CSE 503 – Homework 1 Solution

Truong Nguyen
University of Louisville
Summer 2025

May 14, 2025

Required problems:

1. P47: 1.7, 1.10
2. P71,72,74: 2.1, 2.3, 2.4, 2.6, 2.7, 2.22

1 Problem 1.7, P47

Problem statement: Prove the following formulas:

a $log(X) < X$ for all X

b $log(A^B) = Blog(A)$

Problem solution: We remark that X can not be negative otherwise, $log(X)$ is undefined. So we can only consider the case $X > 0$.

Let $f(X) = X - log(X)$ where we assume, without loss of generality (WLOG), that logarithmic is in base 10. Derivative of $f(X)$ is

$$f'(X) = 1 - \frac{1}{X}.$$

Setting the derivative to zero, we have

$$f'(X) = 0 \iff 1 - \frac{1}{X} = 0 \iff X = 1$$

Next, let us consider the three cases for X:

* When $0 < X < 1$, we have $log(X) < 0$, i.e., taking logarithmic of a fraction less than 1 gives a negative number by definition. This implies $f(X) = X - log(X) > 0$. Hence, $X > log(X)$ for $0 < X < 1$.
* When $X = 1$, we have $f(X = 1) = 1 - log(1) = 1 - 0 = 1 > 0$. This also implies that $X > log(X)$ when $X = 1$.

* When $X \geq 1$, we can see that derivative of $f(X)$ equals $f'(X) = X - \frac{1}{X} \geq 0$ since the fraction $\frac{1}{X} \leq 1$. Therefore, when $X \geq 1$, $f(X)$ is an increasing function. This is equivalent to say that for two numbers $X = x_1$ and $X = x_2$ such that $x_1 > x_2 \geq 1$ then we have $f(x_1) - f(x_2) > 0$. If we choose x_1 to be arbitrarily greater than 1 and $x_2 = 1$ then we have

$$0 < f(x_1) - f(x_2) = x_1 - log(x_1) - (1 - log(1)) = x_1 - log(x_1) - 1$$

$$x_1 - log(x_1) > 1$$

Therefore, combining the results of the above three cases, we have $log(X) < X$ for all $X > 0$.

2 Problem 1.10, P47

Problem statement: What is $2^{100}\,(mod\,5)$?
Problem solution: Let us start with small powers of 2:

$$\begin{aligned}
&2^1 \equiv 2\ (mod\,5) \text{ since 5 divides 2-2}\\
&2^2 \equiv 4\ (mod\,5) \text{ since 5 divides 4-4}\\
&2^3 \equiv 3\ (mod\,5) \text{ since 5 divides 8-3}\\
&2^4 \equiv 1\ (mod\,5) \text{ since 5 divides 16-1}\\
&2^5 \equiv 2\ (mod\,5) \text{ since 5 divides 32-2}\\
&2^6 \equiv 4\ (mod\,5) \text{ since 5 divides 64-4}\\
&2^7 \equiv 3\ (mod\,5) \text{ since 5 divides 128-3}\\
&2^8 \equiv 1\ (mod\,5) \text{ since 5 divides 256-1}\\
&\ldots
\end{aligned}$$

We see that the cycle repeats for every 4 units increase in the power of the number 2. The number 100 is divisible by 4; hence $2^{100} \equiv 1\ (mod\,5)$

3 Problem 2.1, P71

Problem statement: Order the following functions by growth rate:

$$N, \sqrt{N}, N^{1.5}, N^2, NlogN, NloglogN, Nlog^2N, NlogN^2, 2/N, 2^N, 2^{N/2}, 37, N^2logN, N^3\,.$$

Indicate which functions grow at the same rate.
Problem solution: Before ranking them by growth-rate, let us rewrite a few of them as follows and then organize them into the following groups:

Group 1:

$$\begin{aligned} N &= N^1 \\ \sqrt{N} &= N^{1/2} \\ N^{1.5} &= N^{1.5} \\ N^2 &= N^2 \\ N^3 &= N^3 \\ \frac{2}{N} &= 2N^{-1} \\ 37 &= 37N^0 \end{aligned}$$

