Homework 3 Solution

Due Sept 23 before class. Total: 20pts

1. (2pts) Group the following into equivalent Big-Oh functions:
   \( x^2, x, x^2 + x, x^2 - x, \text{ and } \frac{x^3}{(x - 1)} \)

Sol:

- Group one: \( x \)
- Group two: \( x^2, x^2 + x, x^2 - x, \text{ and } \frac{x^3}{(x - 1)} \). Note the denominator in \( \frac{x^3}{(x - 1)} \) is \( O(x) \), and the nominator is \( O(x^3) \). So the expression is in \( O(x^2) \).

2. (3pts) Programs A and B are analyzed and are found to have worst-case running times no greater than \( 150N \log N \) and \( N^2 \), respectively. Answer the following questions, if possible.

   (a) Which program has the better guarantee on the running time for large values of \( N \) (\( N > 10000 \))?

Sol:

we can select \( N = 10^6 \), and plug this value to \( 150N \log N \) and \( N^2 \) respectively. \( 150N \log N \) gets \( 3 \times 10^9 \) and \( N^2 \) gets \( 10^{12} \). \( 150N \log N \) requires shorter execution time. So \( 150N \log N \) has better guarantee on the running time.

   (b) Which program has the better guarantee on the running time for small values of \( N \) (\( N < 100 \))?

Sol:

we can select \( N = 10 \), and plug this value to \( 150N \log N \) and \( N^2 \) respectively. \( 150N \log N \) gets 4950 and \( N^2 \) gets 100. \( N^2 \) requires shorter execution time. So \( N^2 \) has better guarantee on the running time.

   (c) Can program B run faster than program A on all possible inputs?

Sol:
NO. B only runs faster than program A on smaller inputs. It runs much slower than A on large inputs, the cases we care most in algorithm analysis.

3. (4pts) Give a Big-Oh analysis of the running time of for each of the algorithms in figure 3 and figure 4 respectively.

**Figure 3: algorithm 1**

```
ALGORITHM MaxElement(A[0..n−1])
//Determines the value of the largest element in a given array
//Input: An array A[0..n−1] of real numbers
//Output: The value of the largest element in A
maxval ← A[0]
for i ← 1 to n−1 do
  if A[i] > maxval
    maxval ← A[i]
return maxval
```

**Sol:**

The basic operation is comparison performed by statement \( A[i] > \text{maxval} \).

The number of basic operations \( C_{op} \) is indicated in the for loop condition. It is \( n−1 \) from loop index 1 to loop index \( n−1 \).

The execution time \( T = C_{op} \times t_{op} = n \times t_{op} \in O(n) \) as \( t_{op} \) is a constant on a computer.

**Figure 4: algorithm II**

```
ALGORITHM UniqueElements(A[0..n−1])
//Determines whether all the elements in a given array are distinct
//Input: An array A[0..n−1]
//Output: Returns “true” if all the elements in A are distinct
//        and “false” otherwise
for i ← 0 to n−2 do
  for j ← i+1 to n−1 do
return true
```

**Sol:**

The basic operation is comparison performed by statement \( A[i] = A[j] \).
The basic operation is performed \( n - 1 \) times (inner index from 1 to \( n - 1 \)) when \( i = 0 \), \( n - 2 \) times (inner index from 2 to \( n - 1 \)) when \( i = 1 \), \( n - 3 \) times (inner index from 3 to \( n - 1 \)) when \( i = 2 \), and so on. It is performed 1 time when \( i = n - 2 \).

So totally the basic operation is performed \( (n - 1) + (n - 2) + \ldots + 2 + 1 = n(n - 1)/2 \).

The execution time \( T = C_{op} \times t_{op} = n(n - 1)/2 \times t_{op} \in O(n^2) \) as \( t_{op} \) is a constant on a computer.

4. (3pts) Rank the following functions by increasing order of growth. All the logs are in base 2.

\( N, \sqrt{N}, N^{1.5}, N^2, N \log N, N(\log N)^2, N \log N^2, 2/N, 2^N, 2^{N/2}, 37, N^2 \log N, N^3 \). Indicate which functions grow at the same rate.

Sol: The growth rate in increasing order:

\( 2/N, 37, \sqrt{N}, N, N \log N, N \log N^2, N(\log N)^2, N^{1.5}, N^2, N^2 \log N, N^3, 2^{N/2}, 2^N \).

\( N \log N \) and \( N \log N^2 \) grow at the same rate.

5. (4pts) An algorithm takes 0.5ms for input size 100. How long will it take for input size 500 (assuming that low-order terms are negligible) if the running time is

(a) linear

Sol:

<table>
<thead>
<tr>
<th>input size</th>
<th>basic operation counts</th>
<th>execution time</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>100</td>
<td>( 0.5 , \text{ms} = 100 \times t_{op} )</td>
</tr>
<tr>
<td>500</td>
<td>500</td>
<td>( x , \text{ms} = 500 \times t_{op} )</td>
</tr>
</tbody>
</table>

\( x = 2.5 \, \text{ms} \)

(b) \( O(N \log N) \)

Sol:

<table>
<thead>
<tr>
<th>input size</th>
<th>basic operation counts</th>
<th>execution time</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>( 100 \log 100 )</td>
<td>( 0.5 , \text{ms} = 100 \log 100 \times t_{op} )</td>
</tr>
<tr>
<td>500</td>
<td>( 500 \log 500 )</td>
<td>( x , \text{ms} = 500 \log 500 \times t_{op} )</td>
</tr>
</tbody>
</table>

\( x = 3.3 \, \text{ms} \)
6. (4pts) An algorithm takes 0.5ms for input size 100. How large a problem can be solved in 1 minute (assuming that low-order terms are negligible) if the running time is

(a) linear

Sol:

<table>
<thead>
<tr>
<th>input size</th>
<th>basic operation counts</th>
<th>execution time</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>100</td>
<td>$0.5 , ms = 100 \times t_{op}$</td>
</tr>
<tr>
<td>$x$</td>
<td>$x$</td>
<td>$1 , min = x \times t_{op}$</td>
</tr>
</tbody>
</table>

$x = 12 \times 10^6$

(b) cubic

Sol:

<table>
<thead>
<tr>
<th>input size</th>
<th>basic operation counts</th>
<th>execution time</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>$100^3$</td>
<td>$0.5 , ms = 100^3 \times t_{op}$</td>
</tr>
<tr>
<td>$x$</td>
<td>$x^3$</td>
<td>$1 , min = x^3 \times t_{op}$</td>
</tr>
</tbody>
</table>

$x = 4900$