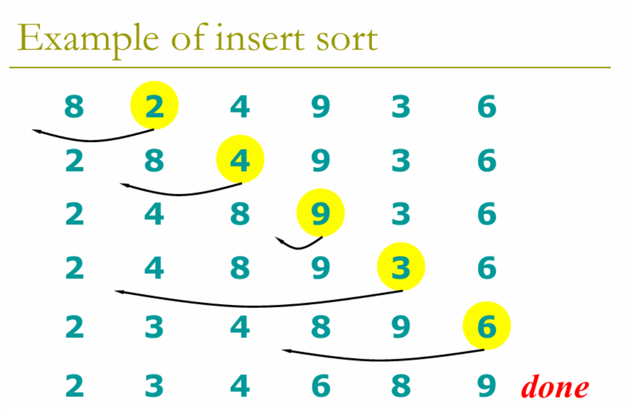
Data Structure Lecture I: Algorithm Complexity Analysis

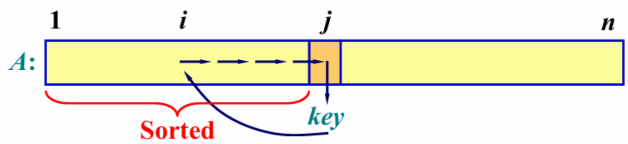
Part 1. Sort Examples

1.Insertion sort



1. 伪代码：

```
INSERTION-SORT(A)
for j ← 2 to length[A]
    do key ← A[j]
       // Insert A[j] into the sorted sequence A[1 .. j − 1]
       i ← j − 1
       while i > 0 and A[i] > key
           do A[i + 1] ← A[i]
              i ← i − 1
       A[i + 1] ← key
```



2. C++实现：

代码块

```
void Insertion_sort(vector<int>& arr){
    int length = arr.size();
    for(int j = 2; j < length; j++){
        int key = arr[j];
        int i = j - 1;
        while(i >= 0 && arr[i] > key){
            arr[i + 1] = arr[i];
            i--;
        }
```

```
            arr[i + 1] = key;
        }
    }
```

3. **Time Complexity Analysis**

INSERTION-SORT(A)	*cost*	*times*
1 **for** $j \leftarrow 2$ **to** $length[A]$	c_1	n
2 **do** $key \leftarrow A[j]$	c_2	$n-1$
3 // Insert $A[j]$ into the sorted sequence $A[1 .. j-1]$	0	$n-1$
4 $i \leftarrow j-1$	c_4	$n-1$
5 **while** $i > 0$ and $A[i] > key$	c_5	$\sum_{j=2}^{n} t_j$
6 **do** $A[i+1] \leftarrow A[i]$	c_6	$\sum_{j=2}^{n} (t_j - 1)$
7 $i \leftarrow i-1$	c_7	$\sum_{j=2}^{n} (t_j - 1)$
8 $A[i+1] \leftarrow key$	c_8	$n-1$

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5 \sum_{j=2}^{n} t_j + c_6 \sum_{j=2}^{n} (t_j - 1) + c_7 \sum_{j=2}^{n} (t_j - 1) + c_8(n-1)$$

Best case:

- The best case occurs if the array is already sorted

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5(n-1) + c_8(n-1)$$
$$= (c_1 + c_2 + c_4 + c_5 + c_8)n - (c_2 + c_4 + c_5 + c_8)$$

***an* + *b* (linear function)**

Worst case:

- The worst case occurs if the array is in reverse sorted order

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5\left(\frac{n(n+1)}{2} - 1\right) + c_6\left(\frac{n(n-1)}{2}\right) + c_7\left(\frac{n(n-1)}{2}\right) + c_8(n-1)$$
$$= \left(\frac{c_5}{2} + \frac{c_6}{2} + \frac{c_7}{2}\right)n^2 + \left(c_1 + c_2 + c_4 + \frac{c_5}{2} - \frac{c_6}{2} - \frac{c_7}{2} + c_8\right)n - (c_2 + c_4 + c_5 + c_8)$$

$an^2 + bn + c$ (quadratic function)

The `a` , `b` , `c` coefficients are related to particular machines.When $n \to \infty$, we don' t care about them.

