The Complexity of Algorithms (3A)

Young Won Lim
4/3/18

Please send corrections (or suggestions) to youngwlim@hotmail.com.

This document was produced by using LibreOffice and Octave.

Complexity Analysis

- to compare algorithms at the idea level
  ignoring the low level details
- To measure how fast a program is
- To explain how an algorithm behaves
  as the input grows larger

https://discrete.gr/complexity/

Counting Instructions

• Assigning a value to a variable	x= 100;
• Accessing a value of a particular array element	A[i]
• Comparing two values	(x > y)
• Incrementing a value	i++
• Basic arithmetic operations	+, –, *, /
• Branching is not counted	if else

https://discrete.gr/complexity/

Asymptotic Behavior

- avoiding tedious instruction counting
- eliminate all the minor details
- focusing how algorithms behaves when treated badly
- drop all the terms that grow slowly
- only keep the ones that grow fast as **n** becomes larger

https://discrete.gr/complexity/

Finding the Maximum

```
M = A[0];
for (i=0; i<n; ++i) {
        if (A[i] >= M) {
                M = A[i];
        }
}
```

// M is set to the 1st element

// if the (i+1)th element is greater than M,

// M is set to that element (new maximum value)

```
int A[n];      // n element integer array A
int M;         // the current maximum value found so far
               // set to the 1st element, initially
```

https://discrete.gr/complexity/

Worst and Best Cases

int A[4];

i=0	A[0]
i=1	A[1]
i=2	A[2]
i=3	A[3]

Case 1:
Worst Case

A[0]=1	➡ M=1
A[1]=2	➡ M=2
A[2]=3	➡ M=3
A[3]=4	➡ M=4

Case 2:
Best Case

A[0]=4	➡ M=4
A[1]=3	
A[2]=2	
A[3]=1	

```
for (i=0; i<n; ++i) {
    if (A[i] >= M) {      // always n comparisons
        M = A[i];         // the updating of M depends on the data
    }                     // minimum 1 update, maximum n updates
```

https://discrete.gr/complexity/

Assignment

```
M = A[0];                         // 2 instructions
for (i=0; i<n; ++i) {
        if (A[i] >= M) {
                M = A[i];
        }
}
```

A[0] – **1** instruction

M = – **1** instruction

https://discrete.gr/complexity/

Loop instructions

```
for (i=0; i<n; ++i) {
        if (A[i] >= M) {
                M = A[i];
        }
```

Initialization * **1**

i=0	– one instruction
i<n	– one instruction

Update * **n**

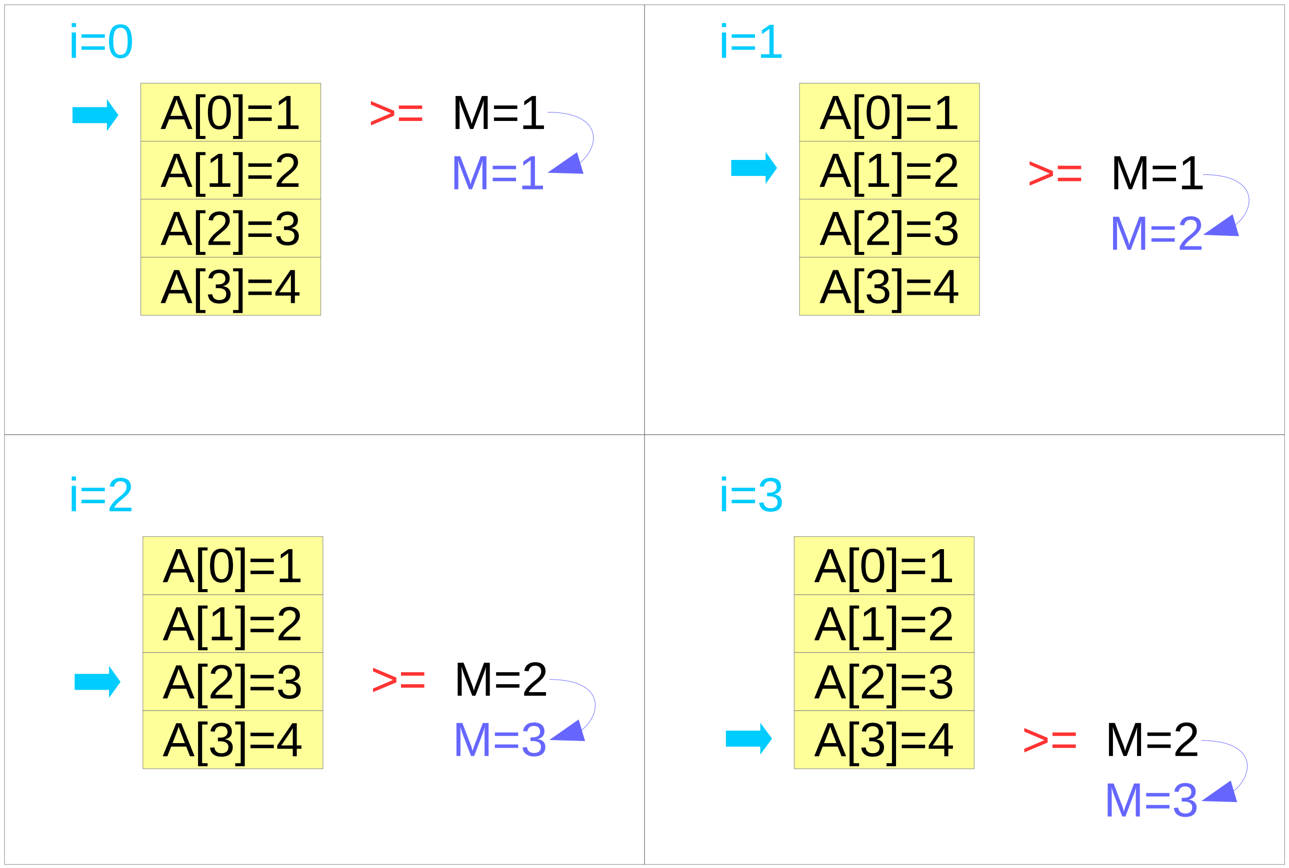
++i	– one instruction
i<n	– one instruction