Group 2:

$$\begin{aligned} NlogN &= Nlog(N) \\ NloglogN &= Nlog(log(N)) \\ Nlog^2N &= N(log(N))^2 \\ NlogN^2 &= Nlog(N^2) = 2Nlog(N) \\ N^2logN &= N^2log(N) \end{aligned}$$

Group 3:

$$\begin{aligned} 2^N &= 2^N \\ 2^{N/2} &= (2^{0.5})^N = (\sqrt{2})^N \end{aligned}$$

First, we will order in Group 1 by growth rate. From left to right, we order every term in Group 1 where growth-rate increase:

$$2N^{-1},\ 37N^0,\ N^{1/2},\ N^1,\ N^{1.5},\ N^2,\ N^3$$

The above ordering is easy and straightforward to see since we know that the higher the power of N, the better / faster the growth rate.

Next, we will do the same for Group 2:

$$Nlog(log(N)),\ Nlog(N),\ 2Nlog(N),\ N(log(N))^2,\ N^2log(N)$$

To get above ordering, we first compare $NloglogN$ versus $NlogN$. From problem 1, we know that $logN$ approaches infinity slower than N for all $N > 0$. Thus, as $N \to \infty$, N has a higher growth rate than $logN$. Thus, $NlogN$ has a higher growth rate than $NloglogN$. Obviously, $2NlogN$ has a slightly higher growth rate than $NlogN$ due to the factor of 2.0. Next, we will compare $N(logN)^2$ versus $2NlogN$. It is straightforward to see that $logN > 2.0$ for large values of N; therefore, $N(logN)^2$ will outperform growth $2NlogN$ as $N \to \infty$. Using the same reasoning that as $N \to \infty$, N has higher growth rate than $logN$, we can conclude N^2logN out growth $N(logN)^2$. This concludes the ordering for Group 2.

For Group 3, it is also straightforward to conclude that 2^N will outperform growth $2^{N/2}$ because the base $2 > \sqrt{2}$

Now, we will merge the ranking/ordering of Group 1 and 2 to have the following ordering (from left to right yield increasing growth-rate):

$2N^{-1}$, $37N^0$, $N^{1/2}$, N, $Nlog(log(N))$, $Nlog(N)$, $2Nlog(N)$, $N(log(N))^2$, $N^{1.5}$, N^2, N^2logN, N^3

To get this ordering, we note that $N^{0.5}$ will approach infinity at a much faster rate than the logarithmic functions. This implies $N^{1.5} = N \times N^{0.5}$ will grow faster than the $N \times (logN)^2$ terms. In particular, $(logN)^2$ still has a slower growth rate than $\sqrt{N}$.

Finally, the final ordering is:

$\frac{2}{N}$, 37, $N^{1/2}$, N, $NloglogN$, $NlogN$, $2NlogN$, $N(logN)^2$, $N^{1.5}$, N^2, N^2logN, N^3, $\sqrt{2}^N$, 2^N

because $\sqrt{2}^N$ will out-growth any terms in Group 1 & Group 2

Terms that have same growth-rate are:

1. $NlogN$ and $2NlogN$. The factor of 2.0 won't make a difference when $N \to \infty$
2. 2^N and $\sqrt{2}^N$ because taking the logarithms of these 2 terms, we have: $Nlog(2)$ and $\frac{N}{2}log(2)$. The factor of $\frac{1}{2}$ won't make a difference when $N \to \infty$

4 Problem 2.3, P71

Problem statement: Which function grows faster: $NlogN$ or $N^{1+\epsilon/\sqrt{logN}}$ where $\epsilon > 0$.
Problem solution: Let us define:

$$\begin{aligned} f(N) &= NlogN \\ g(N) &= N^{1+\epsilon/\sqrt{logN}} = N \times N^{\epsilon/\sqrt{logN}} \end{aligned}$$

