Algorithm Analysis

Lecture 2

Lecture 2

https://livecodestream.dev/post/complete-guide-to-understanding-time-and-space-complexity-of-algorithms/ Algorithm Complexity

Recap Lecture 1

Data structures has wide applications •

Design an **efficient** search engine, as good as Google's • Build an **efficient** security system based on face recognition • Understand the human genome and trace your ancestry

Data structures are a key for designin **ıs**

[redacted]

Recap Lecture 1

• Independent of Hardware, OS, and compiler

• Consider each operation (comparison, assignment) as a program step

• To keep stuff simple we consider each operation cost is 1 unit of time

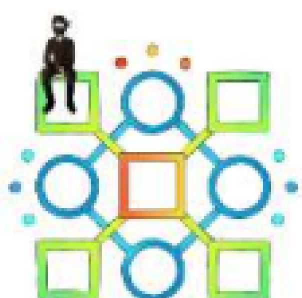
A sequence of operations:

`count = count + 1;` 1 unit of time

`sum = sum + count;` 1 unit of time

→ Total Cost = 2

Don't count



comments and declarations

Estimate the performance of an algorithm through

The **number of operations** required to process an **input of certain size**

`i=0;`	**1**
`sum = 0;`	**1**
`while(i<N ){`	**N+1**
`sum ++;`	**N**
`i++`	**N**

}	

ndition always
s one more time
the loop itself

$$T(N) = 1 + 1 + N+1 + N + N$$

$$T(N) = 3N+3$$

https://www.ft.com/content/d24fce5e-58fc-11e6-9f70-badea1b336d4

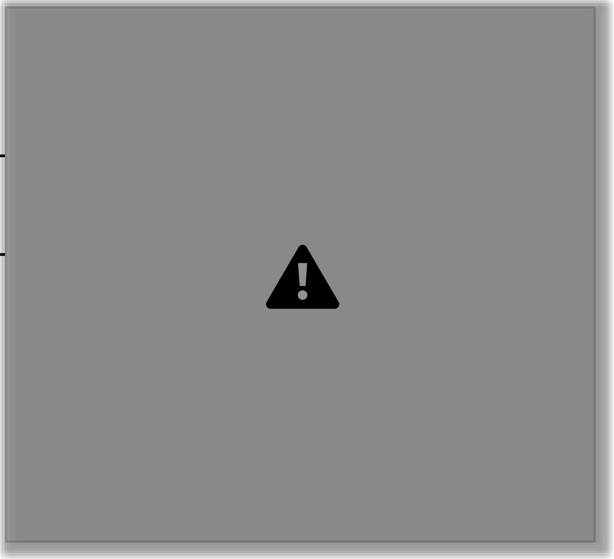
For student

Chapter 2 of Alan Weiss’s book

Chapter 2 of Adam Drozdek’s book

`i=0;`	**1**
`sum = 0;`	**1**
`while(i<N ){`	**N+1**
`sum ++;`	**N**
`i++`	**N**

}	



```
T(N) = 1 + 1 + N+1 + N + N
T(N) = 3N+3
```

TWO important things to note

- What is the effect of constant in *T(N)* = 3N + 3 ***?***
- How T(n) varies for different N (input) ?
  - Check for N =5, 10, 20, 50, 100, 500, 1000, 10000, 100000

8

Analysis of Loop

70000 60000 50000 40000 30000 20000 10000 0

N

3N+3

-5000 0 5000 10000 15000 20000 25000 -10000

	T(N)= 3N+3
10	33
20	63
30	93
40	123
50	153
100	303
150	453
200	603
5000	15003
10000	30003
15000	45003
20000	60003

Example: Nested Loop

<u>Cost</u> <u>Times</u>

```
i=1; 1 1 sum = 0; 1 1 while (i <= n) { 1 n+1
j=1; 1 n
while (j <= n) { 1 n*(n+1) sum += i; 1 n*n
```

```
j++; 1 n*n
}
i++; 1 n
}
```

