Aayush Kapar
CSE2010
Homework 2 written

1. R-4.8, p. 182, provide explanation
Order the following functions by asymptotic growth rate.
4 n log(n)+2n 2^{10} $2^{\log(n)}$ 3 n+100 log(n) 4n 2^n n^2 +10n n^3 n log(n)

(BEST)
2^{10} //constant
3 n+100 log(n)
//Big O - n Same as 4n. Both are linear. log(n) gets dropped since its a lower rate
4n
//Linear graph. It's a constant growth graph.
n log(n)
//linearithmic graph. Combines linear and log(n) graphs.
4n log(n) + 2n
//Big O = n log(n). It is the same time complexity as the one above. In comparison, just because it has more constants, it can be worse.
n^2 +10n
//Big O = n^2. Its quadratic graph. The growth rate is bigger than ones above.
n^3
//Higher growth rate than quadratic.
$2^{\log(n)}$
//Exponential growth.
2^n
(WORST)

2. R-4.13, p. 182, provide explanation
Give a big-Oh characterization, in terms of n, of the running time of the example5 method shown in Code Fragment 4.12.

There are 3 for loops nested within each other. Lets label the loops, A, B, and C. Loop C is within loop B and look B is within loop A. There is a comparison that runs inside loop C. This comparisons runs n * j(The count of times Loop B has already ran). This is linear so it runs n times. Loop B runs n^2 times. n * n^2 = n^3
(line 39) A = n
(line 41) B = n^2
(line 42) C = n*(1+2+3…(j))

Conditional if statement = n^2 * (n*(1+2+3…(j)))
Big O = n^3

3. R-4.19, p. 184
Show that O(max{ f(n),g(n)}) = O(f(n) +g(n)).

C * max{ f(n),g(n)} = f(n) +g(n) when $n >= n_0$

f(n) <= max{ f(n),g(n)} and g(n) <= max{ f(n),g(n)} when $n >= n_0$

Therefore

f(n) + g(n) <= 2 * max{ f(n),g(n)} when $n >= n_0$

So C = 2 $n >= n_0$

So f(n) + g(n) <= max{ f(n),g(n)} (remove constants)

4. Consider $f(n) = 4n^2 + 3n-1$, mathematically show that f(n) is $O(n^2)$, $\Omega(n^2)$, and $\Theta(n^2)$.

1) $O(n^2) = 4n^2 + 3n-1 >= cn^2$ C > 0 and $n >= n_0$ $n_0 = 1$ C = 1

2) $3n-1 >= cn^2 - 4n^2$

3) $3n-1 >= n^2(c - 4)$ 4) $3n-1 >= n^2(c - 4)$ 5) $\frac{f(n)}{g(n)} <= \frac{C\,g(n)}{g(n)}$

6) $\frac{3n-1}{n^2} >= \frac{n^2(c-4)}{n^2}$ 7) $\frac{3n-1}{n^2} >= c - 4$

Input n $3 - 1 + 4 >= c$ $3 - 1 + 4 >= 6$ so C = 6

C 6 <= 1

So for all N when greater then n_0, f(n) <= Og(n)

$O(n^2)$ C = 6 n_0 = 1

$\Omega(n^2)$ C = 1 n_0 = 1

$\theta(n^2)$ C = 6 n_0 = 1

5. For finding an item in a sorted array, consider “ternary search,” which is similar to binary search. It compares array elements at two locations and eliminates 2/3 of the array. To analyze the number of comparisons, the recurrence equations are T(n) = 2 + T(n/3), T(2) = 2, and T(1) = 1, where n is the size of the array. Explain why the equations characterize “tertiary search” and solve for T(n).

T(n) = 2 + T(n/3), T(2) = 2, and T(1) = 1,
T(n) = 2+ (1 + T(n/9))
T(n) = 3+ (1 + T(n/27))
$T(n) = x + T(n/3^x)$ base case $n/2^x = 1$ so $2^x = n$ so $x = \log_3 n$
$T(n) = \log_3 n + T(n/n)$
$T(n) = \log_3 n + 1$
$T(n) = \log_3 n$

6. To analyze the time complexity of the “brute-force” algorithm in the programming part of this assignment, we would like to count the number of all possible schedules.

(a) Explain the number of all possible schedules in terms of n (number of candidate courses) and m (number of time slots per candidate course).

(b) Consider a computer that can process 1 billion schedules per second, n is 10, m is 20, explain the number of hours needed to process all possible schedules.

(c) If we do not want the computer to spend more than 1 minute, explain the largest n the computer can process when m is 20.

We would need to count the total amount of combinations possible. We dont need to care about the order. For example choosing ClassA timeA and ClassB timeB is the same as chooisng ClassB timeB then ClassA timeA. We do have a order of preference, however we can use other algorithms to figure out which schedule fits in the best with our preference.

Total number of combinations = n! / r!(n - r)!
n = number of objects
r = number of samples

For our problem, the total amount of objects to test is:

n = the total amount of courses * m the total amount of times for each course
This would give us the total number of objects we need to consider
r = the total amount of courses since we want to figure out the total amount of combinations with the size of the amount of courses, therefore our sample size

If we had 10 as total number of courses(n) and 20 as total amount of time available per course(m), there will be a total of 200 different objects. That makes our n = 200, and r = 10. There would be a total of 22,451,004,309,013,280 combinations possible. This would take the computer capable of doing 1 billion schedules per seconds a total of 22451004 seconds
which is 374183 minutes
which is 6236 hours
which is 260 days

The largest n the computer can process can be calculated by first looking at the total amount of schedules it can process: 1 billion * 60 = 60 billion per minute. Then we can input it into the equation:
60,000,000,000 = 20 * r! / r! ((20 * r) - r)!. Then we can use algebra to solve this equation

$$\frac{20r!}{r!(20r-r)!} = 20\frac{r!}{r!(20r-r)!} = \frac{20}{(20r-r)!} = \frac{20}{(19r)!}$$

$$60{,}000{,}000{,}000 = \frac{20}{(19r)!}$$

$6 < r < 7$

Total amount of courses is between 6 and 7. Since we cannot look into half a course, the total amount of courses we could search with 20 total amounts of time available per course is 6 courses.