CP2410 Practical 02

Sihan Chen

March 25, 2023

Question 1.

2^{10} belongs to Constant Time, with complexity of $O(1)$;

$2\log n$ belongs to Logarithmic Time, with complexity of $O(\log n)$;

$4n$ belongs to Linear Time, with complexity of $O(n)$;

$3n + 100\log n$ belongs to Linear & Logarithmic Time, with complexity of $O(n + \log n)$;

$n\log n$ belongs to Quasilinear Time, with complexity of $O(n\log n)$;

$4n\log n + 2n$ belongs to Quasilinear & Linear Time, with complexity of $O(n\log n + n)$;

$n^2 + 10n$ belongs to Quadratic & Linear Time, with complexity of $O(n^2 + n)$;

n^3 belongs to Cubic Time, with complexity of $O(n^3)$;

2^n belongs to Exponential Time, with complexity of $O(2^n)$;

So, the asymptotic growth rate order is:

$$2^{10} < 2\log n < 4n < 3n + 100\log n < n\log n < 4n\log n + 2n < n^2 + 10n < n^3 < 2^n$$

Question 2.

For algorithm A ($8n\log n$) to be better than algorithm B ($2n^2$), it requires:

$$8n\log n \leq 2n^2$$

divide $2n$ on both side, we got:

$$4\log n \leq n$$

when $n = 16$:

$$4\log 16 = 4 \times 4 = 16$$

so we know $n_0 = 16$, therefore for $n \geq 16$, algorithm A is better than B.

Question 3.

Since $d(n)$ is $O(f(n))$, we know there is a constant C_1, and a positive integer n_1, so that:

$$|d(n)| \le C_1 \cdot f(n)$$

for all $n \ge n_1$.

By multiplying $|a|$ to both sides, which gives us:

$$|a| \cdot |d(n)| \le |a| \cdot C_1 \cdot f(n)$$

for all $n \ge n_1$.

Let constant $C = |a| \cdot C_1$ and positive integer $n_0 = n_1$, then we got:

$$|a \cdot d(n)| \le C \cdot f(n)$$

for all $n \ge n_0$.

Therefore, we know $a \cdot d(n)$ is $O(f(n))$, for any constant $a > 0$.

Question 4.

Example 1

```
def example1(S):
        n = len(S)                          # 1
        total = 0                           # 1
        for j in range(n):                  # n
                total += S[j]               # 2n
        return total                        # 1
```

Total time complexity is $3n + 3$, and Big-Oh is $O(n)$.

Example 2

```
def example2(S):
        n = len(S)                          # 1
        total = 0                           # 1
        for j in range(0, n, 2):            # n/2
                total += S[j]               # 2(n/2)
        return total                        # 1
```

Total time complexity is $\frac{3}{2}n + 3$, and Big-Oh is $O(n)$.

Example 3

```
def example3(S):
        n = len(S)                          # 1
        total = 0                           # 1
        for j in range(n):                  # n
                for k in range(1 + j):      # n(n+1)/2
                        total += S[k]       # 2(n(n+1)/2)
        return total                        # 1
```

Total time complexity is $\frac{3}{2}n^2 + \frac{5}{2}n + 3$, and Big-Oh is $O(n^2)$.

Example 4

```
def example4(S):
        n = len(S)                          # 1
        prefix = 0                          # 1
        total = 0                           # 1
        for j in range(n):                  # n
                prefix += S[j]              # 2n
                total += prefix             # n
        return total                        # 1
```

Total time complexity is $3n + 4$, and Big-Oh is $O(n)$.

Example 5

```
def example5(A, B):
        n = len(A)                                  # 1
        count = 0                                   # 1
        for i in range(n):                          # n
                total = 0                           # n
                for j in range(n):                  # n^2
                        for k in range(1 + j):      # n*(n(n+1)/2)
                                total += A[k]       # 2(n*(n(n+1)/2))
                if B[i] == total:                   # 2n
                        count += 1                  # n
        return count                                # 1
```

Total time complexity is $\frac{3}{2}n^3 + \frac{3}{2}n^2 + 4n + 3$, and Big-Oh is $O(n^3)$.