CSE 015: Discrete Mathematics
Homework 7

Fall 2021
Provided Solution

1 Asymptotic Notation

To answer these questions we have to recall that in asymptotic notation we can ignore lower order terms and constants multiplying the highest-degree term. Then, we can simply consider which function grows faster as n tends to infinity.

a) in this case $f(n)$ is a first order polynomial, i.e., $f(n)$ asymptotically grows as n, and so $f(n)$ is $O(n^2)$

b) $f(n)$ asymptotically grows as $n \log n$, and so $f(n)$ is $O(n^2)$. To this end, recall that $\log n$ grows slower than n.

c) in this case $f(n)$ is a quadratic polynomial, i.e., $f(n)$ asymptotically grows as n^2. So $f(n)$ is $O(n^2)$. (if you are taken aback by this fact, revise the definition of big-O and see that you can indeed find suitable constants to satisfy the definition.)

d) in this case $f(n)$ asymptotically grows as $n^{1/2}$ and so $f(n)$ is $O(n^2)$.

e) in this case $f(n)$ asymptotically grows as n^3 and so $f(n)$ is not $O(n^2)$.

2 Asymptotic Notation

The question asks to sort the functions so that each function big-O of the next. Therefore if you just type the correct order is fine. Here we provide a bit of extra explanation. $\log n$ grows slower than any polynomial in n. Polynomials grow asymptotically slower than exponentials of the type a^n with $a > 1$. Hence the sorted sequence is the following:

$$\log n \quad \sqrt{n} \quad n \quad n \log n \quad n^2 \quad n^2 \log n \quad n^4 \quad 2^n \quad 3^n$$

3 Asymptotic Growth

To find the correct answer you can use any computational method or numerical tool – the method is not important as long as you find the correct answer.
We start finding out the largest size of the input that can be processed by the three algorithms in one hour of computational time on computer A. As specified in the problem statement, in one hour computer A can execute $3.6 \cdot 10^9$ operations. For each algorithm we need to find the largest n that does not exceed such number. For algorithm 1, the answer was already given and is 26820. Indeed, note that $f_1(26829) = 3599888403 < 3.6 \cdot 10^9$, while $f_1(26830) = 3600156732 > 3.6 \cdot 10^9$. The answer for f_2 can be easily found solving the linear inequality $10n + 4 < 3.6 \cdot 10^9$ with $n \in \mathbb{N}$. This gives $n = 359999999$. For

the third algorithm we have to solve the inequality $2^n < 3.6 \cdot 10^9$ with $n \in \mathbb{N}$. This is immediately solved taking the $\log_2$ of both sides. This gives the answer $n = 31$.

Next we find the largest size of the input that can be processed by the three algorithms in one hour of time on computer B. As stated in the problem, B is 100 times faster than A and can therefore execute $3.6 \cdot 10^{11}$ operations in one hour. We solve again the three inequalities with this new bound. For f_1 we have to solve the inequality $5n^2 + 34n + 12 < 3.6 \cdot 10^{11}$ with $n \in \mathbb{N}$. The solution is 268324 (note that the second degree polynomial has two roots, but we only consider the positive one because the solution must be a natural number, i.e., non negative). For f_2 the solution is 35999999999 and can be found as above. Similarly, for f_3 we apply the same reasoning as above and obtain $n = 38$.

What do we observe? Computer B is 100 times faster than computer A, but the size of the largest problem we can solve has not equally increased for the three algorithms. For the algorithm 1, which is $O(n^2)$, in one hour of computational time with computer B we are able to solve inputs that are roughly 10 times larger than what we can solve with one hour of computational time on computer A (observe that $268324/26830 \approx 10$). For algorithm 2, which is $O(n)$ in one hour of computational time with computer B we are able to solve problem instances that are about 100 times larger than what we can solve with one hour of computational time on computer A. But for algorithm 3, which is $O(2^n)$ the situation is very different. Even though computer B is 100 times faster than computer A, we are only able to solve problem instances that are slightly bigger, i.e., we can go from size 31 to size 38. This result shows why when we study algorithms we avoid using algorithms with exponential complexity. When dealing with exponential algorithms, large gains in computational power lead to only marginal improvements in the size of the problems we can solve!