

Time complexity of loops

Alexandra Stefan

Review log and exponent and closed form (solutions) for common summations

$$x^p = N \Rightarrow p = \log_x N \text{ (apply } \log_x \text{ on both sides)}$$
$$\log_a N = \frac{\log_b N}{\log_b a}$$

The **closed form** is the solution for the summation. It is an expression that is equivalent to the summation, but does NOT have any \sum in it.

We need the closed form in order to find Θ for that summation.

$$\sum_{k=1}^N k = \frac{N(N+1)}{2} = \Theta(N^2)$$
$$\sum_{k=1}^N k^2 = \frac{N(N+1)(2N+1)}{6} = \Theta(N^3)$$
$$\sum_{k=1}^N x^k = \frac{x^{N+1} - 1}{x - 1} = \Theta(x^N) \text{ when } x > 1; \text{ for } x < 1 \text{ it is } \Theta(1)$$

Review

Techniques for solving summations (useful for Θ or dominant term calculations)

Note that some of these will NOT compute the EXACT solution for the summation

Independent case (term in summation does not have the variable of the summation).

$$\sum_{k=1}^N \mathbf{S} = S + S + S + \dots + S = \mathbf{NS} \quad (= S \sum_{k=1}^N 1 = SN)$$

Pull constant in front of summation: $\sum_{k=1}^N (Sk) = S \sum_{k=1}^N k = S \frac{N(N+1)}{2} = \Theta(SN^2)$

Break summation in two summations

$$\sum_{k=1}^N (kS + k^2) = \sum_{k=1}^N kS + \sum_{k=1}^N k^2 = S \sum_{k=1}^N k + \sum_{k=1}^N k^2 = S \frac{N(N+1)}{2} + \frac{N(N+1)(2N+1)}{6} = \Theta(SN^2 + N^3)$$

Drop lower order term from summation term. E. g. $10k$ is lower order compared to k^2 :

$$\sum_{k=1}^N (10k + k^2) = \sum_{k=1}^N k^2 = \frac{N(N+1)(2N+1)}{6} = \Theta(N^3)$$

Use approximation by integrals for increasing or decreasing $f(k)$ – to be covered later

$$\sum_S^N f(k) = \Theta(F(N) - F(S)) \quad (\text{where } F \text{ is the antiderivative of } f)$$

Notations, Conventions, Reminders

- $\Theta(1)$ – We use $\Theta(1)$ when the number of instructions executed does NOT depend on the problem size. They are a fixed number (e.g. 9 instructions)
- Notation:
 - TC_{1iter} = time complexity (as Θ) for all instructions executed in 1 iteration of the loop (fct calls included)
 - $lg = \log_2$ *e.g.* $lgN = \log_2 N$
- The general method (that uses summation) handles all loop cases correct.
- When computing the number of loop repetitions we will use the approximate number given by the dominant term, and not worry if we are off by ± 1 or \pm constant. E.g. we will use N instead of $N+1$ or $N-1$ and we will use $\lfloor \log_2 N \rfloor$ instead of $1 + \lfloor \log_2 N \rfloor$
- Use the correct time complexity for function CALLS based on the time complexity (of the code in the definition) and the arguments passed when called. E.g.

function declaration/header:

```
void linear_search(int num1[], int T, int val) // has time complexity  $\Theta(T)$ ,
```

when called with M for T , e.g.

```
res = linear_search(nums1, M, 7); // has time complexity  $\Theta(M)$ 
```

Change of variable

```
for(i=1; i<=N; i=i+1){ // i takes consecutive values
...
}

for(i=0; i<=N; i=i+5){ // i does NOT take consecutive values
... // $\Theta(i)$  code
}

for(i=1; i<=N; i=i*2){ // i does NOT take consecutive values
... // $\Theta(i)$  code
}
```

When the variable in a loop does not take consecutive values, it becomes harder to calculate the following (and be confident in your answer):

1. How many iterations the loop does
2. The time complexity for the entire loop

We will use a change of variable to see how the original loop variable depend on another value that DOES take consecutive values.

