 to n {
        key = A[i]
        j = i - 1;
        while (j > 0) and (A[j] > key) {
            j = j - 1
        }
        A[j+1] = key
    }
}

Termination: when i == n+1 we have A[1..i-1] which leads to A[1..n]
Insertion Sort

InsertionSort(A, n) {
    for i = 2 to n {
        key = A[i]
        j = i - 1;
        while (j > 0) and (A[j] > key) {
            j = j - 1
        }
        A[j+1] = key
    }
}

\[ T = t_2 + t_3 + \ldots + t_n \] where \( t_i \) is number of while expression evaluations for the \( i^{th} \) for loop iteration
Analysing Insertion Sort

• $T(n) = c_1n + c_2(n-1) + c_3(n-1) + c_4T + c_5(T - (n-1)) + c_6(T - (n-1)) + c_7(n-1)$

• What can $T$ be?
  - Best case -- inner loop body never executed
  - $T = t_2 + t_3 + \ldots + t_n$
    $$\sum_{j=2}^{n} t_j = t_2 + t_3 + \ldots + t_n = 1 + 1 + \ldots + 1 = n - 1$$
    $$T(n) = c_1n + c_2(n-1) + c_3(n-1) + c_4(n-1) + c_7(n-1)$$
    $$T(n) = (c_1 + c_2 + c_3 + c_4 + c_7) \cdot n - (c_2 + c_3 + c_4 + c_7)$$
  - $T(n) = an - b$
Gaussian Closed Form can be defined as:

\[ \sum_{j=1}^{n} j = 1 + 2 + \cdots + n = \frac{n(n+1)}{2} \]

Thus, we have:

\[ \sum_{j=2}^{n} j = 2 + 3 + \cdots + n = \frac{n(n+1)}{2} - 1 \]

Similarly, we obtain:

\[ \sum_{j=2}^{n} (j - 1) = \cdots = \frac{n(n+1)}{2} - n = \frac{n(n+1) - 2n}{2} = \frac{n(n-1)}{2} \]
Analysing Insertion Sort

- Worst case -- inner loop body executed for all previous elements

\[ \sum_{j=2}^{n} j = \frac{n(n+1)}{2} - 1 \]
\[ \sum_{j=2}^{n} (j-1) = \frac{n(n-1)}{2} \]

\[ T(n) = c_1 n + c_2 (n-1) + c_3 (n-1) + c_4 \left( \frac{n(n+1)}{2} - 1 \right) + c_5 \left( \frac{n(n-1)}{2} \right) + c_6 \left( \frac{n(n-1)}{2} \right) \]
\[ + c_7 (n-1) = \left( \frac{c_4}{2} + \frac{c_5}{2} + \frac{c_6}{2} \right) n^2 + \left( c_1 + c_2 + c_3 + \frac{c_4}{2} - \frac{c_5}{2} - \frac{c_6}{2} + c_7 \right) n - \left( c_2 + c_3 + c_4 + c_7 \right) \]

- Average case

- **???

- **???

- \[ T(n) = an^2 + bn - c \]
Simplifications

- Abstract statement costs
- Order of growth

![Graph showing running time vs. inputs size with lines for best, average, and worst cases.](image)
Upper Bound Notation

- InsertionSort’s run time is $O(n^2)$
  - run time is in $O(n^2)$
  - Read O as “Big-O”
- In general a function
  - $f(n)$ is $O(g(n))$ if there exist positive constants $c$ and $n_0$ such that $f(n) \leq c \cdot g(n)$ for all $n \geq n_0$
Insertion Sort Is $O(n^2)$

• Proof:
  § Use the formal definition of $O$ to demonstrate that $an^2 + bn + c = O(n^2)$
  
  $O(g(n)) = \{ f(n) : \text{there exist positive constants } c \text{ and } n_0 \text{ such that } 0 \leq f(n) \leq cg(n) \text{ for all } n \geq n_0 \}$.
  
