SC1007 Tutorial 4
Algorithm Analysis

Dr Liu Siyuan

Email: syliu@ntu.edu.sg

Office hour: Monday & Wednesday 4-5pm

Office: N4-2C-117a

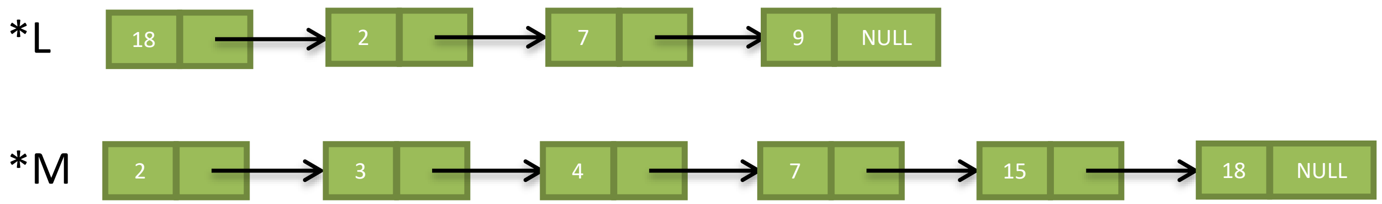
Question 1

- The function subset() takes two linked lists of integers and determines whether the first is a subset of the second.
- Give the worst-case running time of subset as a function of the lengths of the two lists.
- When will this worst case happen?

```c
typedef struct _listnode{
  int item;
  struct _listnode *next;
} ListNode;

//Check whether integer X is an element of linked list Q
int element (int X, ListNode* Q)
{
  int found;  //Flag whether X has been found
  found = 0;
  while ( Q != NULL && !found) {
     found = Q->item == X;
     Q = Q->next;
  }
  return found;
}

// Check whether L is a subset of M
int subset (ListNode* L, ListNode* M)
{
  int success; // Flag whether L is a subset so far
  success = 1;
  while ( L != NULL && success) {
      success = element(L->item, M);
      L = L->next;
  }
  return success;
}
```




```c
typedef struct _listnode{
  int item;
  struct _listnode *next;
} ListNode;

//Check whether integer X is an element of linked list Q
int element (int X, ListNode* Q)
{
  int found; //Flag whether X has been found
  found = 0;
  while ( Q != NULL && !found) {
     found = Q->item == X;
     Q = Q->next;
  }
  return found;
}

// Check whether L is a subset of M
int subset (ListNode* L, ListNode* M)
{
  int success; // Flag whether L is a subset so far
  success = 1;
  while ( L != NULL && success) {
      success = element(L->item, M);
      L = L->next;
  }
  return success;
}
```

C1

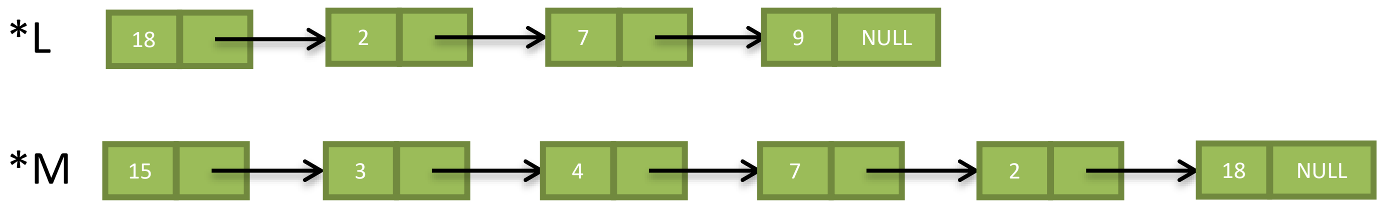
C2

- 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

- Worst case2: The element in L is not in M, e.g., 9

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




```c
typedef struct _listnode{
  int item;
  struct _listnode *next;
} ListNode;

//Check whether integer X is an element of linked list Q
int element (int X, ListNode* Q)
{
  int found;  //Flag whether X has been found
  found = 0;
  while ( Q != NULL && !found) {
     found = Q->item == X;
     Q = Q->next;
  }
  return found;
}

// Check whether L is a subset of M
int subset (ListNode* L, ListNode* M)
{
  int success; // Flag whether L is a subset so far
  success = 1;
  while ( L != NULL && 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 running time:

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

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

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

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

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

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) ∈ O(g(n))	f(n) ∈ Ω(g(n))	f(n) ∈ Θ(g(n))
0	✓		
$\mathbf{0 < C < \infty}$	✓	✓	✓
∞		✓	

Time Complexity in $|L||M|$

- $f(|L|,|M|) = |L||M| - \frac{(|L|-1)(|L|-2)}{2}, g(|L|,|M|) = |L||M|$
  - $\Omega(|L||M|)$: 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

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

The running time:

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

Question 2

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

```
void Q2a (int N)
{
    int j, k;
    for (j=1; j<=N;j*=3)
        for(k=1;k<=N; k*=2)
            printf("SC1007\n");
}
```

```
 void Q2b (int N)
{
    int i;
    if(N>0)
    {
        for(i=0;i<N;i++)
            printf("SC1007\n");
        Q2b(N-1);
        Q2b(N-1);
    }
}
```