Taking the ratio $f(N)/g(N)$, we have:

$$\begin{aligned} \frac{f(N)}{g(N)} &= \frac{NlogN}{N \times N^{\epsilon/\sqrt{logN}}} = \frac{logN}{N^{\epsilon/\sqrt{logN}}} \\ &= \frac{logN}{e^{(logN)^{\epsilon/\sqrt{logN}}}} \\ &= \frac{logN}{e^{logN \times \epsilon/\sqrt{logN}}} \\ &= \frac{logN}{e^{\epsilon\sqrt{logN}}} \end{aligned}$$

So far, we've learned that exp() function always out grow log() function i.e., exp() function should explode faster than log() as $N \to \infty$. Hence, the above ratio for $\frac{f(N)}{g(N)} \to 0$ as $N \to \infty$. Therefore, $N^{1+\epsilon/\sqrt{logN}}$ grows faster than $NlogN$.

5 Problem 2.4, P71

Problem statement: Prove that for any constant k, $log^k(N) = o(N)$
Problem solution: We are asked to prove that for any constant k, $log^k(N)$ grows much slower than N. To compare growth-rate of 2 monotonically increasing functions, we can compare growth-rate of their logarithms. In particular, let's define:

$$f(N) = log^k(N)\,,$$
$$g(N) = N\,.$$

Taking the log of the 2 functions, we have:

$$log\,f(N) = k\,log(log(N))\,,$$
$$log\,g(N) = logN\,.$$

From problem 2.1, we have learned that $loglogN$ has a slower growth-rate than $logN$ and $k = O(1)$ since k is a fixed constant. This implies $logf(N) = o(log(N))$, i.e., $log^k(N) = o(N)$.

6 Problem 2.6, P71

Problem statement: 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, \$65536, etc.

a. What would be the fine on day N?

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

Problem solution: It looks like we are going to have the following relationship between the day number and fine amount:

Day	Fine	$R(t) = F(t)/F(t-1)$
1	2	UNK
2	4	2
3	16	4
4	256	16
5	65536	256
...	...	...

In the above table, we also define $R(t) = F(t)/F(t-1)$ where $F(t)$ is the fine amount of day t. So looks like the pattern is:

$$F(1) = 2^1$$
$$F(2) = 2^2$$
$$F(3) = 2^4$$
$$F(4) = 2^8$$
$$F(5) = 2^{16}$$
$$F(N) = ...$$

We see the pattern, for the powers of 2, here:

$$\begin{aligned} 1 &= 2^{1-1} = 2^0 \\ 2 &= 2^{2-1} \\ 4 &= 2^{3-1} \\ 8 &= 2^{4-1} \\ 16 &= 2^{5-1} \end{aligned}$$

So

$$F(N) = 2^{2^{N-1}}$$

To reach D dollar, we want to find a day t such that $F(t) = 2^{2^{t-1}} = D$. Taking log base 2 to both side, we have

$$2^{t-1} = log_2 D$$

Taking log base 2 again to both side:

$$\begin{aligned} t - 1 &= log_2(log_2(D)) \\ t &= log_2(log_2(D)) + 1 \end{aligned}$$

We would need to take approximately $log_2(log_2(D)) + 1$ days for us to reach D dollars.

7 Problem 2.7, P71 & P72

Problem statement: For each of the following six program fragments:

a. Give an analysis of the running time (Big-Oh will do).

b. Implement the code in the language of your choice, and give the running time

c. Compare your analysis with the actual running times.

Problem solution: For the six programs, please refer to pseudo-code given in Page 72 of the Data Structure and Algorithm Analysis textbook.

For part (a.)

1. Program #1: run time $O(n)$
2. Program #2: run time $O(n^2)$
3. Program #3: run time $O(n^3)$
4. Program #4: run time $O(n^2)$ but more like $0.5n^2$
5. Program #5: run time $O(n^5)$ since it takes i-loop n operations, and it takes j-loop n^2 operations and k-loop n^2 operations.