Another way to look at this. Most common loops that appear in code, generate values from either an arithmetic series or a geometric one. With the change of variable we identify the series.

Change of variable – Math/programming

Arithmetic Series – easy case

```
for (i=0; i<=N; i=i+5) {  
  ...  
}
```

Values of i: 0, 5, 10, 15, 20, ..., i, ... $i_{\text{last}} \leq N$
 $5*0, 5*1, 5*2, 5*3, 5*4, \dots, 5e, \dots, 5p$

Values of e: 0, 1, 2, 3, 4, ..., e, ..., p

$$\Rightarrow i = 5e$$

$$\Rightarrow 5p = i_{\text{last}} = N \quad (\text{we used } = \text{ instead of } \leq \text{ to make math easy})$$

$$\Rightarrow p = N/5$$

The table below has ONE ROW for EACH ITERATION of the loop.

It shows:

- the value of i at that iteration
- the expression (pattern) of i (as a function of a variable e that takes consecutive values)
- what values e takes

e		i
0	$5*0$	0
1	$5*1$	5
2	$5*2$	10
3	$5*3$	15
...
e	$5*e$	i
...
p	$5*p$	$i_{\text{last}} \leq N$

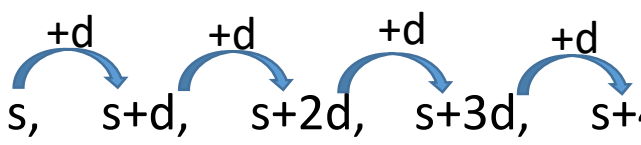
Change of variable – Math/programming

Arithmetic Series – general case

```
for (i=s; i<=(3*N+7) ; i=i+d) {  
  ...  
}
```

The table below has ONE ROW for EACH ITERATION of the loop.
It shows:

- the value of i at that iteration as as a function of a variable e that takes consecutive values
- what values e takes



Values of i: $s, s+d, s+2d, s+3d, s+4d, \dots, s+e*d, \dots s+e*p=i_{\text{last}} \leq (3N+7)$

Values of e: $0, 1, 2, 3, 4, \dots, e, \dots, p$

$\Rightarrow i = s+e*d$

$\Rightarrow s+p*d = i_{\text{last}} = (3N+7)$ (we used use = instead of \leq to make math easy)

$\Rightarrow p = (3N+7-s)/d$

e	i
0	s
1	s+d
2	s+2d
3	s+3d
...	...
e	i
...	...
p	$i_{\text{last}} = s+dp \leq (3N+7)$

Change of variable – Math/programming

Geometric series – easy case

```
for (i=1; i<=N; i=i*2) {  
  ...  
}
```

Values of i: 1, 2, 4, 8, 16, ..., i, ... i_{last} ≤ N

2⁰, 2¹, 2², 2³, 2⁴, ..., 2^e, ..., 2^p

Values of e: 0, 1, 2, 3, 4, ..., e, ..., p

⇒ i = 2^e

⇒ 2^p = i_{last} = N (we used use = instead of ≤ to make math easy)

⇒ p = log₂N

The table below has ONE ROW for EACH ITERATION of the loop.
It shows:

- the value of i at that iteration
- the expression (pattern) of i (as a function of a variable e that takes consecutive values)
- what values e takes

e		i
0	2 ⁰	s
1	2 ¹	2s
2	2 ²	4s
3	2 ³	8s
...
e	2 ^e	i
...
p	2 ^p	i _{last} ≤ N

Change of variable – Math/programming

Geometric series – general case

```
for (i=s; i<= (5*N-3) ; i=i*d) {  
  ...  
}
```

The table below has ONE ROW for EACH ITERATION of the loop.
It shows:

- the value of i at that iteration
- the expression (pattern) of i (as a function of a variable e that takes consecutive values)
- what values e takes

