Week-2 Practical

CP2410: Algorithm and Data Structure

Nguyen Quoc Minh Quan

13740328

GitHub: https://github.com/minhquan0902/CP2410--Practicals.git

# Task 1:

(R-3.8) Order the following functions by asymptotic growth rate. 4n log n + 2n 2 10 2 log n 3n + 100 log n 4nlogn+2n, 2^10, 2^(logn), 3n+100logn, 4n, 2^n, n^2 +10n, n^3, nlogn

Solution:

Order from least to greatest

2^10, 2^(logn), 3n+100logn, 4n, nlogn, 4nlogn+2n, n^2+10n, n^3, 2^n

Explanation:

We have:

In the sequence bellow, if a function f (n) precedes a function g(n), we say that f (n) is asymptotically better than g(n). 1, lgn, n, nlgn, n^2, n^3, 2^n

2^10 is constant value

2^(logn) is linear O(n) by the definition of log

3n+100logn is O(n)

4n is also O(n) and larger than 3n+100logn because the 4n term is larger than the 3n term

nlogn is O(nlogn)

4nlogn+2n is also O(nlogn) because the nlogn term dominates the 2n term

N^2 +10n is O (n^2)

N^3 is O(n^3)

2^n is O (2^n) – exponential

## Task 2: 2. (R-3.2) The number of operations executed by algorithms A and B is 8 n log n and 2n^2 , respectively. Determine n0 such that A is better than B for n ≥ n0.

Algorithm A: 8nlogn

Algorithm B: 2n^2

The graphs describing the behavior of these algorithms start out with A higher (slower) than B, and eventually cross. After the point where they cross, B is always higher than A. Therefore, we need to find the point where they cross, that is the value where:

8nlogn = 2n^2.

Applying algebra, we get:

8nlogn = 2n^2

⇔4nlogn=n^2

⇔4logn=n

⇔4 = n/logn

Solve for n we have n=16 since 4 =16/log2(16) = 16/4 =4

Thus n0 = 17, since for all n >= 17, A will be faster than B (at 16 they're equal.)

## Task 3: (R-3.9) Show that if d(n) is O(f(n)), then a*d(n) is O(f(n)), for any constant a > 0.

given that d(n) = O(f(n))

now a*d(n) = O(a*f(n)) where a is any constant

now, considering Big O notation rule, O(kn) is O(n) where k is constant

⇨ O(a*f(n)) is also O(f(n))
⇨ Proved

**Task 4:** See the following functions from ch03/exercises.py in the sample code. For each of example1, to example5, determine the running time, in big Oh notation, of the function in terms of n.

```
def example1(s):
    n = len(s)
    total = 0
    for j in range(n):
        total += s[j]
    return total
```

there are 2 operations before the loop, the loop runs based off of n, overall runtime is O(n)

```
def example2(S):
    n = len(S)
    total = 0
    for j in range(0, n, 2):
        total += S[j]
    return total
```

2 operations before the loop, the loop runs n / 2 times, Overall runtime = O(n)

```
def example3(S):
    n = len(S)
    total = 0
    for j in range(n):
        for k in range(1 + j):
            total += S[k]
        return total
```

2 operations before the loop, inner loop runs 1 + 2 + n times Overall run time is O(n^2)

```
def example4(S):
    n = len(S)
    prefix = 0
    total = 0
    for j in range(n):
        prefix += S[j]
        total += prefix
    return total
```

3 operations before the loop, loop runs n times, Overall run time is O(n)

```
def example5(A, B):
    n = len(A)
    count = 0
    for i in range(n):
        total = 0
        for j in range(n):
            for k in range(1 + j):
                total += A[K]
            if B[i] == total:
                count += 1
            return count
```

2 operations before the loop. outer loop runs n times, Overall run time is O(n^3)