SC1007 Tutorial 4
Algorithm Analysis and Searching

Dr Liu Siyuan

Email: syliu@ntu.edu.sg

Office: N4-2C-72a

Question 1

The function subset() takes two linked lists of integers and determines whether the first is a subset of the second. Assume there are no duplicate numbers in each list.

a) When will the worst case happen?

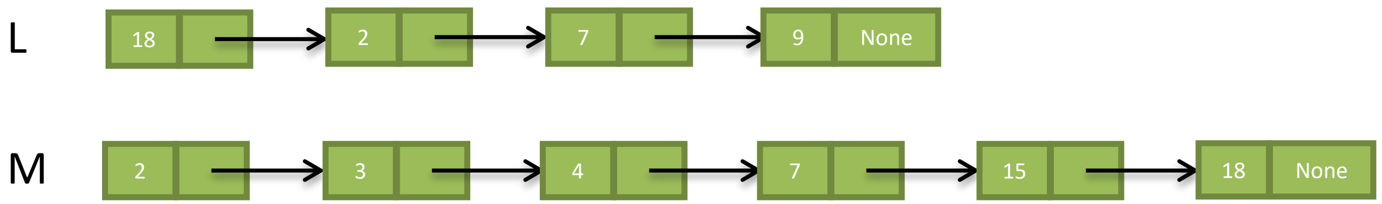
b) In the worst case, how many item comparison operations will be made?

c) Give the worst-case time complexity of subset as a function of the lengths of the two lists.

d) Write down the worst-case time complexity in asymptotic notations in terms of the lengths of the two lists.

```python
# Check whether integer X is an element of linked list Q
def element(X, Q):
    found = False  # Flag whether X has been found
    while Q is not None and not found:
        found = (Q.item == X)
        Q = Q.next
    return found

# Check whether L is a subset of M
def subset(L, M):
    success = True  # Flag whether L is a subset so far
    while L is not None and success:
        success = element(L.item, M)
        L = L.next
    return success
```




```python
# Check whether integer X is an element of linked list Q
def element(X, Q):
    found = False  # Flag whether X has been found
    while Q is not None and not found:
        found = (Q.item == X)
        Q = Q.next
    return found

# Check whether L is a subset of M
def subset(L, M):
    success = True  # Flag whether L is a subset so far
    while L is not None and success:
        success = element(L.item, M)
        L = L.next
    return success
```

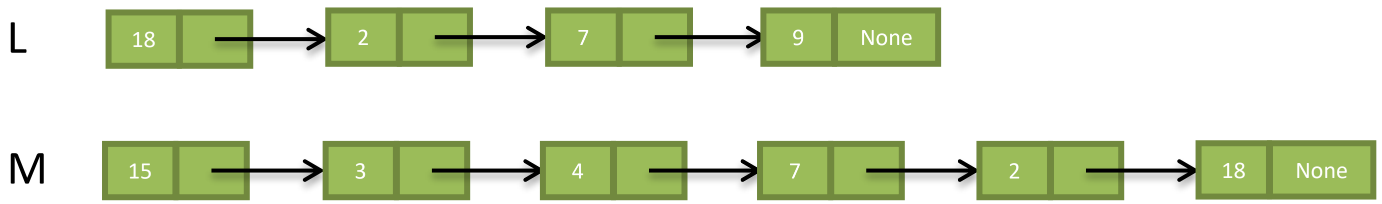
C1

C2

- Node 18: C1+6*C2
- Node 2: C1+C2
- Node 7: C1+4*C2
- Node 9: C1+6*C2

- Worst case1: Check for an element, e.g., 18, until the last element of M matches. The number of comparisons is 6, i.e., |M|
- Worst case2: The element in L is not in M, e.g., 9. The number of comparisons is 6, i.e., |M|

- When the size of M is large, C1 is negligible.




```python
# Check whether integer X is an element of linked list Q
def element(X, Q):
    found = False  # Flag whether X has been found
    while Q is not None and not found:
        found = (Q.item == X)
        Q = Q.next
    return found

# Check whether L is a subset of M
def subset(L, M):
    success = True  # Flag whether L is a subset so far
    while L is not None and success:
        success = element(L.item, M)
        L = L.next
    return success
```

C2

Let |L| and |M| indicate the length of the linked list, L and M. Assuming there are no duplicate numbers in each list, and |L|<|M|.

