

How to Analyze Time Complexity

Yifeng Li and Alioune Ngom *

Abstract

In this lab, we give some examples of how to analyze the time complexity of an algorithm.

1 Concepts and Methodology

Procedure of Time Complexity Analysis:

(1)**Counting Primitive Operations:** Given an algorithms, count the number of primitive operations, and represent it as a function of the input size n : $f(n)$ or $T(n)$. Some common examples of primitive operations:

1. Evaluating an expression, e.g. $a - 5 + c\sqrt{b}$.
2. Assigning a value to a variable, e.g. $a \leftarrow 23$.
3. Indexing into an array, e.g. $a[i]$.
4. Calling a method, e.g. $v.method()$.
5. Returning from a method, e.g. `return a`.

Please note: (i) a primitive operation takes constant time. (ii) Different types of primitive operations may take different constant time in real machine. But we only need to know the number of primitive operations regardless of specific primitive operation. For example

*School of Computer Sciences, 5115 Lambton Tower, University of Windsor, 401 Sunset Avenue, Windsor, Ontario, N9B 3P4, Canada (phone: +1-519-253-3000 Ext.3789; email: {li11112c, angom}@uwindsor.ca).

1. $a[i] \leftarrow 23$ 2.
2. $i \leftarrow i + 1$ 2
3. *if* $a[i] > 5$ 2.
4. *for* $i \leftarrow 0$ *to* $n - 1$ $1(\text{initialization}) + 2n(i++) + (n + 1)(\text{compare}) = 3n + 2.$
5. *for* $i \leftarrow 1$ *to* 0 $1(\text{initialization}) + 1(\text{compare}) = 2.$ i will not increase.

(2) According to $f(n)$, obtain the Bio-Oh notation to represent/measure the time complexity.

Definition of Big-Oh Notation: Given functions $f(n)$ and $g(n)$, we say that $f(n)$ is $O(g(n))$ if there are positive constants c and n_0 such that $f(n) \leq cg(n)$ for $n \geq n_0$.

We can use this definition to justify the time complexity of an algorithms in terms of big-Oh notation. Examples:

(i) $f(n) = 5n + 8$ is $O(n)$. Since $f(n) \leq cg(n) \Rightarrow 5n + 8 \leq cn \Rightarrow n \geq \frac{8}{c-5}$, we can find $c = 6, n_0 = 8$.

(ii) $f(n) = 3n^3 + 20n^2 + 5$ is $O(n^3)$. Since $f(n) \leq cg(n) \Rightarrow 3n^3 + 20n^2 + 5 \leq (3 + 20 + 5)n^3 \Rightarrow 3n^3 + 20n^2 + 5 \leq 28n^3$, we obtain $c = 28$, for any $n \geq 1$ this inequality holds, so $n_0 = 1$.

(iii) $3 \log n + 5$ is $O(\log n)$. Since $f(n) \leq cg(n) \Rightarrow 3 \log n + 5 \leq c \log n \Rightarrow \log n \geq \frac{5}{c-3} \Rightarrow n \geq 2^{\frac{5}{c-3}}$, we can find $c = 8, n_0 = 2$, or $c = 4, n_0 = 32$. Note: there are many other c and n_0 , they are not unique.

Some proposition can help you justify the Big-Oh notation of $f(n)$.

1. If $f(n) = a_0 + a_1n + \dots + a_d n^d$, and $a_d > 0$, then $f(n)$ is $O(n^d)$. For example, $f(n) = 3n^3 + 20n^2 + 5$ is $O(n^3)$.
2. Use the smallest possible class of functions. For example, $f(n) = 5n + 8, f(n) \leq cn, f(n) \leq cn^2, f(n) \leq cn^3$, and \dots , but $f(n)$ is $O(n)$.
3. Use the simplest expression of the class. For example, $f(n) = 5n + 8$ is $O(n)$ instead of $O(5n)$; $f(n) = 2n \log n + 8$ is $O(n \log n)$ instead of $O(2n \log n)$.

4. Lower-order terms in $f(n)$ can be dropped. Always keep in mind that: $1 \leq \log n \leq n \leq n \log n \leq n^2 \leq n^3 \leq e^n$. For example, $f(n) = 3n^2 + 5n \log n + 100n + 4$ is $O(n^2)$.