Insertion sort works best with small amounts of data , since the coef of insert sort is the smallest(only compare and voluation), while other sort algorithms need more operations.

2.Merge Sort

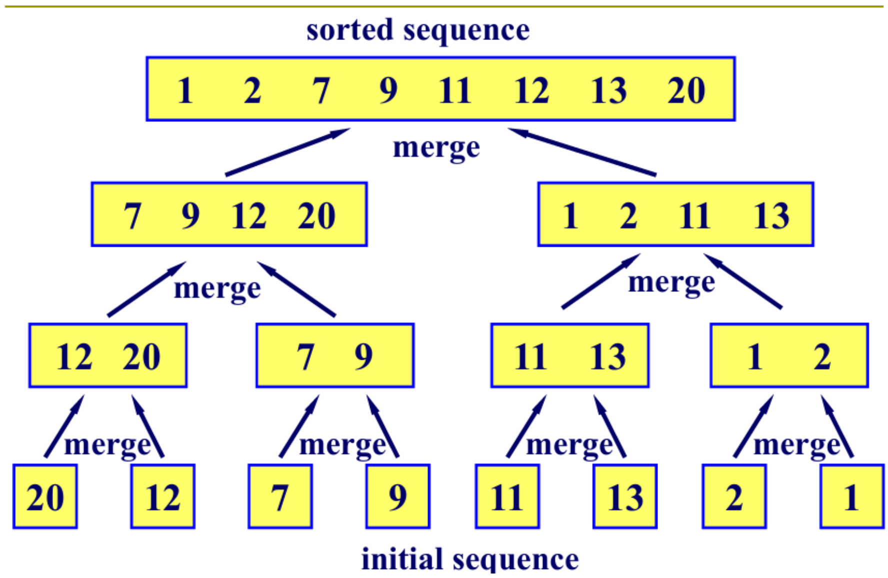
Idea : divide and conquer

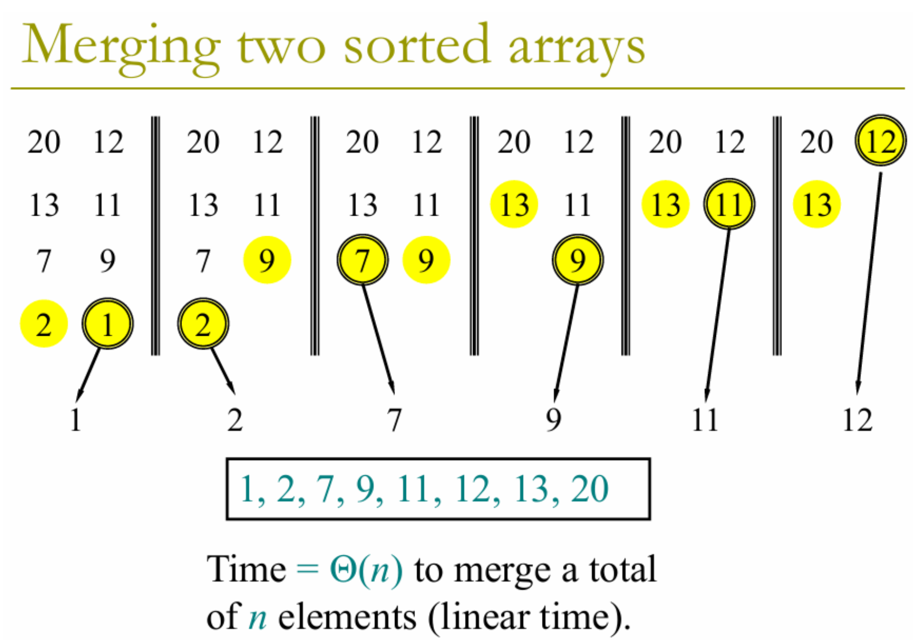
1. pseudocode

MERGE-SORT $A[1 \,..\, n]$

1. If $n = 1$, done.
2. Recursively sort $A[1 \,..\, \lceil n/2 \rceil]$ and $A[\lfloor n/2 \rfloor + 1 \,..\, n]$
3. **"Merge"** the 2 sorted lists.