Worst case example: the first |L|-1 elements of L are from the last |L|-1 elements of M in reverse order, and the last element of L is not in M.

The number of comparisons:

$$= \underbrace{|M| + (|M|-1) + \cdots + (|M|-(|L|-2))}_{\text{The first } |L|-1 \text{ elements in L}} + \underbrace{|M|}_{\text{The last element in L}}$$

$$= (|L|-1)|M| - (1+2+\cdots+(|L|-2)) + |M|$$

$$= |L||M| - \frac{\big(1+(|L|-2)\big)\times(|L|-2)}{2}$$

$$= |L||M| - \frac{(|L|-1)(|L|-2)}{2} = ?$$

$$1+2+\cdots+n = \frac{(1+n)n}{2}$$

Asymptotic Notations (Review)

- Given $f(n), g(n)$
  - $\Omega(g(n))$: set of functions that grow at higher or same rate as g
  - $\Theta(g(n))$: set of functions that grow at same rate as g
  - $O(g(n))$: set of functions that grow at lower or same rate as g

$\lim_{n\to\infty} \frac{f(n)}{g(n)}$	$f(n) \in O(g(n))$	$f(n) \in \Omega(g(n))$	$f(n) \in \Theta(g(n))$
$\mathbf{0}$	✓		
$\mathbf{0 < C < \infty}$	✓	✓	✓
∞		✓	

Time Complexity in $|L|, |M|$

Time complexity:

$$f(|L|, |M|) = c_2 \times (|L||M| - \frac{(|L|-1)(|L|-2)}{2}) = \Theta(|L||M|)$$

- $f(|L|, |M|) = c_2 \times (|L||M| - \frac{(|L|-1)(|L|-2)}{2}), g(|L|, |M|) = |L||M|$
  - $\Omega(|L||M|)$: set of functions that grow at higher or same rate as g
  - $\Theta(|L||M|)$: set of functions that grow at same rate as g
  - $O(|L||M|)$: set of functions that grow at lower or same rate as g

$$\lim_{|M|\to\infty} \frac{f(|L|, |M|)}{g(|L|, |M|)} = \lim_{|\mathrm{M}|\to\infty} \frac{c_2 \times (|L||M| - \frac{(|L|-1)(|L|-2)}{2})}{|L||M|} = c_2$$

Question 2

- Find the number of print used in the following functions. Write down its time complexity in Θ notation in terms of N.

```python
def Q2a(N):
    j = 1
    while j <= N:
        k = 1
        while k <= N:
            print("SC 1007")
            k *= 2
        j *= 3
```

```python
def Q2b(N):
    if N > 0:
        for i in range(N):
            print("SC 1007")
        Q2b(N - 1)
        Q2b(N - 1)
```

```python
def Q2a(N):
    j = 1
    while j <= N:
        k = 1
        while k <= N:
            print("SC 1007")
            k *= 2
        j *= 3
```

N	Number of print	k value when inner loop stops	j value when outer loop stops
1	1*1	2^1	3^1
10	3*4	2^4	3^3
100	5*7	2^7	3^5

- For the inner loop:

$$2^{K-1} \leq N \leq 2^K$$
$$(K-1) \leq log_2 N \leq \mathrm{K}$$
$$K \leq log_2 N + 1 \leq K + 1$$
$$K = \lfloor log_2 N \rfloor + 1$$

- For the outer loop:

$$3^{J-1} \leq N \leq 3^J$$
$$(J-1) \leq log_3 N \leq \mathrm{J}$$
$$J \leq log_3 N + 1 \leq J + 1$$
$$J = \lfloor log_3 N \rfloor + 1$$

- The number of print is $JK = (\lfloor log_3 N \rfloor + 1)(\lfloor log_2 N \rfloor + 1)$