Example: $f(n) = 3n^3 - 20n^2 + 5n - 2$ is $O(n^3)$. $f(n) \leq 3n^3 + 5n = (3 + 5)n^3 = 8n^3$, so $c = 8, n_0 = 1$.

Examples:

Algorithm 1 Ex1

Input: An array A storing $n \geq 1$ integers.

Output: The sum of the elements in A .

```

1:  $s \leftarrow A[0]$  {2}
2: for  $i \leftarrow 1$  to  $n - 1$  do {3(n-1)+2=3n-1}
3:    $s \leftarrow s + A[i]$  {3(n-1)}
4: end for
5: return  $s$  {1}

```

$f(n)_1 = 2 + 3n - 1 + 3(n - 1) + 1 = 6n - 1$ is $O(n)$.

Algorithm 2 Ex2

Input: An array A storing $n \geq 1$ integers.

Output: The sum of the elements at even cells in A .

```

1:  $s \leftarrow A[0]$  {2}
2: for  $i \leftarrow 2$  to  $n - 1$  by increments of 2 do {3[(n-1)/2] + 2}
3:    $s \leftarrow s + A[i]$  {3[(n-1)/2]}
4: end for
5: return  $s$  {1}

```

$f(n)_2 = 2 + 3[(n - 1)/2] + 2 + 3[(n - 1)/2] + 1$ is $O(n)$.

Algorithm 3 Ex3

Input: An array A storing $n \geq 1$ integers.

Output: The sum of the prefix sums in A .

```

1:  $s \leftarrow 0$  {1}
2: for  $i \leftarrow 0$  to  $n - 1$  do {3n+2}
3:    $s \leftarrow s + A[0]$  {3n}
4:   for  $j \leftarrow 1$  to  $i$  do { $\sum_{i=0}^{n-1} (3i+2) = 3(0+1+2+\dots+n-1) + 2n = 3n(n-1)/2 + 2n$ }
5:      $s \leftarrow s + A[j]$  { $3 \sum_{i=0}^{n-1} (i) = 3(0+1+2+\dots+n-1) = 3n(n-1)/2$ }
6:   end for
7: end for
8: return  $s$  {1}

```

In line 4 of Algorithm 3, the total number of iterations is $\sum_{i=0}^{n-1} i$; the corresponding number of primitive operations is $\sum_{i=0}^{n-1} (3i + 2) = 3n(n - 1)/2 + 2n$. When you encounter a loop or a nested loop, you need to count the number of iterations first, and then use our formula $3(\text{iter}\#) + 2$. Please use this formula in the nested loop correctly. In line 4, the number of primitive operations is $\sum_{i=0}^{n-1} (3i + 2)$, not $3[\sum_{i=0}^{n-1} i] + 2$.

$$f(n)_3 = 1 + 3n + 2 + 3n + 3n(n - 1)/2 + 2n + 3n(n - 1)/2 + 1 \text{ is } O(n^2).$$

Algorithm 4 Ex4

Input: An array A storing $n \geq 1$ integers.

Output: The sum of the prefix sums in A .

```

1:  $s \leftarrow A[0]$  {2}
2:  $t \leftarrow s$  {1}
3: for  $i \leftarrow 1$  to  $n - 1$  do {(3(n-1)+2)}
4:    $s \leftarrow s + A[i]$  {3(n-1)}
5:    $t \leftarrow t + s$  {2(n-1)}
6: end for
7: return  $t$  {1}
```

$$f(n)_4 = 2 + 1 + 3(n - 1) + 2 + 3(n - 1) + 2(n - 1) + 1 \text{ is } O(n).$$

Algorithm 5 Ex5

Input: Arrays A and B each storing $n \geq 1$ integers.