T(N) = 1 + 1 + (n+1) + n + n*(n+1)+n*n+ n*n+ n = $\mathbf{3n^2 + 3n + 3}$

→ The time required for this algorithm is **proportional to n^2**

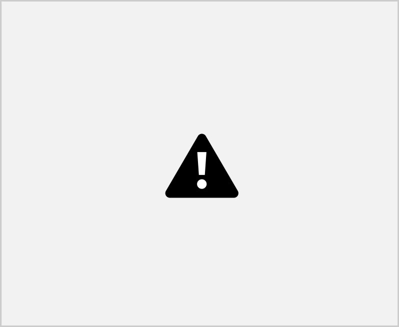
Analysis of Nested Loop

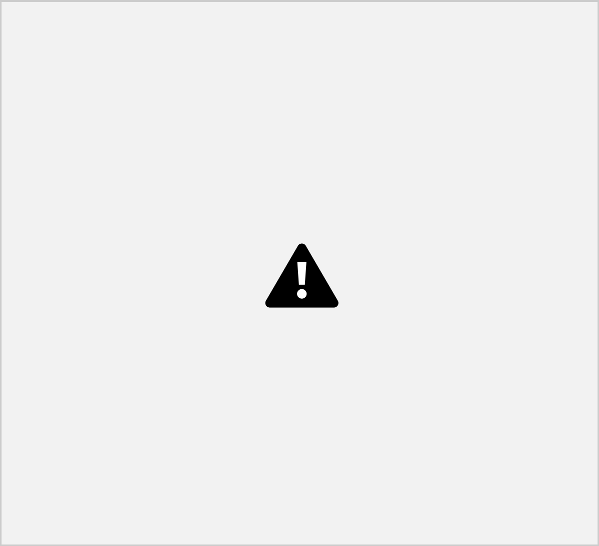
- **2 important things to Note**
  - What is the effect of constant and low order term in

in $T(N) = 3N^2 + 3N+3$ *for large N ?*

– How T(n) varies for different N (input) ?

- Check for N =5, 10, 20,50,100, 500, 1000, 10000 ,100000





Example:

- If **T(n) = 7n+100**
- What is T(n) for different values of n???

n T(n) Comment

107

1 Contributing factor is 100

Example:

- If **$T(n) = 7n+100$**
- What is $T(n)$ for different values of n???

n T(n) Comment

107
135
170

1 Contributing factor is 100 5 Contributing factor is 7n and 100 10 Contributing factor is 7n and 100

Example:

- If **$T(n) = 7n+100$**
- What is T(n) for different values of n???

n T(n) Comment

107

135
170
800

1 Contributing factor is 100 5 Contributing factor is 7n and 100 10 Contributing factor is 7n and 100 100 Contribution of 100 is small

Example:

- If **T(n) = 7n+100**
- What is T(n) for different values of n???

n T(n) Comment

107
135

170
800
7100

1 Contributing factor is 100 5 Contributing factor is 7n and 100 10 Contributing factor is 7n and 100 100 Contribution of 100 is small 1000 Contributing factor is 7n

Example:

- If **T(n) = 7n+100**
- What is T(n) for different values of n???

n T(n) Comment

107
135
170
800
7100
70100

1 Contributing factor is 100 5 Contributing factor is 7n and

100 10 Contributing factor is 7n and 100 100 Contribution of 100 is small 1000 Contributing factor is 7n 10000 Contributing factor is 7n

Example:

- If **T(n) = 7n+100**
- What is T(n) for different values of n???

n T(n) Comment

107

135
170
800
7100
70100

1 Contributing factor is 100

5 Contributing factor is 7n and 100 10 Contributing factor is 7n and 100 100 Contribution of 100 is small

1000 Contributing factor is 7n

10000 Contributing factor is 7n

10^6 7000100 What is the contributing factor????