Key subroutine: ***MERGE***





2. C++ Implementation

代码块

```
void Merge_sort(vector<int>& arr){
    auto helper = [&](this auto && helper, int left, int right) -> void{
        if(left == right - 1){
            return;
        }
        int middle = (left + right) / 2;

        // devide
        helper(left, middle);
```

```
            helper(middle, right);

            vector<int> left_part(arr.begin() + left, arr.begin() + middle);
            vector<int> right_part(arr.begin() + middle, arr.begin() + right);

            // conquer
            int p1 = 0, p2 = 0, p = left;
            while(p1 < middle - left && p2 < right - middle){
                if(left_part[p1] < right_part[p2]){
                    arr[p] = left_part[p1];
                    p1++;
                }
                else{
                    arr[p] = right_part[p2];
                    p2++;
                }
                p++;
            }
        }
}
```

Occupy a lot of memory!

Complexity analysis:

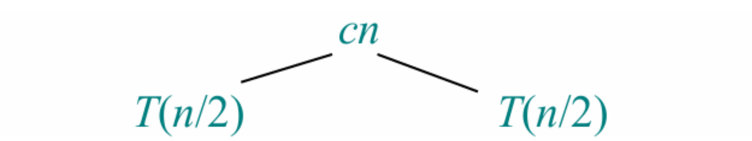
Let's discuss the time complexity of the merge sort:

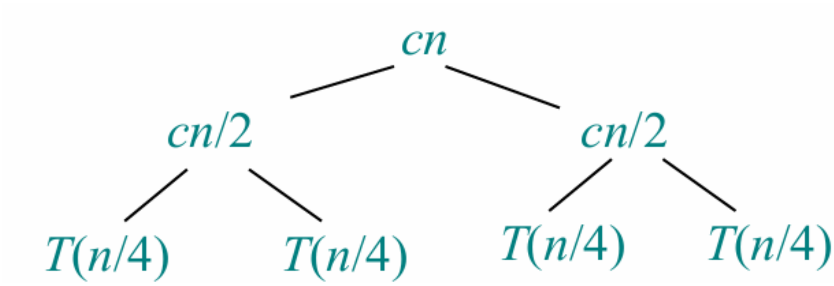
$$T(N) = 2T(\frac{N}{2}) + cn$$

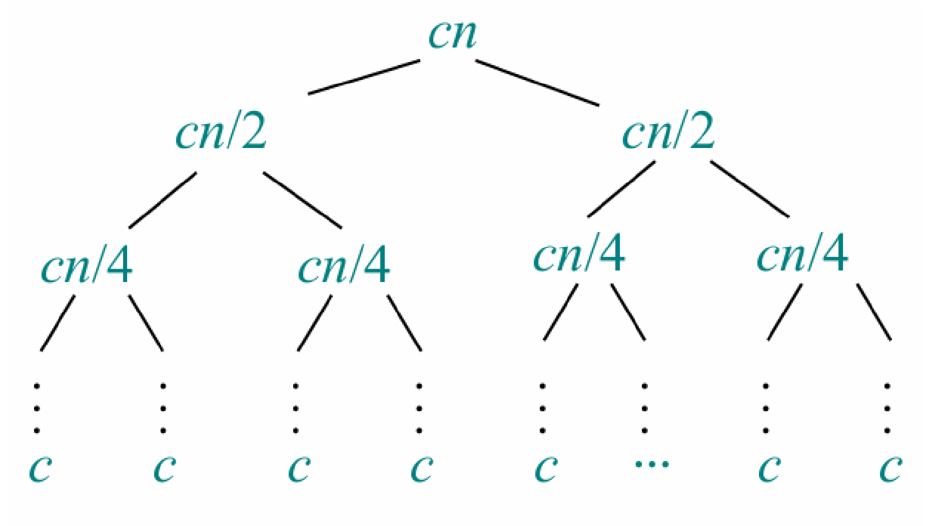
$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1, \\ 2T(n/2) + \Theta(n) & \text{if } n > 1. \end{cases}$$