6. Program #6: run time $O(n^4)$. To see this, we first consider that without if-statement, then Program 6 is exactly like Program 5 so without if-statement, its runtime should be $O(n^5)$. However, because of the if-statement, the k loop is only executed when i divides j. This only happens when $j = k * i$ for some integer k. This means, sum is only updated when:

   - $i = 1, 2, 3, 4, ..., n$ and
   - $j = i, 2i, 3i, 4i, ..., i * i$ and
   - $k = 1, 2, 3, ..., j \approx O(n^2)$

   Hence, overall order of operation for Program #6 is $O(n^4)$

For parts (b.) and (c.) Please see attached Python code that generates the following figures of actual run-times for Programs #1 to #6. What I did was: for each Program #2 to #6, since the theoretical orders for run-time are polynomial (non-linear), we use N-values from 100 to 200 with increment of 20. However, for Program #1, since order is linear i.e., $O(N)$, we know it will run pretty fast; hence, I chose larger N values such as $N = 1000, 3000, 5000, 7000, 9000$. Note that, if we choose N large, then Program # 2 to # 6 can take a while to run especially #5 and #6.

In the following figures (1), (2), (3), (4), (5) and (6), I present the log-log plots where the x-axis is for different values of N and the y-axis is for recorded runtime (in micro-seconds). For each program, I also plotted the theoretical runtime orders, given in part (a.) We can see that for each program, the lines are parallel e.g., theoretical and actual lines have the same slope for each program. This indicates that the theoretical and actual running times agree. I.e., they are verified to be the same orders