***DEDUCTION**: When approximating T(n) we can IGNORE the 100 term for very large value of n and say that T(n) can be approximated by 7(n)*

$T(N)=cN$ and $T(N)=N$

- **Note** we are estimating Algorithm complexity with reference to size of the input
- $T_1(N)=N$ and $T_2(N)=30N$ will have same effect with increase in input size
- Effect of input size on above T(N)

10	$t_1=10$	$t_2=300$

T(N)=cN and T(N)=N

- **Note** we are estimating Algorithm complexity with reference to size of the input
- $T_1(N)=N$ and $T_2(N)=30N$ will have same effect with increase in input size
- Effect of input size on above T(N)

10	t_1=10	t_2=300
20	$2t_1$	$2t_2$

T(N)=cN and T(N)=N

- **Note** we are estimating Algorithm complexity with reference to size of the input
- $T_1(N)=N$ and $T_2(N)=30N$ will have same effect with increase in input size

- Effect of input size on above T(N)

10	t_1=10	t_2=300
20	$2t_1$	$2t_2$
100	$10t_1$	$10t_2$
1000	$100t_1$	$100t_2$

Example 2

$$T(n) = n^2 + 100n + \log_{10}n + 1000$$

n T(n) n^2 100n $\log_{10}$n 1000

	Val	%	Val	%	Val	%	Val
1101	1	0.1%	100	9.1%	0	0%	1000

%

1 90.8%

Example 2

$$\mathbf{T(n) = n^2 + 100n + log_{10}n + 1000}$$

n T(n) n^2 100n $log_{10}n$ 1000

	Val	%	Val	%	Val	%	Val

1101	1	0.1%	100	9.1%	0	0%	1000
2101	100	5.8%	1000	47.6%	1	0.05%	1000

%

1 90.8% 10 47.6%

Example 2

$$T(n) = n^2 + 100n + \log_{10}n + 1000$$

n T(n) n^2 100n $\log_{10}n$ 1000

	Val	%	Val	%	Val	%	Val
1101	1	0.1%	100	9.1%	0	0%	1000
2101	100	5.8%	1000	47.6%	1	0.05%	1000
21002	10000	47.6%	10000	47.6%	2	0.99%	1000

%

1 90.8% 10 47.6% 100 4.76%

Example 2

$$T(n) = n^2 + 100n + \log_{10}n + 1000$$

n T(n) n^2 100n $\log_{10}n$ 1000

	Val	%	Val	%	Val	%	Val
1101	1	0.1%	100	9.1%	0	0%	1000
2101	100	5.8%	1000	47.6%	1	0.05%	1000

21002	10000	47.6%	10000	47.6%	2	0.99%	1000

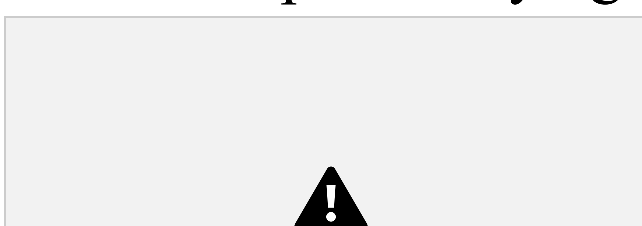
%

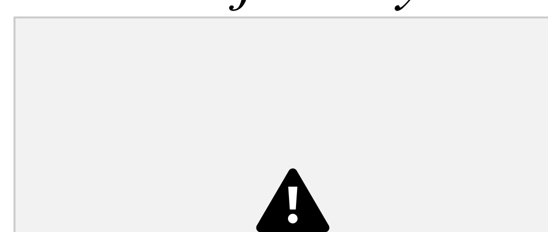
1 90.8% 10 47.6% 100 4.76% 10^5 10,010,001,005 10^{10} 99.9% 10^7.099% 5 0.0% 1000 0.00%

When approximating T(n) we can ***IGNORE the last 3 terms*** *and say that T(n) can be approximated by* n^2

24 **Rate of Growth**

• Consider the example of buying *Gold and Metal jewelry* •





+

Cost: cost_of_gold + cost_of_metal

Cost ~ cost_of_gold (approximation)

Rate of Growth

- The low order terms in a function are relatively insignificant for **large** n

$n^4 + 100n^2 + 10n + 50 \sim n^4$

i.e., we say that $n^4 + 100n^2 + 10n + 50$ and n^4 have the same **rate of growth**

Partial SUM

```
int sum( int n )
{
    int partialSum;
    1 partialSum = 0;
    2 for( int i = 1; i <=
    n;  ++i )


   3 partialSum += i * i * i;

    4 return partialSum;
}
```

Cost

$$T(N)= 6N+4$$

If we had to perform all this work everytime we needed to analyze a program, the task would quickly **become infeasible**.