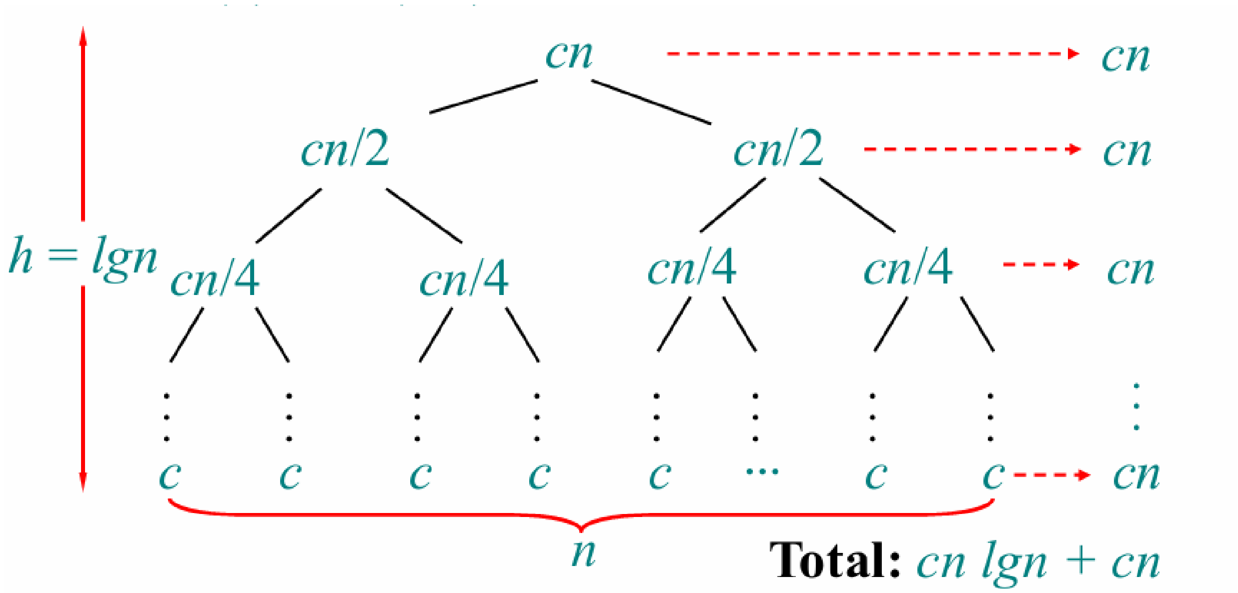
Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.

We draw the recursive tree:









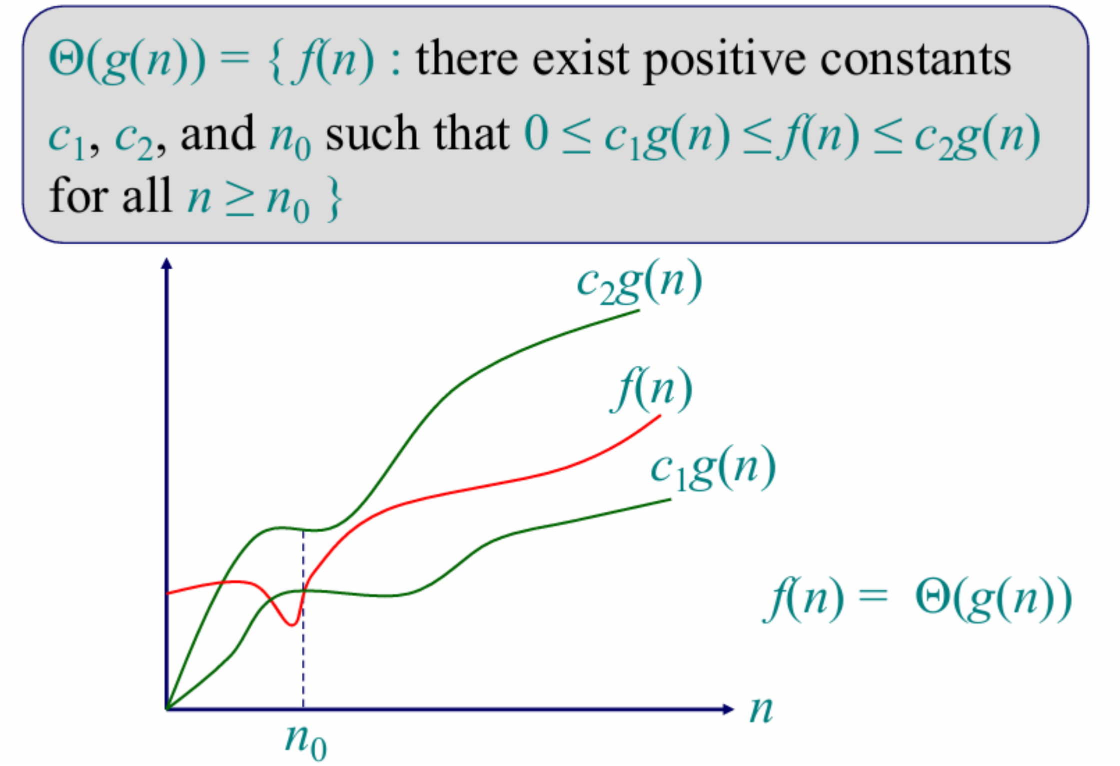
Part 2. Complexity Theory

1.Kind of analyses

- Worst case (usually)
- Average case (sometimes)
- Best case (bogus, no meaning)