Common Complexity Classes

Order of Growth	Class	Example
1	Constant	Finding midpoint of an array
$\log_2 n$	Logarithmic	Binary Search
n	Linear	Linear Search
$n\log_2 n$	Linearithmic	Merge Sort
n^2	Quadratic	Bubble Sort
n^3	Cubic	Matrix Inversion (Gauss-Jordan Elimination)
2^n	Exponential	Fibonacci Sequence (recursive)
n!	Factorial	Travelling Salesman Problem

```python
def Q2b(N):
    if N > 0:
        for i in range(N):
            print("SC 1007")
        Q2b(N - 1)
        Q2b(N - 1)
```

N	Number of printf	inner loop stops		outer loop stops	
1	1*1	2:	2^1	3:	3^1
10	3*4	16:	2^4	27:	3^3
100	5*7	128:	2^7	243:	3^5

- Time number of print is $JK = (\lfloor log_3 N\rfloor + 1)(\lfloor log_2 N\rfloor + 1)$
- $\lim_{N\to\infty} \frac{(\lfloor log_3 N\rfloor+1)(\lfloor log_2 N\rfloor+1)}{(log_2\ N)^2} = \frac{1}{log_2 3}$
- The time complexity is $\Theta((log_2\ N)^2)$

```python
def Q2b(N):
    if N > 0:
        for i in range(N):
            print("SC 1007")
        Q2b(N - 1)
        Q2b(N - 1)
```

- $W_1 = 1$
- $W_2 = 2 + W_1 + W_1$
- $W_N = N + W_{N-1} + W_{N-1}$

$$= N + 2W_{N-1}$$

$$= N + 2(N - 1 + 2W_{N-2})$$

$$= N + 2(N-1) + 2^2 W_{N-2}$$

$$= \mathrm{N} + 2(\mathrm{N}-1) + 2^2(N-2) + \cdots + 2^{N-1}(W_1) = \sum_{t=0}^{N-1} 2^t (N-t)$$

$$= N\sum_{t=0}^{N-1} 2^t - \sum_{t=0}^{N-1} 2^t t$$

$$= N\sum_{t=0}^{N-1} 2^t - 2\sum_{t=1}^{N-1} 2^{t-1} t = ?$$

Series

- Geometric Series

$$G_n = \frac{a(1-r^n)}{1-r} \qquad \sum_{t=0}^{N-1} 2^t = \frac{1-2^N}{1-2} = 2^N - 1$$

- Arithmetic Series

$$A_n = \frac{n}{2}[2a + (n-1)d] = \frac{n}{2}[a_0 + a_{n-1}]$$

- Arithmetico-geometric Series

$$\sum_{t=1}^{k} t2^{t-1} = 2^k(k-1) + 1 \qquad \sum_{t=1}^{N-1} 2^{t-1}t = 2^{N-1}(N-2) + 1$$

- Faulhaber's Formula for the sum of the p-th powers of the first n positive integers

$$\sum_{k=1}^{n} k^2 = \frac{n(n+1)(2n+1)}{6}$$

$$\sum_{k=1}^{n} k^3 = \frac{n^2(n+1)^2}{4}$$

*Derivation is in note section 0.7.4.1

The number of print: $W_N = N\sum_{t=0}^{N-1} 2^t - 2\sum_{t=1}^{N-1} 2^{t-1}t = 2^{N+1} - 2 - N$

Common Complexity Classes

Order of Growth	Class	Example
1	Constant	Finding midpoint of an array
$\log_2 n$	Logarithmic	Binary Search
n	Linear	Linear Search
$n\log_2 n$	Linearithmic	Merge Sort
n^2	Quadratic	Bubble Sort
n^3	Cubic	Matrix Inversion (Gauss-Jordan Elimination)
2^n	Exponential	Fibonacci Sequence (recursive)
n!	Factorial	Travelling Salesman Problem

```python
def Q2b(N):
    if N > 0:
        for i in range(N):
            print("SC 1007")
        Q2b(N - 1)
        Q2b(N - 1)
```