Output: The number of elements in B equal to the sum of prefix sums in A .

```

1:  $c \leftarrow 0$  {1}
2: for  $i \leftarrow 0$  to  $n - 1$  do {3n+2}
3:    $s \leftarrow 0$  {1}
4:   for  $j \leftarrow 0$  to  $n - 1$  do { $\sum_{i=0}^{n-1} (3n + 2) = n(3n + 2)$ .(iteration# is  $\sum_{i=0}^{n-1} n = n^2$ )}
5:      $s \leftarrow s + A[0]$  { $3n^2$ . (number of iterations is  $n^2$ )}
6:     for  $k \leftarrow 1$  to  $j$  do { $\sum_{i=0}^{n-1} \sum_{j=0}^{n-1} (3j + 2) = 3n^2(n - 1)/2 + 2n$ }
7:        $s \leftarrow s + A[k]$  { $3 \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} (j) = 3n^2(n - 1)/2$ }
8:     end for
9:   end for
10:  if  $B[i] = s$  then {2n}
11:     $c \leftarrow c + 1$  {2n}
12:  end if
13: end for
14: return  $c$  {1}
```

In line 6 of Algorithm 5, the number of iterations is $\sum_{i=0}^{n-1} \sum_{j=0}^{n-1} (j)$. When you encounter a loop or a nested loop, you need to count the number of iterations first, and then use

our formula $3(\text{iter}\#) + 2$. Please note that in line 6, the number of primitive operations is $\sum_{i=0}^{n-1} \sum_{j=0}^{n-1} (3j + 2)$, not $3[\sum_{i=0}^{n-1} \sum_{j=0}^{n-1} (j)] + 2$.

$$f(n)_5 = \dots \text{ is } O(n^3).$$

2 Asymptotic Algorithm Analysis

Two Steps:

1. Count the number of primitive operations of each statement, and then sum them to obtain $f(n)$. Because several primitive operations also take constant time totally, hereafter, we assume the constant number of primitive operations to be 1. For example, $a[i] \leftarrow 23$ is assumed to be 1.

1. $a[i] \leftarrow 23$ 1.
2. $i \leftarrow i + 1$ 1.
3. *if* $a[i] > 5$ 1.
4. $temp = a;$ $a = b;$ $b = temp;$ 1.
5. *for* $i \leftarrow 0$ *to* $n - 1$ n .

2. Obtain the big-Oh notation through $f(n)$.

Examples:

Algorithm 6 Ex1

Input: An array A storing $n \geq 1$ integers.

Output: The sum of the elements in A .

- 1: $s \leftarrow A[0]$ {1}
 - 2: **for** $i \leftarrow 1$ **to** $n - 1$ **do** {n-1}
 - 3: $s \leftarrow s + A[i]$ {n-1}
 - 4: **end for**
 - 5: **return** s {1}
-

$$f(n)_1 = 2n \text{ is } O(n).$$

$$f(n)_2 = n + 1 \text{ is } O(n).$$

Algorithm 7 Ex2

Input: An array A storing $n \geq 1$ integers.

Output: The sum of the elements at even cells in A .

```
1:  $s \leftarrow A[0]$  {1}
2: for  $i \leftarrow 2$  to  $n - 1$  by increments of 2 do  $\{(n-1)/2\}$ 
3:    $s \leftarrow s + A[i]$   $\{(n-1)/2\}$ 
4: end for
5: return  $s$  {1}
```

Algorithm 8 Ex3

Input: An array A storing $n \geq 1$ integers.

Output: The sum of the prefix sums in A .

```
1:  $s \leftarrow 0$  {1}
2: for  $i \leftarrow 0$  to  $n - 1$  do {n}
3:    $s \leftarrow s + A[0]$  {n}
4:   for  $j \leftarrow 1$  to  $i$  do  $\{\sum_{i=0}^{n-1} i = 0+1+2+ \dots +n-1 = n(n-1)/2\}$ 
5:      $s \leftarrow s + A[j]$   $\{0+1+2+ \dots +n-1 = n(n-1)/2\}$ 
6:   end for
7: end for
8: return  $s$  {1}
```

$$f(n)_3 = n^2 + n + 2 \text{ is } O(n^2).$$

Algorithm 9 Ex4

Input: An array A storing $n \geq 1$ integers.

Output: The sum of the prefix sums in A .

```
1:  $s \leftarrow A[0]$  {1}
2:  $t \leftarrow s$  {1}
3: for  $i \leftarrow 1$  to  $n - 1$  do {n-1}
4:    $s \leftarrow s + A[i]$  {n-1}
5:    $t \leftarrow t + s$  {n-1}
6: end for
7: return  $t$  {1}
```

$$f(n)_4 = 3n \text{ is } O(n).$$

$$f(n)_5 = n^3 + n^2 + 3n + 3 \text{ is } O(n^3).$$

3 Application of Stack

1. Reverse Polish Notation/ Postfix Expression Evaluation

(1) First of all, we need to convert a regular expression into a postfix expression. We

Algorithm 10 Ex5

Input: Arrays A and B each storing $n \geq 1$ integers.

