Tutorial 1

Running Time and Complexity Analysis

Lecturer: Dr Andrew Hines *TA: Esri Ni*

Question 1. List five attributes of a good algorithm

Correctness, Speed, Efficiency, Security, Robustness, Clarity, Maintainability

Question 2. List the primitive operations found on each line in the `find_max` code below.

```python
def find_max(data):
    '''Return the max element from a non-empty Python list'''
    biggest = data[0]              # initial max val to beat
    for val in data                # for each value
        if val > biggest:          # bigger than current biggest?
            biggest = val          # found a new biggest
    return biggest
```

1: function call, array creation 3: assignment, array access; 4: assignment, array index access; 5: if test, comparison ;6: assignment; 7: return

Question 3. A linear complexity corresponds to a running time that is at most $\mathcal{O}(n)$. Name two other examples of complexity classes and state whether they are expected to be higher or lower $\mathcal{O}(n)$.

Lower: constant: $\mathcal{O}(c)$ or $\mathcal{O}(1)$. Logarithmic: $\mathcal{O}(log(n))$.
Higher: Quadratic: $\mathcal{O}(log(n^2))$. Exponential: $\mathcal{O}(2^n)$. Factorial: $\mathcal{O}(n!)$

Question 4. Give the class of complexity (Big-$\mathcal{O}$ notation) of algorithms with the following estimated running times:

(i) $T1(n) = 3n^3 + 8n^2 + 2n + 7$

(ii) $T2(n) = 6log(2n) + 6$

(iii) $T3(n) = 6k + 9$

(a) $\mathcal{O}(n^3)$ (b) $\mathcal{O}(logn)$ (c) $\mathcal{O}(c)$ or $\mathcal{O}(1)$

Question 5. Order the following functions by asymptotic growth rate.

$4nlogn + 2n$	$3n + 100logn$	$n^2 + 10n$
2^{10}	$4n$	n^3
2^{logn}	2^n	$nlogn$

2^{10}, 2^{logn}, $3n + 100logn$, $4n$, $nlogn$, $4nlogn + 2n$, $n^2 + 10n$, n^3 , 2^n

Question 6. Show that $(n + 1)^5$ is $\mathcal{O}(n^5)$.

By the definition of big-$\mathcal{O}$, we need to find a real constant $c > 0$ and an integer constant $n_0 \geq 1$ such that $(n + 1)^5 \leq c(n^5)$ for every integer $n \geq n_0$. Since $(n + 1) \leq 2n$ for $n \geq 1$, we have that $(n + 1)^5 \leq (2n)^5 = 32n^5 = c(n5)$ for $c = 32$ and $n \geq n_0 = 1$.

Question 7. Give a big-$\mathcal{O}$ characterisation, in terms of n, of the running time of the function shown in Code Fragment below:

```python
def sum_seq(S):
    '''Return the sum of the elements in sequence S.'''
    n = len(S)
    total = 0
    for j in range(n): # loop for 0 to n−1
        total += S[j]
    return total
```

The `sum_seq` function runs in $\mathcal{O}(n)$ time.

Question 8.
Give a big-$\mathcal{O}$ characterisation, in terms of n, of the running time of the function shown in Code Fragment below:

```python
def sum_pre(S):
    '''Return the sum of the prefix sums of sequence S.'''
    n = len(S)
    total = 0
    for j in range(n):
        for k in range(1+j):
            total += S[k]
    return total
```

Hint: Consider the number of times the inner loop is executed and how many primitive operations occur in each iteration, and then do the same for the outer loop.
Solution: The function runs in $\mathcal{O}(n^2)$ time.

Question 9. Communication security is extremely important in computer networks, and one way many network protocols achieve security is to encrypt messages. Typical cryptographic schemes for the secure transmission of messages over such networks are based on the fact that no efficient algorithms are known for factoring large integers. Hence, if we can represent a secret message by a large prime number p, we can transmit, over the network, the number $r = p \cdot q$, where $q > p$ is another large prime number that acts as the encryption key. An eavesdropper who obtains the transmitted number r on the network would have to factor r in order to figure out the secret message p. Using factoring to figure out a message is very difficult without knowing the encryption key q. To understand why, consider the following naive factoring algorithm:

```python
for p in range(2,r):
    if r % p == 0: # if p divides r
        return 'The secret message is p!'
```

(a) Suppose that the eavesdropper uses the above algorithm and has a computer that can carry out in 1 microsecond (1 millionth of a second) a division between two integers of up to 100 bits each. Give an estimate of the time that it will take in the worst case to decipher the secret message p if the transmitted message r has 100 bits.

(b) What is the worst-case time complexity of the above algorithm? Since the input to the algorithm is just one large number r, assume that the input size n is the number of bytes needed to store r, that is, $n = (log_2 r)/8 + 1$, and that each division takes time $\mathcal{O}(n)$.

Since r is represented with 100 bits, any candidate p that the eavesdropper might use to try to divide r uses also at most 100 bits. Thus, this very naive algorithm requires 2^{100} divisions, which would take about 2^{80} seconds, or at least 2^{55} years. Even if the eavesdropper uses the fact that a candidate p need not ever be more than 50 bits, the problem is still difficult. For in this case, 2^{50} divisions would take about 2^{30} seconds, or about 34 years.
Since each division takes time $\mathcal{O}(n)$ and there are 2^{4n} total divisions, the asymptotic running time is $\mathcal{O}(n \cdot 2^{4n})$.