Tom Fenyak

CPSC 3273 Homework Module 03

Exercise 1 (15 points)

Consider two algorithms A1 and A2 that have the running times T1(n) and T2(n), respectively.

T1(n) = 100 nlg(n) and T2(n) = n2.

1. (12 points) Use the definition of O() to show that T1(n) € O(T2(n))
   1. We need to find c and n0 such that

0 <= 100 nlg(n) <= c.n^2

0 <= 100 lg(n) / n <= c // dividing by n^2

If we choose any n value where n >= 1 and a number c0 to fufill the statement above we will can say that 100 nlg(n) € O(n^2). In other words, there will be a value that we can plug into c that will always be larger than 100 lg(n)/n when we reach a certain n.

1. (3 points) Which algorithm should you use? A1 or A2? Justify
   1. I would use algorithm A1 because when you increase n or the sample size the running time will not be as long as A2, making it more efficient than A2.

Exercise 2 (15 points)

Consider two algorithms A1 and A2 that have the running times T1(n) and T2(n), respectively.

T1(n) = n3 + 3n and T2(n) = 50n2.

1. (12 points) Use the definition of Ω() to show that T1(n) € Ω(T2(n))

We need to find c and n0 such that

0 <= c.50n^2 <= n^3 + 3n

0 <= c.50 <= n + 3 / n // dividing by n^2 this will be true for all n >= n0

We need to choose a c0 such that 0 <= c0 <= n.

In words, this is a lower bound because there exists constants c and n0 where 50n^2 will always be greater than 0 and less than n^3 + 3n.

1. (3 points) Which algorithm should you use? A1 or A2? Justify

We would use A2 because it is a lower bound which means at some point it is going to be less than A1 until infinity, making it more efficient. When we start producing larger sample sizes A2 will be more efficient.

Exercise 3 (25 points)

Consider the polynomial P(n) of degree k:

P(n) = aknk + ak-1nk-1+…..+ a1n + a0. with all ai > 0

Using the definition of Θ(nk), prove that

P(n) € Θ(nk).

To prove that P(n) € Θ (n^k) we need to prove that P(n) € O(n^k) and P(n) € Ω(n^k)

To prove that it is a lower bound we have to first prove this

0 <= c.n^k <= n^k

0 <= c <= 1 🡪 There is going to be a number that we can assign c that will make this valid

To prove that it is an upper bound we first have to prove this

0 <= n^k <= c.n^k

0 <= 1 <= c 🡪 there is going to be a value c that can always be larger than 1, making this valid

There exists c1, c2, and n0 such that 0 <= c1.n^k <= c2.n^k for all n >= n0. Since it can be an asymptotic lower and upper bound, this makes it an asymptotic tight bound.

Exercise 4 (25 points)

List all the functions below from the lowest to the highest order

n 2n nlg(n) ln(n) lg(n) sqrt(n) n2 + lg(n)

en n2 2n-1 lg(lg(n)) n3 (lg(n))2 n! n – n3 + 7n6

1. lg(lg(n))
2. lg(n)
3. (lg(n))^2
4. ln(n)
5. sqrt(n)
6. n
7. n^2
8. n^2 + lg(n)
9. n^3
10. n – n^3 + 7n^6
11. nlg(n)
12. 2^(n-1)
13. 2^n
14. e^n
15. n!

Exercise 5 (20 points)

Consider this algorithm:

a = 0

for i=0 to n

for j=i+1 to n

a = a + 2

The objective is to find the number Na of additions performed by the statement *a = a + 2*. Inspire yourself from the analysis of the naïve sorting algorithm to:

1. Express Na as a function of n.

F(N) = (1/2)n^2 – (1/2)n

1. Express the final value of *a* when the algorithm ends

When the program ends a = 2(i +2)

1. Provide the best bound for Na.

The best bound for N is F(N) € Θ (n^2)