การวิเคราะห์ความซับซ้อนอัลกอริทึม
(Algorithm Complexity Analysis)

โดย ผศ. ดร. จรัสศรี รุ่งรัตนาอุบล

สอวน. ค่าย 2

13 มีนาคม 2562

อัลกอริทึมคืออะไร

- เป็นกระบวนงานหรือ ขั้นตอนวิธี สำหรับการแก้ปัญหาในแต่ละขั้นตอนเพื่อให้บรรลุเป้าหมาย หรือแก้ปัญหาได้
- กระบวนการซึ่งสามารถนำไปดำเนินงานได้โดยเครื่องคอมพิวเตอร์ และมีคุณสมบัติหลัก ดังนี้
  - มีขั้นตอนการทำงานที่ไม่กำกวม
  - กระบวนการเรียบง่ายแต่เพียงพอต่อการบรรลุวัตถุประสงค์
  - เป็นกระบวนการที่มีจุดจบ ให้คำตอบ

การวิเคราะห์อัลกอริทึม

- ปัญหาหนึ่งๆ อาจมีวิธีแก้ปัญหาได้มากกว่า 1 วิธีในสภาวะแวดล้อมเดียวกัน
- ดังนั้นการเปรียบเทียบว่าวิธีไหนจะมีประสิทธิภาพดีกว่ากันจึงเป็นเนื้อหาที่เราต้องการเรียนรู้
- การวัดประสิทธิภาพของอัลกอริทึมสามารถทำได้ 2 วิธี คือ
  1. การใช้เนื้อที่ในหน่วยความจำ
  2. ประสิทธิภาพของเวลาในการทำงาน โดยมากจะวัดโดยการจับเวลาในการรันโปรแกรม หรืออาจจะนับจำนวนชุดคำสั่งที่ถูกใช้ในขณะรัน

Time and Space Complexity

- Sometimes, there are more than one way to solve a problem. We need to learn how to compare the performance different algorithms and choose the best one to solve a particular problem
  1. **Time complexity** of an algorithm quantifies the amount of time taken by an algorithm to run as a function of the length of the input
  2. **Space complexity** of an algorithm quantifies the amount of space or memory taken by an algorithm to run as a function of the length of the input
- Complexity depends on lots of things like hardware, operating system, processors, etc. However, we don't care about this.
- We will only consider the **execution time of an algorithm**.

https://www.hackerearth.com/practice/basic-programming/complexity-analysis/time-and-space-complexity/tutorial

Example 1

กำหนดค่าให้อาร์เรย์ขนาด n

วิธีที่ 1

```
list[0] = 0;
list[1] = 1;
list[2] = 2;
list[3] = 3;
…
list[n-1]=n;
```

วิธีที่ 2

```
for(i = 0; i<n; i++)
  list[i] = i;
```

จะใช้เวลาในการทำงานเท่ากันคือ O(n)
ถึงแม้ว่าโปรแกรมจะมีขนาดต่างกัน

Example 2

		เวลาที่ใช้
1.	กำหนดค่า sum = 0	1
2.	i = 0	1
3.	while i < n do	n+1
4.	i = i +1	n
5.	sum = sum + i	n
6.	คำนวณค่าเฉลี่ย mean = sum/n	1
		= 3n + 4

Example 3

```
1. found = 0;
2. loc = 1;
3. while ((loc <= n) && !found)
4.       if(item == a[loc])
5.          found = 1;
6.       else   loc++;
```

```
1. 1
2. 1
3. n + 1
4. n
5. 1
6. n
= 3n + 4
```

Example 4:

```
int count = 0;
for (int i = 0; i < N; i++)
    for (int j = 0; j < i; j++)
        count++;
```

Lets see how many times **count++** will run.

When $i = 0$, it will run 0 times.
When $i = 1$, it will run 1 times.
When $i = 2$, it will run 2 times and so on.

Total number of times **count++** will run is $0 + 1 + 2 + \ldots + (N-1) = \frac{N*(N-1)}{2}$. So the time complexity will be $O(N^2)$.

Example 5:

```
int count = 0;
for (int i = N; i > 0; i /= 2)
    for (int j = 0; j < i; j++)
        count++;
```

This is a tricky case. In the first look, it seems like the complexity is $O(N * logN)$. N for the $j's$ loop and $logN$ for $i's$ loop. But its wrong. Lets see why.

Think about how many times **count++** will run.