  § If any of $a$, $b$, and $c$ are less than 0 replace the constant with its absolute value
    - $0 \leq f(n) \leq k \cdot g(n)$ for all $n \geq n_0$ ($k$ and $n_0$ must be positive)
    - $0 \leq an^2 + bn + c \leq kn^2$
    - $0 \leq a + b/n + c/n^2 \leq k$

• Question
  § Is InsertionSort $O(n)$?
Lower Bound Notation

- InsertionSort’s run time is $\Omega(n)$
- In general a function
  - $f(n)$ is $\Omega(g(n))$ if there exist positive constants $c$ and $n_0$ such that $0 \leq c \cdot g(n) \leq f(n)$ $\forall n \geq n_0$
- Proof:
  - Suppose runtime is $an + b$
    - $0 \leq cn \leq an + b$
    - $0 \leq c \leq a + b/n$
Asymptotic Tight Bound

• A function $f(n)$ is $\Theta(g(n))$ if there exist positive constants $c_1$, $c_2$, and $n_0$ such that $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$ for all $n \geq n_0$

• Theorem
  ▪ $f(n)$ is $\Theta(g(n))$ iff $f(n)$ is both $O(g(n))$ and $\Omega(g(n))$
Exercise: Asymptotic Notation

- Use the formal definition of $\Theta$

$$\Theta(g(n)) = \{ f(n) : \text{there exist positive constants } c_1, c_2, \text{ and } n_0 \text{ such that } 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n) \text{ for all } n \geq n_0 \}.$$  

To demonstrate that $\frac{1}{2} n^2 - 3n = \Theta(n^2)$

Solution:

$0 \leq c_1 n^2 \leq \frac{1}{2} n^2 - 3n \leq c_2 n^2$ for all $n \geq n_0$

Note that $c_1$ and $c_2$ must be positive constants.

For sufficiently large $n$, the term $\frac{1}{2}$ is kept.

$n=7$ is the smallest value for $c_1$ to be a positive constant.

$1$Within set notation, a colon means “such that”
Exercise: Asymptotic Notation

- Use the formal definition of $\Theta$

$$\Theta(g(n)) = \{ f(n) : \text{there exist positive constants } c_1, c_2, \text{ and } n_0 \text{ such that}$$
$$0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n) \text{ for all } n \geq n_0 \}.$$\footnote{1}

to demonstrate that $6n^3 \neq \Theta(n^2)$

Solution:

$$c_1 n^2 \leq 6n^3 \leq c_2 n^2 \text{ for all } n \geq n_0$$

$$6n \leq c_2 \therefore n \leq \frac{c_2}{6}$$

This can not be true for sufficiently large $n$
since $c_2$ must be a constant
Other Asymptotic Notations

• A function $f(n)$ is $o(g(n))$ if $\exists$ positive constants $c$ and $n_0$ such that
  \[ f(n) < c \cdot g(n) \quad \forall \quad n \geq n_0 \]

• A function $f(n)$ is $\omega(g(n))$ if $\exists$ positive constants $c$ and $n_0$ such that
  \[ c \cdot g(n) < f(n) \quad \forall \quad n \geq n_0 \]

• Intuitively,
  - $o()$ is like $<$
  - $\omega()$ is like $>$
  - $\Theta()$ is like $=$
  - $O()$ is like $\leq$
  - $\Omega()$ is like $\geq$
Asymptotic Comparisons

- We can draw an analogy between the asymptotic comparison of two functions \( f \) and \( g \) and the comparison of two real numbers \( a \) and \( b \)
  - \( f(n) = O(g(n)) \) is like \( a \leq b \)
  - \( f(n) = \Omega(g(n)) \) is like \( a \geq b \)
  - \( f(n) = \Theta(g(n)) \) is like \( a = b \)
  - \( f(n) = o(g(n)) \) is like \( a < b \)
  - \( f(n) = \omega(g(n)) \) is like \( a > b \)

- Abuse of notation:
  - \( f(n) = O(g(n)) \) indica que \( f(n) \in O(g(n)) \)
Exercise: Asymptotic Notation

Check whether these statements are true:

a) In the worst case, the insertion sort is $\Theta(n^2)$
b) $2^{2n} = O(2^n)$
c) $2^{n+1} = O(2^n)$
d) $\Theta(n) + \Theta(1) = \Theta(n)$
e) $O(n^2) + O(n^2) = O(n^2)$
f) $O(n) \times O(n) = O(n)$
Summary

• Analyse the running time used by an algorithm via asymptotic analysis
  ▪ asymptotic (O, \( \Omega \), \( \Theta \), o, \( \omega \)) notations
  ▪ use a generic uniprocessor random-access machine
  ▪ Time and space complexity (input size)
  ▪ Best, average and worst case