

Homework 1

SiLin Huang

UNI: sh3334

written

Due: 4:00pm on Friday, September 23, 2016

This homework has two parts: a written section and a programming section.

Written (50 pts)

For the written section of this assignment, type up your answers and submit a computer based document to us. You can submit MS Word doc files, pdf files, or txt files.

1. (10 pts) Weiss, Exercise 2.1

Page 50

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

a) Increasing order of function by growth rate:

$2/N < 37 < \sqrt{N} < N < N \log \log N < N \log N < N \log(N^2) < N \log^2 N < N^{1.5} < N^2 < N^2 \log N < N^3 < 2^{N/2} < 2^N$

b) Functions that grow at the same rate:

i) The time complexity of the function $N \log N$ is $O(N \log N)$.

The time complexity of the function $N \log(N^2)$ is:

$$N \log(N^2) = N (2 \log N)$$

$$= 2 N \log N$$

$$= O(N \log N)$$

$$\text{growth}(N \log(N^2)) = \text{growth}(N \log N)$$

Therefore, $N \log N$ and $N \log(N^2)$ have the same growth rate $O(N \log N)$.

2. (10 pts) Weiss, Exercise 2.6

Page 50

In a recent court case, a judge cited a city for contempt and ordered a fine of \$2 for the first day. Each subsequent day, until the city followed the judge's order, the fine was squared (that is, the fine progressed as follows: \$2, \$4, \$16, \$256, \$65, 536, . . .).

a. What would be the fine on day N ?

Day	Fine (\$)	Pattern: 2^n
1	2	$2^1 = 2^{2^0}$
2	4	$2^2 = 2^{2^1}$
3	16	$2^4 = 2^{2^2}$
4	256	$2^8 = 2^{2^3}$
5	65536	$2^{16} = 2^{2^4}$
...
N	...	$2^{2^{\wedge(N-1)}}$

Therefore, on day N , the fine would be $2^{2^{\wedge(N-1)}}$ dollars.

b. How many days would it take the fine to reach D dollars? (A Big-Oh answer will do.)

$$\begin{aligned}D &= 2^{2^{(N-1)}} \\ \log_2 D &= \log_2 2^{2^{(N-1)}} \\ \log D &= 2^{(N-1)} \\ \log (\log D) &= \log (2^{(N-1)}) \\ \log (\log D) &= N-1 \\ \log (\log D) + 1 &= N\end{aligned}$$

Therefore, $N = \log \log D + 1$ and the Big-Oh notation for $2^{2^{(N-1)}}$ is $O(\log \log N)$.

3. (10 pts) Give an analysis of the Big-Oh running time for each of the following program fragments:

a.

```
int sum = 0;
for ( int i = 0; i < 23; i ++ )
    for ( int j = 0; j < n ; j ++ )
        sum = sum + 1;
```

The first for loop runs 23-number times. The inner for loop runs for each outer for loop iteration. Then, the for loop runs n-number of times... i.e 1 -> $O(n)$, 2 -> $O(n)$...25 -> $O(n)$. Therefore, the total complexity of the given code is : $O(23 * n)$. After eliminating the constant value, the complexity of the given code is $O(n)$.

$O(N)$

b.

```
int sum = 0;
for ( int i = 0; i < n ; i ++ )
    for ( int k = i ; k < n ; k ++ )
        sum = sum + 1;
```

The first for loop runs n-number times. The inner for loop runs for each outer for loop iteration. The inner for loop runs n-number of times...i.e 1 -> $O(n)$, 2 -> $O(n)$. Therefore, the complexity of the given code is $O(n^2)$.

$O(N^2)$

c.

```
public int foo(int n, int k) {
    if(n<=k)
        return 1;
    else
        return foo(n/k,k) + 1;
}
```

The recursive relation of n compares with k every time. Then, the second iteration n is divided by k times. The value is divided by k . Therefore, the recursive function will call n number of times. The time complexity is $O(\log N)$.

