Lecture 11: Time Efficiency of Algorithms (con’t)

The efficiency of an algorithm may depend on the data.

ex. Do a linear search for the number 18 in array A.

Best case:
Worst case:
Average case:

The efficiency of some algorithms do NOT depend on the data.

To judge efficiency we can time the algorithms or count operations. (Operation =

Examples of counting operations:

<table>
<thead>
<tr>
<th>Number of Operations</th>
<th>Big-O (really Θ)</th>
</tr>
</thead>
</table>

1) for (int j = 0; j < N; j++)
   \( X = X \times X + 2 \times Y; \)

2) for (int j = 0; j < N; j++)
   for (int i = 0; i < N; i++)
   \( T = T + 2 \times R - Y; \)

3) for (int j = 0; j < N; j++)
   for (int i = 0; i <= j; i++)
   if (A[i] > 100)
   \( A[i] = 10; \)

Here the number of operations varies. Why?
It is customary to count the number of comparisons.

<table>
<thead>
<tr>
<th>j =</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Times thru i loop</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Each time through the i loop, ______ comparisons are performed

Total Number of Comparisons =

So Big-O:
4) for (int j = 1; j < N; j++)
   for (int i = 0; i < N -j; i++)
       if (A[i] > A[i + 1])
           { 
               Temp = A[i];
               A[i] = A[i + 1];
               A[i + 1] = Temp;
           }

<table>
<thead>
<tr>
<th>j =</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Times thru i loop</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Total Number of Comparisons \[ \text{Big-O} \]

5) for (int j = 1; j <= N^2; j++)
   for (int i = 1; i <= j; i++)
       if (A[j] == A[i])
           T = T + 3;

<table>
<thead>
<tr>
<th>j =</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Times thru i loop</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Number of Comparisons \[ \text{Big-O} \]
When the Efficiency Depends on the Data—Some Classic Algorithms

**Linear Search**

```plaintext
i ← 0
found ← false
while not found
    if A[i] == Target
else
```

**Best Case:**

**Worst Case:**

**Average Case:**

**Binary Search**

```plaintext
First ← 0
Last ← N – 1
found ← false
while not found and First != Last
    Mid ← (First + Last) / 2
    if A[Mid] == Target
else if A[Mid] > Target
else
    if First == Last
        if A[First] == Target
else
```

**Best Case:**

**Worst Case:**

**Average Case:**
Insertion Sort

\[
\text{for } i = 1 \text{ to } N - 1 \\
\quad \text{Temp } \leftarrow A[i] \\
\quad j \leftarrow i - 1 \\
\quad \text{while } j \geq 0 \text{ and } A[j] > \text{Temp} \\
\quad \quad A[j + 1] = A[j] \quad /\text{slide} \\
\quad \quad j \leftarrow j - 1 \\
\quad A[j+1] \leftarrow \text{Temp} \quad /\text{stuff}
\]

Merge Sort

Best Case

Worst Case

Average Case

Time(N) =

Does the time taken by this algorithm depend on the data?

QuickSort

Pick a Pivot point
move all elements < Pivot to the left and elements >= Pivot to the right
(call these Sets I and II)
move the Pivot to a position separating Sets I and II
QuickSort Set I 
QuickSort Set II

If Sets I and II are equal in size Time(N) =

Does the time taken by this algorithm depend on the data?