Loop body * **n**

A[i]	– one instruction	**n**	always
>= M	– one instruction		
A[i]	– one instruction	**1 ~ n**	depending on the comparison
M=	– one instruction		

https://discrete.gr/complexity/

Worst case examples




```
for (i=0; i<n; ++i) {
    if (A[i] >= M) {
        M = A[i];
    }
```

2**n** + 2**n** = 4**n**

Instructions

n comparisons

n updates

https://discrete.gr/complexity/

Best case examples


```
for (i=0; i<n; ++i) {
    if (A[i] >= M) {
        M = A[i];
    }
```

2**n** + 2

Instructions

n comparisons

1 update

https://discrete.gr/complexity/

Asymptotic behavior

M = A[0];	2	instructions
for (i=0; i<**n**; ++i) {	2 + 2**n**	instructions (init + update)
if (A[i] >= M) {	2**n**	instructions
M = A[i];	2 ~ 2**n**	instructions
}		
}		

$$f(n) = \begin{cases} 6n+4 & \text{instructions for the worst case} \\ 4n+6 & \text{instruction for the best case} \end{cases}$$

$f(n) = O(n)$

$f(n) = \Omega(n)$

$f(n) = \Theta(n)$

https://discrete.gr/complexity/

O(**n**) codes

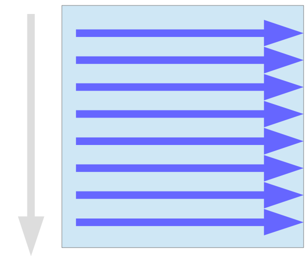
```
// Here c is a positive integer constant
for (i = 1; i <= n; i += c) {
    // some O(1) expressions
}

for (int i = n; i > 0; i -= c) {
    // some O(1) expressions
}
```

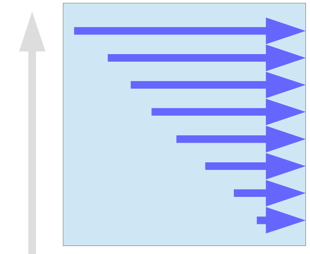
https://stackoverflow.com/questions/11032015/how-to-find-time-complexity-of-an-algorithm

O(n^2) codes

```
for (i = 1; i <=n; i += c) {
   for (j = 1; j <=n; j += c) {
      // some O(1) expressions
   }
}
```



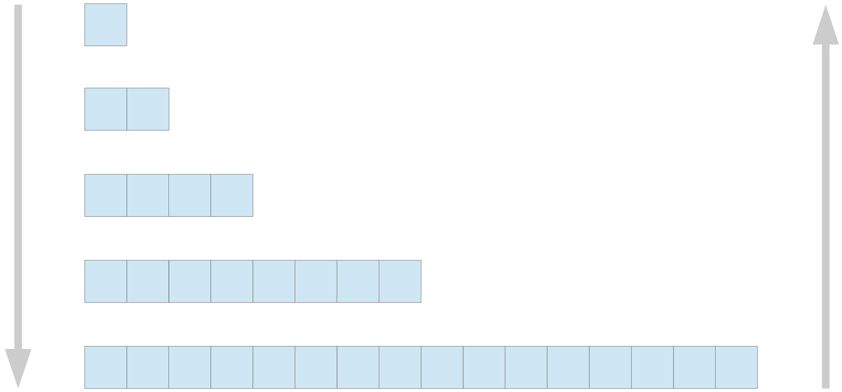
```
for (i = n; i > 0; i += c) {
   for ( j = i+1; j <=n; j += c) {
      // some O(1) expressions
}
```



https://stackoverflow.com/questions/11032015/how-to-find-time-complexity-of-an-algorithm