$O(\log N)$

4. (10 pts) Weiss, Exercise 2.11

Page 52

An algorithm takes 0.5 ms for input size 100. How long will it take for input size 500 if the running time is the following (assume low-order terms are negligible):

a. linear

The run time of linear function: $T(N) = O(N)$

$$N_1 = 100$$

$$T(N_1) = 0.5 \text{ ms}$$

$$N_2 = 500$$

$$\begin{aligned} T(N_2) &= (N_2 / N_1) \times T(N_1) \\ &= (500 / 100) \times 0.5 \text{ ms} \\ &= 5 \times 0.5 \text{ ms} \\ &= \mathbf{2.5 \text{ ms}} \end{aligned}$$

Therefore, the time it takes to complete an algorithm of size 500 is 2.5 ms.

b. $O(N \log N)$

The running time of $N \log N$ function is $T(N) = O(N \log N)$.

$$N_1 = 100$$

$$T(N_1) = 0.5 \text{ ms}$$

$$N_2 = 500$$

$$\begin{aligned} T(N_2) &= ((N_2 \log N_2) / (N_1 \log N_1)) \times T(N_1) \\ &= ((500 \times \log_2 500) / (100 \times \log_2 100)) \times 0.5 \\ &= 5 \times ((\log_2 500) / (\log_2 100)) \times 0.5 \\ &= 2.5 \times ((\log_2 500) / (\log_2 10^2)) \\ &= 2.5 ((\log_2 500) / (2 \times \log_2 10)) \\ &= (2.5 / 2) \times (\log_{10} 500) & \text{** } (\log_c(a) / \log_c(b)) = \log_b a \\ &= 1.25 \times (\log_{10} 500) \\ &= 1.25 \times 2.6989 \\ &= \mathbf{3.3737 \text{ ms}} \end{aligned}$$

Therefore, the time it takes to complete an algorithm of size 500 is 3.374 ms.

c. quadratic

The running time of quadratic function is $T(N) = O(N^2)$.

$$N_1 = 100$$

$$T(N_1) = 0.5 \text{ ms}$$

$$N_2 = 500$$

$$\begin{aligned} T(N_2) &= ((N_2^2) / (N_1^2)) \times T(N_1) \\ &= ((500^2) / (100^2)) \times 0.5 \text{ ms} \\ &= 25 \times 0.5 \text{ ms} \\ &= \mathbf{12.5 \text{ ms}} \end{aligned}$$

Therefore, the time it takes to complete an algorithm of size 500 is 12.5 ms.

d. cubic

The running time of quadratic function is $T(N) = O(N^3)$.

$$N_1 = 100$$

$$T(N_1) = 0.5 \text{ ms}$$

$$N_2 = 500$$

$$\begin{aligned} T(N_2) &= ((N_2^3) / (N_1^3)) \times T(N_1) \\ &= ((500^3) / (100^3)) \times 0.5 \text{ ms} \\ &= 125 \times 0.5 \text{ ms} \\ &= \mathbf{62.5 \text{ ms}} \end{aligned}$$

Therefore, the time it takes to complete an algorithm of size 500 is 62.5 ms.

5. (10 pts) Weiss, Exercise 2.15

Page 52

Give an efficient algorithm to determine if there exists an integer i such that $A_i = i$ in an array of integers $A_1 < A_2 < A_3 < \dots < A_N$. What is the running time of your algorithm?

Solution:

Let c be an integer between 1 and N .

If $A_c < c$, then $A_i < i$, for $i = 1$ to $(c - 1)$.

If $A_c > c$, then $A_i > i$, for $i = (c + 1)$ to N

Algorithm:

- Check if the middle element satisfies $A_i = i$, and if it does the answer is yes.
- If $A_i < i$ we can apply the same strategy to the subarray to the right of the middle element.
- If $A_i > i$ we can apply the same strategy to the subarray to the left of the middle element.

left = 0, right = a.length

```
if (a[0] > 1)
    return -1;
```

```
while ( left <= right ) {

    int mid = (left + right)/2;
```

```
if a[mid] = mid + 1
    return (mid+1);

else if (a[mid] > (mid+1))
    right = mid - 1;

else (a[mid] < (mid+1))
    left = mid + 1;
}
return -1;
```

Runtime Analysis:

**The problem is halved in size at each step, for a constant amount of work.
Therefore, the run time of the algorithm is $O(\log N)$.**