Output: The number of elements in B equal to the sum of prefix sums in A .

```
1:  $c \leftarrow 0$  {1}
2: for  $i \leftarrow 0$  to  $n - 1$  do {n}
3:    $s \leftarrow 0$  {1}
4:   for  $j \leftarrow 0$  to  $n - 1$  do { $n^2$ }
5:      $s \leftarrow s + A[0]$  { $n^2$ }
6:     for  $k \leftarrow 1$  to  $j$  do { $\sum_{i=0}^{n-1} \sum_{j=0}^{n-1} (j) = \sum_{i=0}^{n-1} [1 + 2 + \dots + (n - 1)] = n^2(n - 1)/2$ }
7:        $s \leftarrow s + A[k]$  { $n^2(n - 1)/2$ }
8:     end for
9:   end for
10:  if  $B[i] = s$  then {n}
11:     $c \leftarrow c + 1$  {n}
12:  end if
13: end for
14: return  $c$  {1}
```

can convert a regular expression, e.g. $(1 + 2) \times 5 - 6$, to a postfix expression: $12 + 5 \times 6 -$. Another example, $6 - 5 \times (1 + 2)$ can be converted into $6512 + \times -$. The expression *operand1operatoroperand2* can be converted into *operand1operand2operator*. If the *operand1* or/and *operand2* contains the other sub-expression, do the same thing as the above.

(2) Secondly, we can use stack to evaluate the postfix expression. The algorithm is that: Scan the postfix expression element by element from the beginning to the end. If the current element is an operand then push it in the stack. If the current element is an operator, then pop the top element in the stack as operand 2, and then pop the top element again in the stack as operand 1, then calculate the result of *operand1operatoroperand2*. Push this result in the stack and continue scanning the postfix expression.

2. Parentheses Matching Algorithm Please analyze the time complexity of the algorithm in page 13 of Stack slices as practice.

Quiz of Lab 1

Yifeng Li and Alioune Ngom *

1 Quiz

Count the primitive operations, and give a big-Oh characterization, in terms of n , of the running time of the following algorithm. (10 marks)

Algorithm 1 Quiz1: A simple quadratic algorithm for detecting duplicates.

Input: An array A storing $n \geq 1$ integers.

Output: Return true if array A has duplicates; false otherwise.

```
1: for  $i \leftarrow 0$  to  $n - 1$  do  $\{3n+2\}$ 
2:   for  $j \leftarrow i + 1$  to  $n - 1$  do  $\{3n^2/2 + n/2\}$ 
3:     if  $A[i] = A[j]$  then  $\{3 \sum_{i=0}^{n-1} (n - 1 - i) = 3n^2 - 3n(n - 1)/2 - 3n\}$ 
4:       return true  $\{1\}$ 
5:     end if
6:   end for
7: end for
8: return false  $\{1\}$ 
```

In line: 2, the number of iterations is $\sum_{i=0}^{n-1} (n - 1 - i)$, so the number of primitive operations is $\sum_{i=0}^{n-1} [3(n - 1 - i) + 2] = \sum_{i=0}^{n-1} [3n - 3i - 1] = 3 \sum_{i=0}^{n-1} n - 3 \sum_{i=0}^{n-1} i - \sum_{i=0}^{n-1} 1 = 3n^2 - 3n(n - 1)/2 - n = 3n^2/2 + n/2$.

$f(n) = 3n + 2 + 3n^2/2 + n/2 + 3n^2 - 3n(n - 1)/2 - 3n + 1 = 3n^2 + 2n + 3$ is $O(n^2)$. Because we count the number of primitive operations in worst case, we do not add the number of primitive operations in line 4.

Justification: $f(n) \leq 3n^2 + 2n^2 + 3n^2 = (3 + 2 + 3)n^2$. We can find $c = 8$ and $n_0 = 1$.

*School of Computer Sciences, 5115 Lambton Tower, University of Windsor, 401 Sunset Avenue, Windsor, Ontario, N9B 3P4, Canada (phone: +1-519-253-3000 Ext.3789; email: {li11112c, angom}@uwindsor.ca).