When $i = N$, it will run N times.
When $i = N/2$, it will run $N/2$ times.
When $i = N/4$, it will run $N/4$ times and so on.

Total number of times **count++** will run is $N + N/2 + N/4 + \ldots + 1 = 2 * N$. So the time complexity will be $O(N)$.

Example 6

```
1.   found = 0;                                  1.  .
2.   first = 1;                                  2.  .
3.   last = n-1;                                 3.  .
4.   while ((first <= last) && !found)           4.  .
5.       loc = (first + last)/2;                 5.  .
6.       if(item == a[loc])                      6.  .
7.          last = loc -1;                       7.  .
8.       else if(item > a[loc])                  8.  .
9.          first = loc +1;                      9.  .
10.      else                                    10. .
11.         found = 1;                           11. .
```

Big O

- Time complexity notations
- While analysing an algorithm, we mostly consider O-notation because it will give us an upper limit of the execution time i.e. the execution time in the worst case.
- To compute O-notation we will ignore the lower order terms, since the lower order terms are relatively insignificant for large input.

Let $f(N) = 2 * N^2 + 3 * N + 5$

$O(f(N)) = O(2 * N^2 + 3 * N + 5) = O(N^2)$

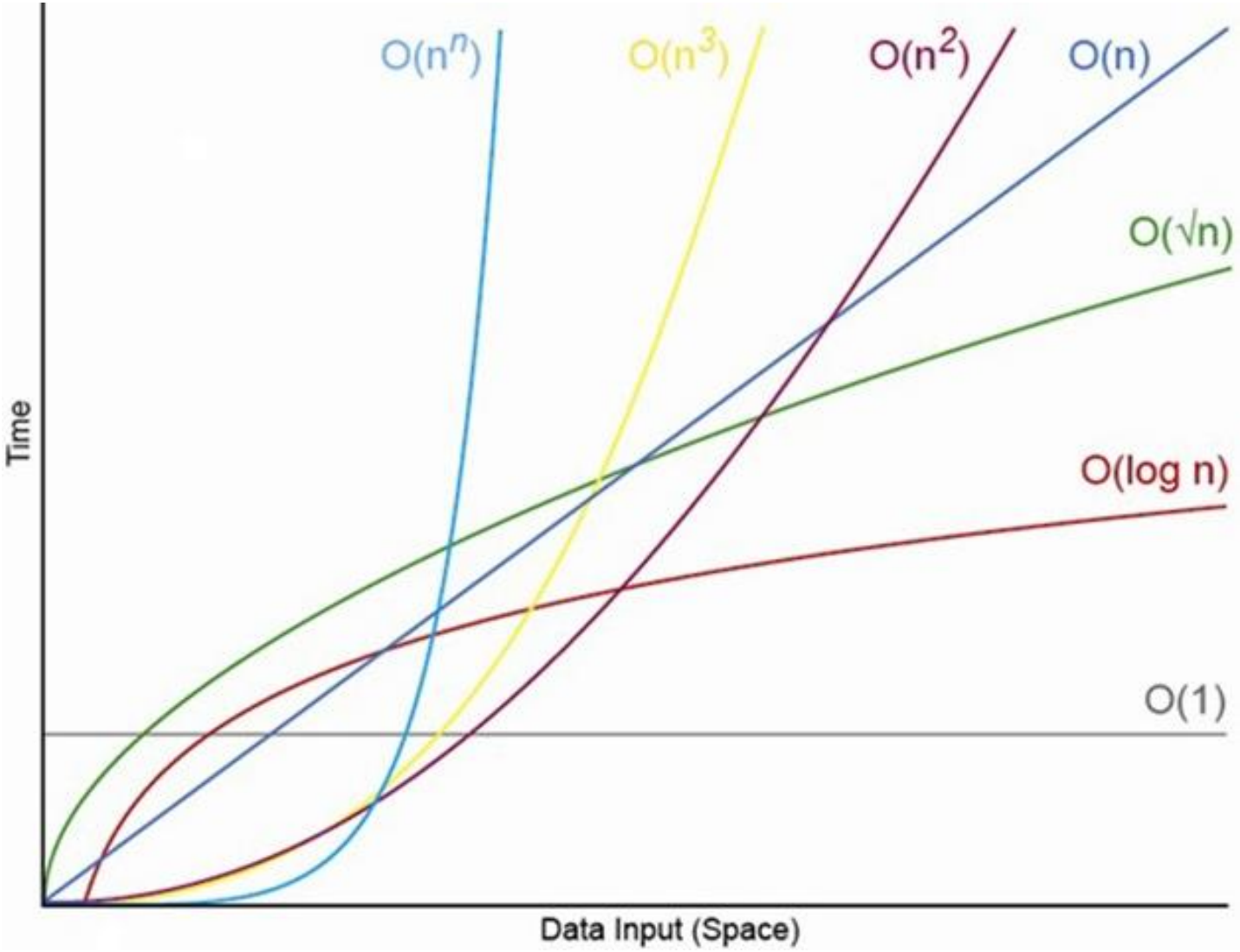
สัญกรณ์บิ๊กโอ (Big O Notation)