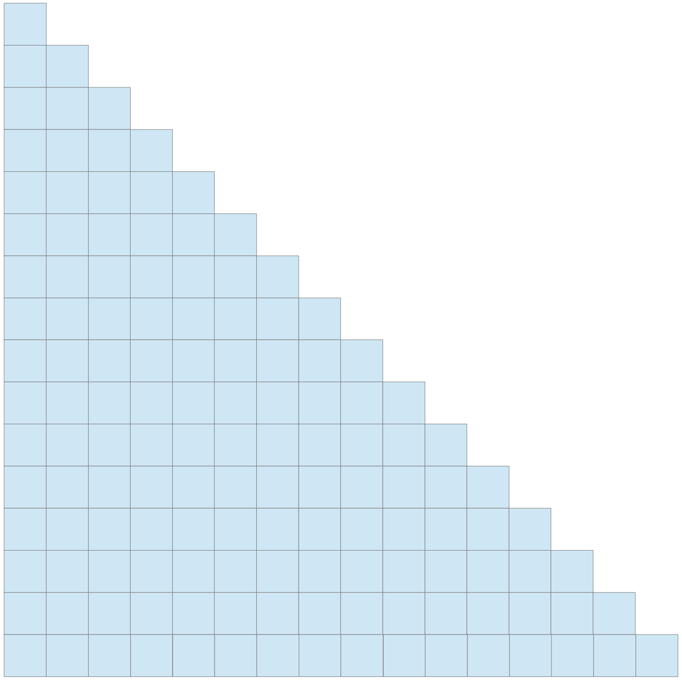
O(log **n**) codes

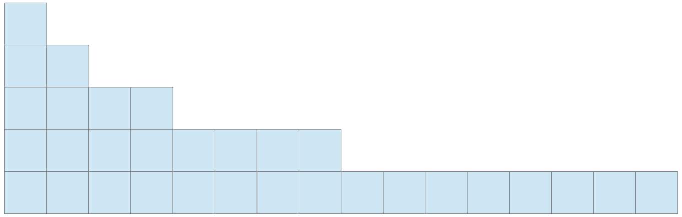
```
for (int i = 1; i <=n; i *= c) {
   // some O(1) expressions
}
for (int i = n; i > 0; i /= c) {
   // some O(1) expressions
}
```



https://stackoverflow.com/questions/11032015/how-to-find-time-complexity-of-an-algorithm

O(**n**) vs. O(log **n**)





https://stackoverflow.com/questions/11032015/how-to-find-time-complexity-of-an-algorithm

O(log **n**) codes

```
// Here c is a constant greater than 1
for (int i = 2; i <=n; i = pow(i, c)) {          // i = i^c
  // some O(1) expressions
}
//Here fun is sqrt or cuberoot or any other constant root
for (int i = n; i > 0; i = fun(i)) {             // i = i^(1/c)
  // some O(1) expressions
}
```

$i = i^2, i = i^3$

https://stackoverflow.com/questions/11032015/how-to-find-time-complexity-of-an-algorithm

O(log log **n**) codes

```
// Here c is a constant greater than 1
for (int i = 2; i <=n; i = pow(i, c)) {          // i = i^c
  // some O(1) expressions
}
//Here fun is sqrt or cuberoot or any other constant root
for (int i = n; i > 0; i = fun(i)) {              // i = i^(1/c)
  // some O(1) expressions
}
```

$i = i^2 \quad (2, 2^2, 2^4, 2^8, 2^{16}, \cdots)$

$i = i^{\frac{1}{2}} \quad (n, n^{\frac{1}{2}}, n^{\frac{1}{4}}, n^{\frac{1}{8}}, n^{\frac{1}{16}}, \cdots)$

https://stackoverflow.com/questions/11032015/how-to-find-time-complexity-of-an-algorithm

O(log log **n**) codes

```
// Here c is a constant greater than 1
for (int i = 2; i <=n; i = pow(i, c)) {          // i = i^c
   // some O(1) expressions
}
//Here fun is sqrt or cuberoot or any other constant root
for (int i = n; i > 0; i = fun(i)) {             // i = i^(1/c)
   // some O(1) expressions
}
```

$i = i^2 \quad (2, 2^2, 2^4, 2^8, 2^{16}, \cdots)$

$i = i^{\frac{1}{2}} \quad (n, n^{\frac{1}{2}}, n^{\frac{1}{4}}, n^{\frac{1}{8}}, n^{\frac{1}{16}}, \cdots)$

https://stackoverflow.com/questions/11032015/how-to-find-time-complexity-of-an-algorithm

Some Algorithm Complexities and Examples (1)

O(1) – Constant Time

not affected by the input size **n**.

O(n) – Linear Time

Proportional to the input size **n**.

O(log n) – Logarithmic Time

recursive subdivisions of a problem

binary search algorithm

O(n log n) – Linearithmic Time

Recursive subdivisions of a problem and then merge them

merge sort algorithm.

https://stackoverflow.com/questions/11032015/how-to-find-time-complexity-of-an-algorithm

Some Algorithm Complexities and Examples (2)

O(n2) – Quadratic Time

bubble sort algorithm

O(n3) – Cubic Time

straight forward matrix multiplication

O(2^n) – Exponential Time

Tower of Hanoi

O(n!) – Factorial Time

Travel Salesman Problem (TSP)

https://stackoverflow.com/questions/11032015/how-to-find-time-complexity-of-an-algorithm

References

[1] http://en.wikipedia.org/
[2]