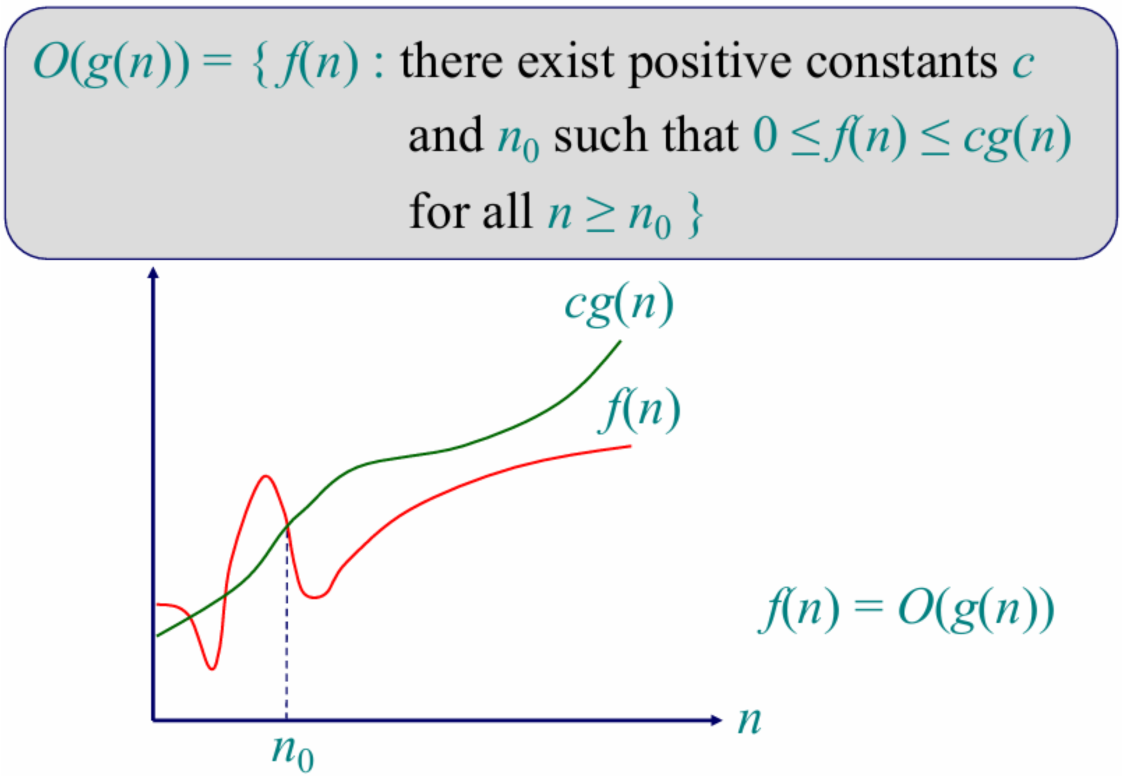
2. Θ, O, Ω Notation

1. Math definition:
    a. Θ Notation



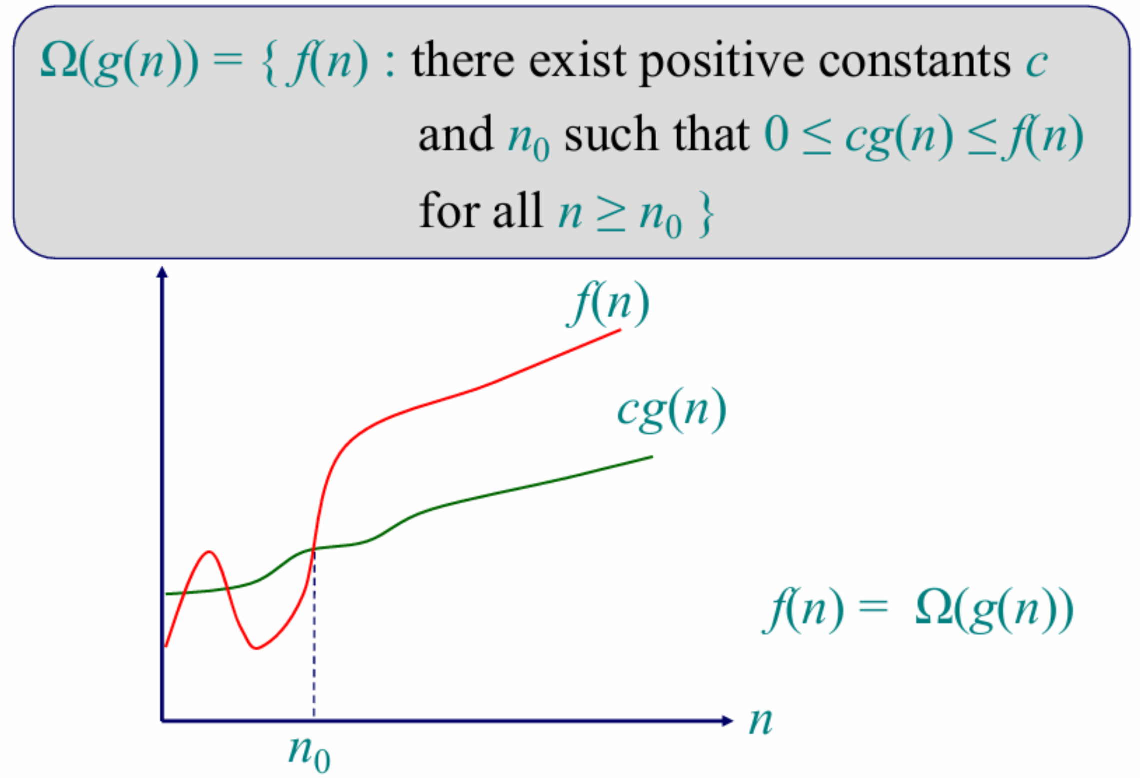
$f(n) = \Theta(g(n))$ can be thought of $f(n) \approx g(n)$ in their scales

b. O Notation



$f(n) = O(g(n))$ can be thought of $f(n) \le g(n)$ in their scales

c. Ω Notation



$f(n) = \Omega(g(n))$ can be thought of $f(n) \ge g(n)$ in their scales

2. Engineering meaning:

When we **drop low order terms** , **ignore leading constants** when $n \to \infty$

For example: $3n^3 + 90n^2 - 5n + 6046 = \Theta(n^3)$

Part3. Recurrences Analyses

✍ Three methods to analyze the complexity of recursive algorithm:

1. Substitution method(数学归纳法)
2. Recursive-Tree method
3. Master Method

1.Substitution method

2.Recursive-tree method

3.Master method