Partial SUM

`int sum( int n )`	**Cost**
`{`	0
`int partialSum;`	
`1 partialSum = 0;`	
`2 for( int i = 1; i <= n; ++i )`	1 i=1 cost is 1, i<=N cost is N+1 ++i cost is N
`3 partialSum += i * i * i;`	
`4 return partialSum;`	
`}`	

T(N)= 6N+4

If we had to perform all this work everytime we needed to analyze a program, the task would quickly **become infeasible**.

Partial SUM

	Cost
`int sum( int n )`	
`{`	
`int partialSum;`	0
`1 partialSum = 0;`	1
`2 for( int i = 1; i <= n; ++i )`	i=1 cost is 1, i<=N cost is N+1 ++i cost is N
`3 partialSum += i * i * i;`	3 ops 1 assignment, 2 multiplication, 1 add Total 4 cost for line 3
`4 return partialSum;`	0
`}`	

T(N)= 6N+4

If we had to perform all this work everytime we needed to analyze a program, the task would quickly **become infeasible**.

Asymptotic complexity studies the efficiency of an algorithm as the input size becomes large

Big-Oh or Big-O

f(n)* is *O(g(n)) *if there exist positive numbers* **c & n_0** *such that* ***f(n)<=cg(n)*** *for all* $n >= n_0$

g(n) is called the upper bound on f(n) OR
f(n) grows at the most as large as g(n)

Big-Oh or Big-O

$f(n)$ is $O(g(n))$ if there exist positive numbers c & n_0 such that $f(n) <= cg(n)$ for all $n >= n_0$

g(n) is called the upper bound on f(n) OR

f(n) grows at the most as large as g(n)

Example:

$T(n) = n^2 + 3n + 4$

$n^2 + 3n + 4 <= 2n^2$ for all $n_0 > 10$

What is f(n) and what is g(n)?

What is c & n

so we can say that T(n) is $O(n^2)$ OR

T(n) is in the order of n^2.

T(n) is bounded above by a + real multiple of n^2

How to choose c and N

What is c & N

- **The definition of big-Oh states only that there must exist certain *c* and *N*, but it <u>does not give any hint of how to calculate these constants.</u>**
- **Second, it does not put any restrictions on these values and gives little guidance in situations when there are many candidates.**

- In fact, there are usually **infinitely many pairs of *cs* and *Ns*** that can be given for the same pair of functions *f* and *g*.

How to choose c and N

We obtain the value of c & n by solving the inequality

Put value of n=1,2,3, in this equation to get value of c for that n

Because it is one inequality with two unknowns, different pairs of

constants c and n for the same function g(n^2) can be determined.

For a fixed g, an infinite number of pairs of c's and n's can be identified.

How to choose c and N

Here N is same as n_0

35 More Examples

- Show that $3n+3$ is $O(n)$.

– Show $\exists c, n_0$: $3n+3 \le cn, \forall n > n_0$.

- $3+3/n <= c$ ….
- $n_0=1$ $c>=6$
- $n_0=2$ $c>=4.5$ so on

- Show that $2n+2$ is $O(n)$.

– $n_0=1$, $c>=4$

– $c=3$, $n_0=2$.

36

More Examples

- Show that $2n^2 + 2n+2$ is $O(n^2)$. – Hold if we let $c=6$, $n_0=1$.

– Hold if we let $c=5$, $n_0=2$

– Hold if we let $c=4$ $n_0=2$

- Show that $3n^2 + 3n+3$ is $O(n^2)$. – Hold if we

let $c=9$, $n_0=1$.