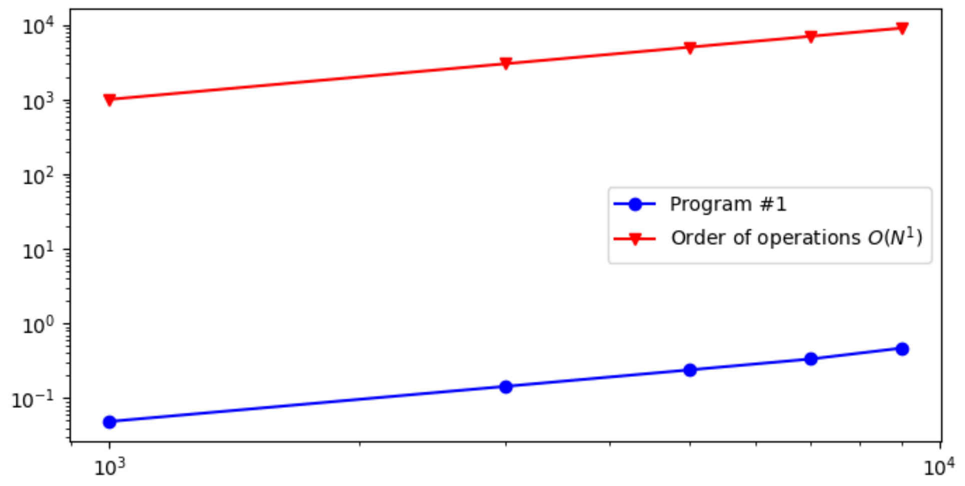

Figure 1: Program #1: Theoretical vs. Actual run times

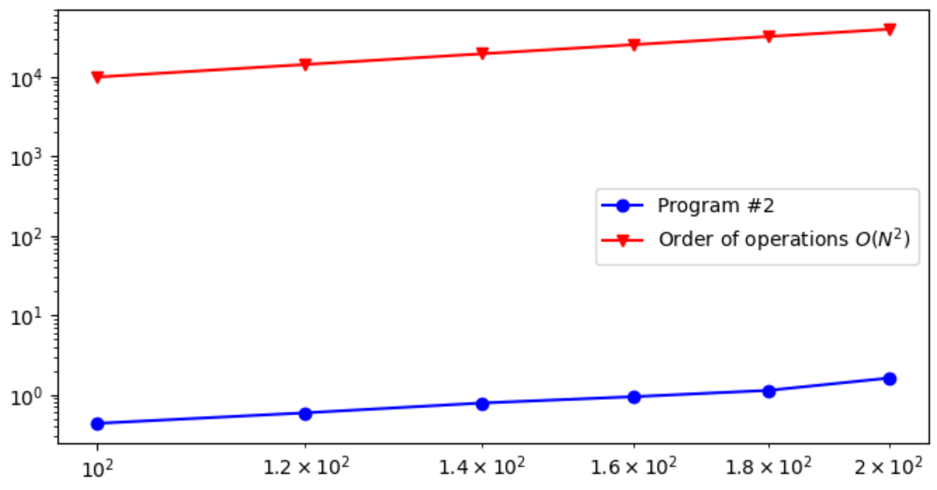

Figure 2: Program #2: Theoretical vs. Actual run times

8 Problem 2.22, P74

Problem statement: Show that X^{62} can be done in 8 multiplications
Problem solution:

1. First we compute $X^2 = X \times X$. This is one multiplication.
2. Next, we compute $X^4 = X^2 \times X^2$.
3. Next, we compute $X^6 = X^4 \times X^2$.
4. Then we compute $X^{12} = X^6 \times X^6$.
5. Compute $X^{24} = X^{12} \times X^{12}$.
6. Evaluate $X^{48} = X^{24} \times X^{24}$. So far, the number of multiplications is 6.
7. Finally, we can get $X^{62} = X^{48} \times X^{12} \times X^2$. Number of multiplication now is exactly 8.