- บิ๊กโอ หรืออันดับขนาดเป็นฟังก์ชันที่ได้จากการคำนวณทางคณิตศาสตร์ โดยคำนึงถึงตัวที่มีผลต่อการทำงานรวมมากที่สุดเมื่อขนาดของฟังก์ชันใหญ่ขึ้น
- เช่น $f(n) = n^4 + 100n^2 + 10n + 50$ ค่า บิ๊กโอ จะเท่ากับ n^4

Big O

- O(1) คืออันดับขนาดที่ใช้เวลาคงที่ เช่น การเข้าถึงสมาชิกแต่ละตัวในอาร์เรย์
- O(n) คืออันดับขนาดที่เวลาในการทำงานขึ้นอยู่กับขนาด เช่น การค้นหาข้อมูล ในอาร์เรย์
- $O(\log_2 n)$ คือจะใช้เวลาทำงาน > O(1) แต่ < O(n) เช่น การค้นหาข้อมูลแบบทวิภาค
- $O(n^2)$ คือ จะใช้เวลา > O(n) เช่น การเรียงลำดับข้อมูลแบบธรรมดา
- $O(n^3)$ คือ จะใช้เวลา > $O(n^2)$ เช่น การทำงานของอาร์เรย์แบบ 3 มิติ
- $O(2^n)$ เป็นแบบที่ใช้เวลานานที่สุด นอกจากนี้ยังมี $O(n \log_2 n)$ ที่มีค่าเพิ่มขึ้นช้ากว่า $O(2^n)$

Big O Graph



Bubble sort

- เป็นการเรียงข้อมูลที่มีขั้นตอนดังนี้

6 5 3 1 8 7 2 4

Bubble sort in c

```
void bbsort(int arr[], int n){




}
```

Create array

```
#define n 100
...
int arr[n];
for (int i = 0; i < n; i++) {
    arr[i] = rand() % n + 1;
    //printf("%d\t", n);
}
```

Merge sort

- เป็นการเรียงข้อมูลตามหลักการ divide and conquer

6 5 3 1 8 7 2 4

```
void merge(int arr[], int l, int m, int r)
{

}
void mergesort(int arr[], int l, int r){
  if (l < r) {
  // Same as (l+r)/2, but avoids overflow for large l and h
     int m = l+(r-l)/2;
     // Sort first and second halves
     mergeSort(arr, l, m);
     mergeSort(arr, m+1, r);
     merge(arr, l, m, r);
  }
}
```

main

```
/* Driver program to test above functions */
int main()
{
        int arr[] = {12, 11, 13, 5, 6, 7};
        int arr_size = sizeof(arr)/sizeof(arr[0]);
        printf("Given array is \n");
        printArray(arr, arr_size);
        mergeSort(arr, 0, arr_size - 1);
        printf("\nSorted array is \n");
        printArray(arr, arr_size);
        return 0;
}
```

Fill in the table (Time in second)

N input	Bubble sort	Merge sort
100	0.009221	0.008525
1000	0.011020	0.008081
10000	0.250800	0.009833
100000	26.660000	0.028690

Problem: Majority

- จงเขียนโปรแกรมเพื่อหาว่าชุดข้อมูลหรืออาร์เรย์ชนิด **int** ที่รับเข้ามามีค่าที่เป็น**ค่าหมู่มากหรือไม่?**
- โดยค่าหมู่มากคือค่าที่มีความถี่เกินครึ่งหนึ่งของจำนวนสมาชิก เช่น

  1 1 1 2 2 5 6 3 3 2 1 1 1 **ไม่มีค่าหมู่มาก ตอบ 0**

  1 1 1 2 1 1 6 3 3 2 1 1 1 **มีค่าหมู่มาก ตอบ 1**