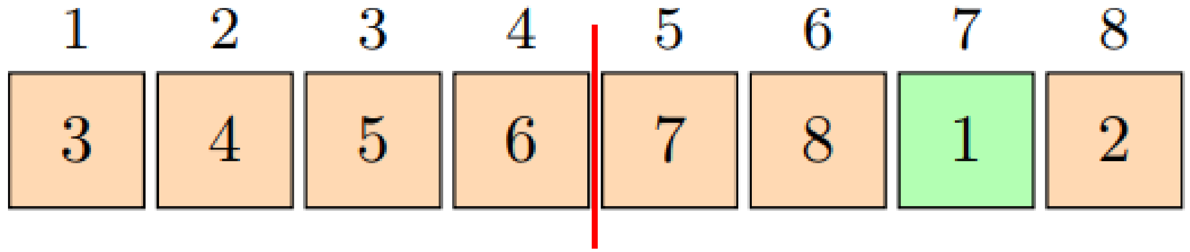
- $W_1 = 1$
- $W_N = N + W_{N-1} + W_{N-1}$
  $= N + 2W_{N-1}$
  $= N + 2(N - 1 + 2W_{N-2})$
  $= N + 2(N - 1) + 2^2 W_{N-2}$
  $= \mathrm{N} + 2(\mathrm{N} - 1) + 2^2(N - 2) + \cdots + 2^{N-1}(W_1) = \sum_{t=0}^{N-1} 2^t (N - t)$
  $= \mathrm{N} \sum_{t=0}^{N-1} 2^t - 2 \sum_{t=0}^{N-1} 2^{t-1} t = 2^{N+1} - 2 - N$
- $\lim\limits_{N\to\infty} \frac{2^{N+1}-2-N}{2^N} = 2$
- The time complexity is $\Theta(2^N)$

Question 3

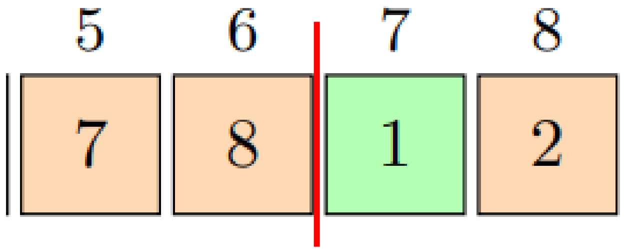
- A sequence, $x_1, x_2, \dots, x_n$, is said to be cyclically sorted if the smallest number in the sequence is x_i $for\ some\ i$, and the sequence, $x_i, x_{i+1}, \dots, x_n, x_1, x_2, \dots, x_{i-1}$ is sorted in increasing order. Design an algorithm to find the minimal element in the sequence in $O(logn)$ time. What is the worst-case scenario?

1	2	3	4	5	6	7	8
6	7	8	1	2	3	4	5

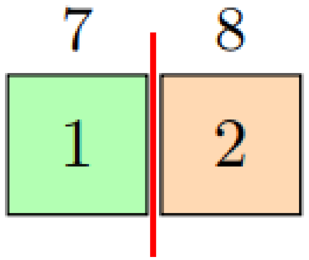
1	2	3	4	5	6	7	8
3	4	5	6	7	8	1	2



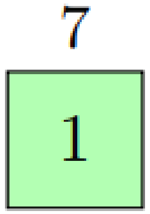
middle = 6, middle > last, i.e., 2.
The minimum is in the second half