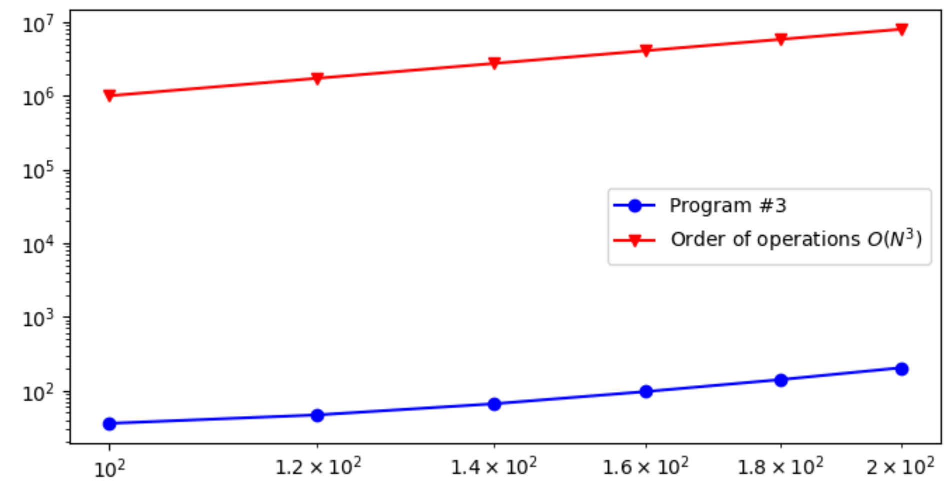

Figure 3: Program #2: Theoretical vs. Actual run times

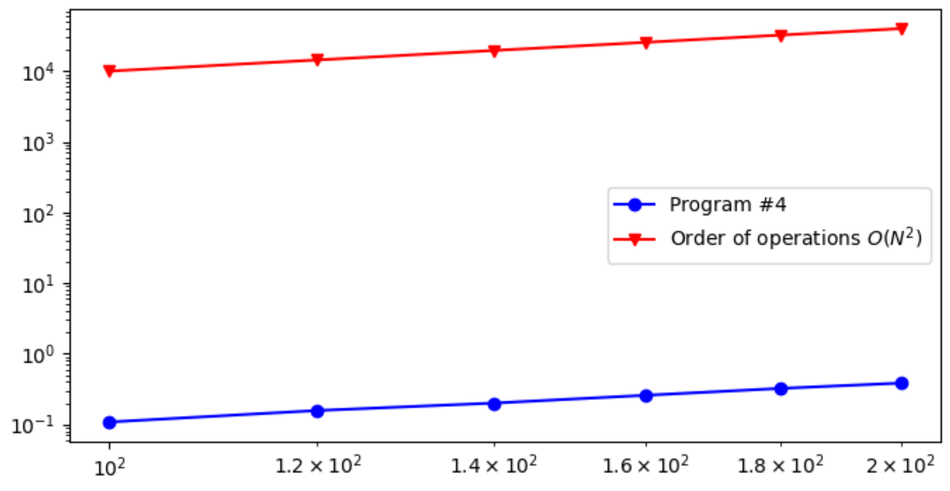

Figure 4: Program #2: Theoretical vs. Actual run times

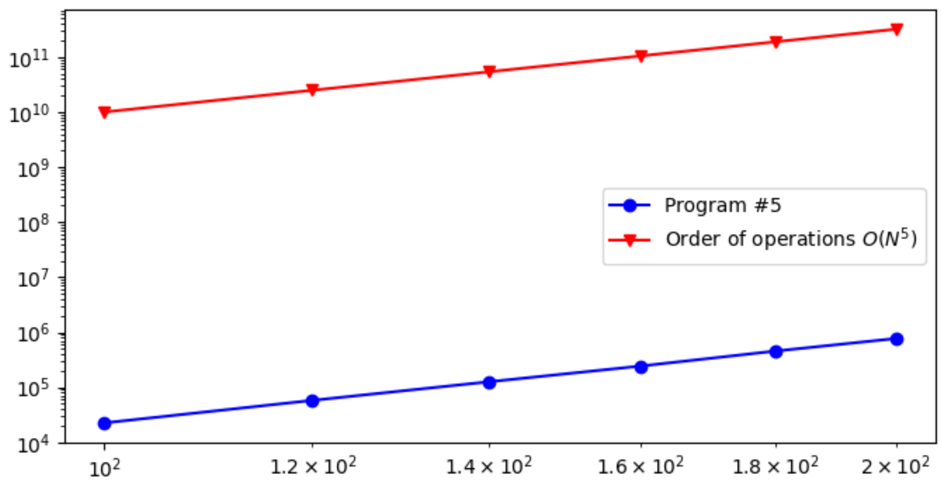

Figure 5: Program #2: Theoretical vs. Actual run times

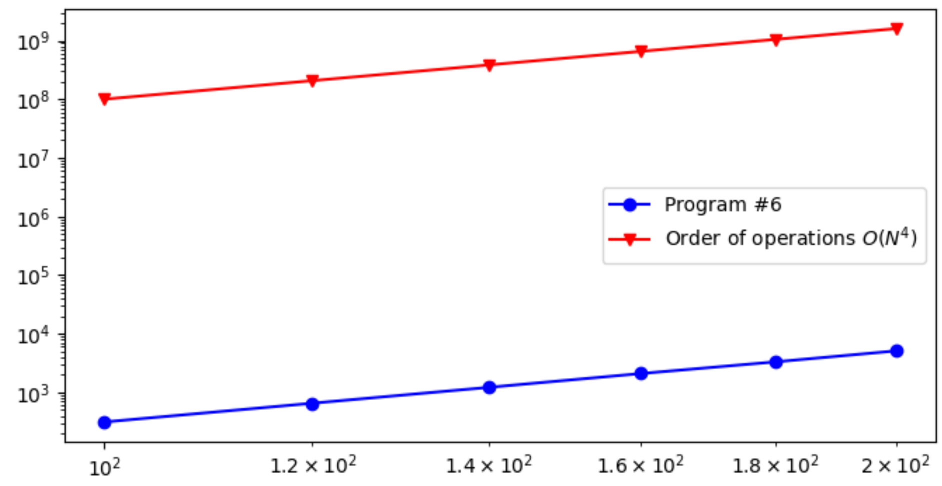

Figure 6: Program #2: Theoretical vs. Actual run times