```c
void Q2a (int N)
{
    int j, k;
    for (j=1; j<=N;j*=3)
        for(k=1;k<=N; k*=2)
            printf("SC1007\n");
}
```

N	Number of printf	k value when inner loop stops	j value when 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

- 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 printf 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	Insertion Sort
n^3	Cubic	Matrix Inversion (Gauss-Jordan Elimination)
2^n	Exponential	The Tower of Hanoi Problem
n!	Factorial	Travelling Salesman Problem

```c
void Q2a (int N)
{
    int j, k;
    for (j=1; j<=N;j*=3)
        for(k=1;k<=N; k*=2)
            printf("SC1007\n");
}
```

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 printf 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 $\Theta((log_2\ N)^2)$

```c
 void Q2b (int N)
{
   int i;
   if(N>0)
   {
       for(i=0;i<N;i++)
           printf("SC1007\n");
       Q2b(N-1);
       Q2b(N-1);
   }
}
```

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

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

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

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

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

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

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

Series

- Geometric Series

$$G_n = \frac{a(1-r^n)}{1-r}$$

- Arithmetic Series

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

- Arithmetico-geometric Series

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

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

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	Insertion Sort
n^3	Cubic	Matrix Inversion (Gauss-Jordan Elimination)
2^n	Exponential	The Tower of Hanoi Problem
$n!$	Factorial	Travelling Salesman Problem

```c
void Q2b (int N)
{
    int i;
    if(N>0)
    {
        for(i=0;i<N;i++)
            printf("SC1007\n");
        Q2b(N-1);
        Q2b(N-1);
    }
}
```

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

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

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

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

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

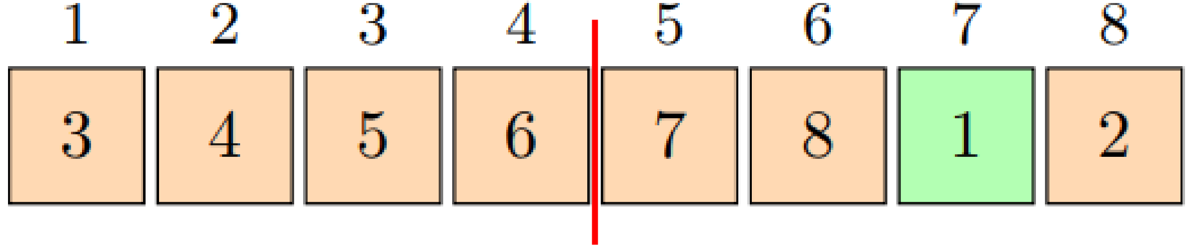
  $= \mathrm{N} \sum_{t=0}^{N-1} 2^t - 2\sum_{t=0}^{N-1} 2^{t-1} t = 2^{N+1} - 2 - N$
- $\lim_{N\to\infty} \frac{2^{N+1}-2-N}{2^N} = 2$
- The time complexity is $\Theta(2^N)$

Question 3

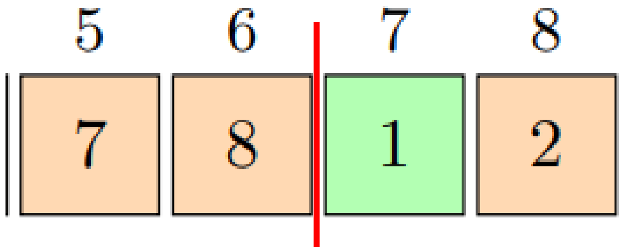
- A sequence, $x_1, x_2, \dots, x_n$, is said to be cyclically sorted if the smallest number in the sequence is x_i $for\ some\ i$, and the sequence, $x_i, x_{i+1}, \dots, x_n, x_1, x_2, \dots, x_{i-1}$ is sorted in increasing order. Design an algorithm to find the minimal element in the sequence in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

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

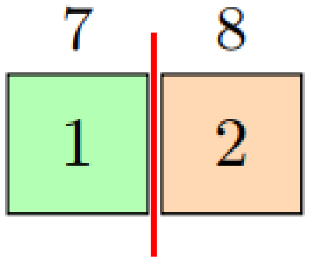
1	2	3	4	5	6	7	8
3	4	5	6	7	8	1	2



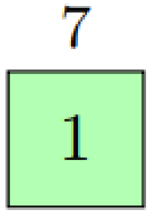
middle = 6, middle > last, i.e., 2.
The minimum is in the second half



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

- The number of comparisons (line 22)
  - W1 = 1
  - W2 = 2
  - W4 = 3
  - W8 = 4
  - ….
- Time complexity
  - $\mathrm{T}(n) = T\left(\frac{n}{2}\right) + c$
- Worst Case scenario
  - We need to cut the array until only one element is left.
  - No differences among scenarios.

```c
9  #include <stdio.h>
10
11
12 int findminimum(int array[], int m, int n)
13 {
14     printf("the m value is %d\n", m);
15     printf("the n value is %d\n", n);
16     int middle;
17     if (m == n)
18         return array[m];
19     else{
20         middle = (n+m)/2;
21         printf("the middle value is %d\n", array[middle]);
22         if (array[middle]<array[n]) //in the first half
23             return findminimum(array, m, middle);
24         else return findminimum(array, middle+1, n); //in the second half
25
26     }
27 }
28
29 int main()
30 {
31   int array[] = {3, 4, 5, 6, 7, 8, 1, 2};
32   int minimum = 0;
33   minimum = findminimum(array, 0, 7);
34   printf("the minimum value is %d", minimum);
35 }
```