– Hold if we let $c=7$, $n_0=3$

How to choose c and N

Here N is same as n_0 The crux of the matter is that the value of *c* depends on which *N* is chosen, and vice versa.

To choose the best *c* and *N,* it should be determined for which *N* a certain term in *f* becomes the largest and stays the largest.

In Equation 2.2, the only candidates for the largest term are $2n^2$ and $3n$; these terms can be compared using the inequality $2n^2 > 3n$ that holds for $n > 1.5$.

Thus, $N = 2$ and $c \geq 3\}$

How to choose c and N

- The point is that *f* and *g* grow at the same rate.
- g is almost always greater than or equal to f if it is multiplied by a constant c.

Big-O example, graphically

Show that $30n+8$ is $O(n)$.

Show $\exists c, n_0$: $30n+8 \le cn$, $\forall n > n_0$.

Let $c=31$, $n_0=8$. Assume $n>n_0=8$. Then

$cn = 31n = 30n + n > 30n+8$, so $30n+8 < cn$.

$30n+8$

n

$\in O(n)$

- Note $30n+8$ isn't less than n *anywhere* ($n>0$).
- It isn't even less than $31n$ *everywhere*.

$cn = 31n$

- But it *is* less than $31n$ everywhere to

$30n+8$

the right of n=8. $n>n_0=8$ →

Increasing n →

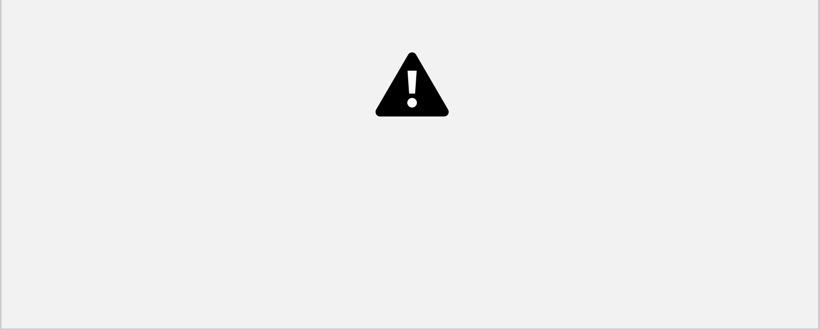
Example

- For $n^2 - 3n + 10$ Find constants c and n_0 exist such that $cn^2 > n^2 - 3n + 10$ for all $n \geq n_0$.

c is 3 and n_0 is 2

$3n^2 > n^2 - 3n + 10$ for all $n \geq 2$.

Thus, the algorithm requires no more than kn^2 time

 is $\mathbf{O(n^2)}$

Big Oh Example

- There is no unique set of values for n_0 and c in proving the asymptotic bounds

- Prove that $100n + 5 = O(n^2)$

- $100n + 5 \le 100n + n = 101n \le 101n^2$

for all $n \ge 5$

$n_0 = 5$ and $c = 101$ is a solution

- $100n + 5 \le 100n + 5n = 105n \le 105n^2$

for all $n \ge 1$

$n_0 = 1$ and $c = 105$ is also a solution

Must find **SOME** constants c and n_0 that satisfy the asymptotic notation relation

Nested For Loops

- **For Loop** *can be confusing as three statements are embedded in one …As T(N) is rough estimate, so some analysis use 1 for entire for loop …1 cost for each line or step*

sum = 0; O(1)

for(i=0; i<N; i++) O(N)

for(j=0; j<N; j++) ��(��^{2})

sum += arr[i][j]; O(N^2)

$$T(N) \approx O(1) + O(N) + O(N^2) + O(N^2) = O(N^2)$$

*The total number of times a statement executes = outer loop times * inner loop times*

44

Nested For Loops

Rough estimate

for(i=0; i<N; i++) O(N) arr[i][i] =0; O(N)

sum = 0; O(1) for(i=0; i<N; i++) O(N)

for(j=0; j<N; j++) $O(N^2)$ sum += arr[i][j];

$O(N)^2$

$O(N)+O(N^2) = O(N^2)$