middle = 8, middle > last, i.e., 2
The minimum is in the second half



middle = 1, middle<last, i.e., 2
The minimum is in the first half



Only one element

- Time complexity

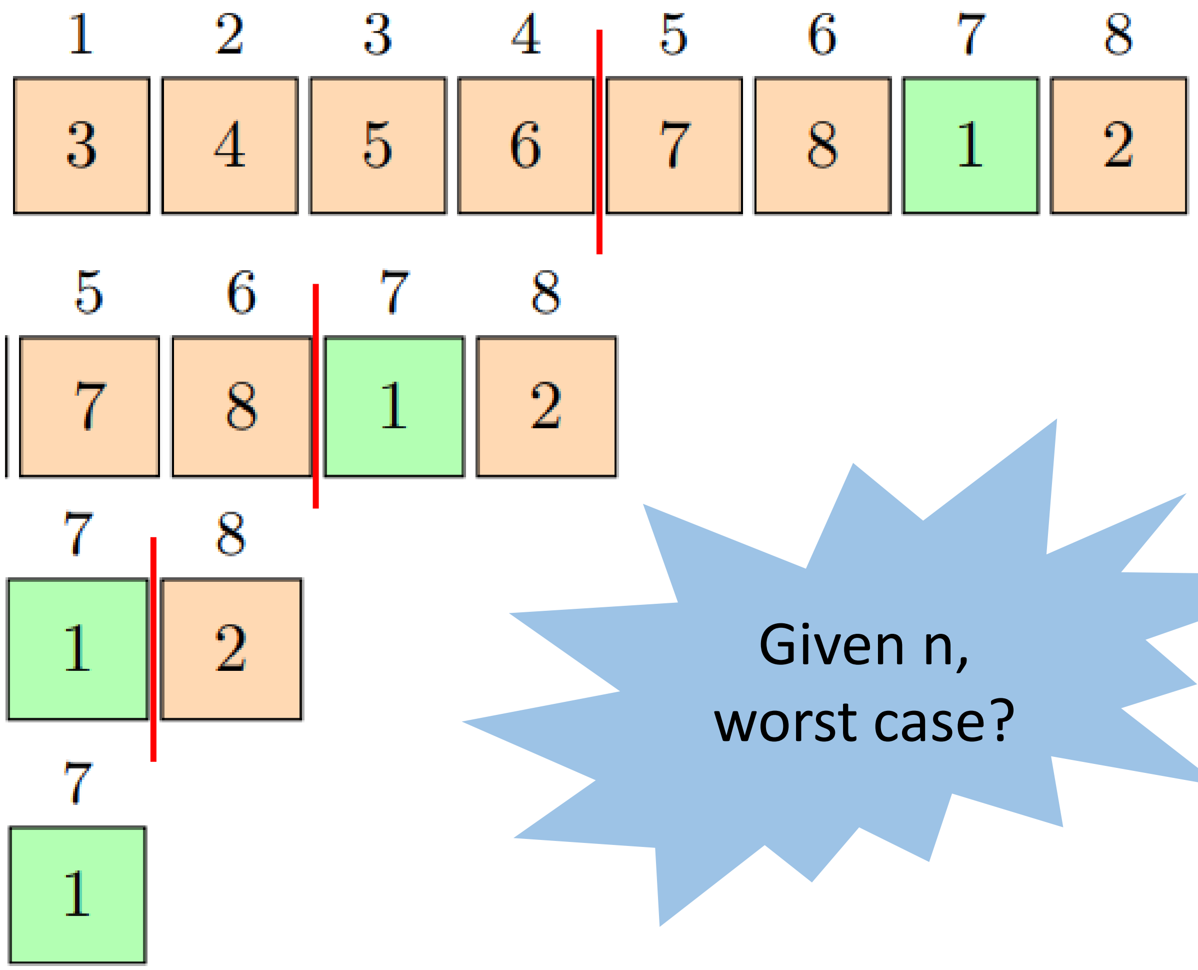
$$T(n) = T\left(\frac{n}{2}\right) + c$$
$$T\left(\frac{n}{2}\right) = T\left(\frac{n}{4}\right) + c$$
$$T(n) = T\left(\frac{n}{4}\right) + 2c$$
$$\vdots$$
$$T(n) = T\left(\frac{n}{2^k}\right) + kc$$

$$0 < n/2^k \le 1$$
$$k \ge \log_2 n$$
$$k = \lceil \log_2 n \rceil$$
$$T(n) = T\left(\frac{n}{2^k}\right) + kc$$
$$= T(1) + kc$$
$$= (\lceil \log_2 n \rceil + 1)c = \Theta(\log_2 n)$$

```python
def find_minimum(array, m, n):  3 usages
    if m == n:
        return array[m]
    else:
        middle = (m + n) // 2
        if array[middle] < array[n]:  # in the first half
            return find_minimum(array, m, middle)
        else:  # in the second half
            return find_minimum(array, middle + 1, n)

array = [3, 4, 5, 6, 7, 8, 1, 2]
minimum = find_minimum(array, m: 0, len(array) - 1)
print(f"the minimum value is {minimum}")
```

c



middle = 6, middle > last, i.e., 2.
The minimum is in the second half

middle = 8, middle > last, i.e., 2
The minimum is in the second half

Given n,
worst case?

middle = 1, middle<last, i.e., 2
The minimum is in the first half

Only one element

Given n, all cases have to run the same number of comparisons until only one element is left.