Values of i: $s \cdot d^0, s \cdot d^1, s \cdot d^2, s \cdot d^3, s \cdot d^4, \dots, s \cdot d^e, \dots, s \cdot d^p = i_{\text{last}} \leq (5N-3)$

Values of e: 0, 1, 2, 3, 4, ..., e, ..., p

$\Rightarrow i = s \cdot d^e$

$\Rightarrow s \cdot d^p = i_{\text{last}} = (5N - 3)$ (we used use = instead of \leq to make math easy)

$\Rightarrow p = \log_d \frac{5N-3}{s}$

e	i=
0	$s \cdot d^0$
1	$s \cdot d^1$
2	$s \cdot d^2$
3	$s \cdot d^3$
...	...
e	$s \cdot d^e$
...	...
p	$s \cdot d^p = i_{\text{last}} \leq (5N-3)$

The GENERAL steps for computing the time complexity of loops:

1. Compute TC_{1iter} : the Time Complexity of all the instructions executed in ONE iteration of the loop.
2. Write a “loose” summation over the loop variable of the time complexity of the body (e.g. $\sum_k TC_{1iter}$)
3. Change of variable step. The summation must be over CONSECUTIVE values. If it is not, a change of variable is needed. Does the loop variable (say k) go in consecutive values?
 1. Yes. Write the summation explicitly. E.g. $\sum_{k=1}^N TC_{1iter}$
 2. No. DO change of variable: $k = f(e)$ where e goes from 0 (or 1) to p in consecutive values. Use loop condition to solve for p (note that even if you have $k < N$, you can use $k_{last} = f(p) = N$ since we do not need the exact count). Rewrite the summation using u in sigma, the expression you got for p and replacing k with f(e) in the term. (E.g. if $k = 4e$ and $k < N$, solve: $4p = N \Rightarrow p = N/4$, then do the change of variable:
$$\sum_k k^2 = \sum_{e=0}^{N/4} (4e)^2$$
 (Notice that p does not show in the final answer))
4. Solve the summation. Give the closed form. If a closed form is not possible compute bounds.
5. Give Θ (or O and/or Ω). (Keep the dominant term with the multiplication constant if needed.)

** If this loop was nested in another, the answer from step 5 will be used to compute the time complexity of the body of that immediately outer loop (step 1 in calculations for that outer loop) , and the process continues.

The Four Cases of Time Complexity Calculations for Loops

Case 1: independent + NO change of variable

```
for(i=1; i<=N; i=i+1) {  
    res = linear_search(nums1,M,7); // Θ(M)  
    printf("index = %d\n", res);  
}
```

1. $TC_{1iter}(i) = \Theta(M)$
 2. $\sum_i \Theta(M) = \sum_i M$ (TC_{1iter} independent of loop variable i)
 3. Change of variable is not needed (i takes consecutive values: 1,2,3,...N) \Rightarrow N loop repetitions
 4. General solution: $\sum_i M = \sum_{i=1}^N M = MN$
- OR:
- Because at step 2 we found that TC_{1iter} is independent of i , instead of summation, \sum , we can use:
- (loop repetitions) * $TC_{1iter} \Rightarrow N * \Theta(M) = \Theta(MN)$
5. $\Theta(MN)$ ****This answer cannot have i or T in it**

Assume that the function `linear_search(int num1[], int T, int val)` has time complexity $\Theta(T)$, with detailed count: $7 * T + 9$. When called with M for T it becomes $7*M+9 = \Theta(M)$

Iteration of loop		i	$TC_{1instr} = \Theta(M)$	Sample detailed count:
1 st		1	M	7M + 12
2 nd		2	M	7M + 12
3 rd		3	M	7M + 12
...	
i th		i	M	7M + 12
...	
(N-1) th		N-1	M	7M + 12
N th		N	M	7M + 12

Total: $M+M+M+\dots+M+\dots M = N * M = NM$
Adding all the terms in the TC_{1iter} column gives the time complexity for all the code.

NM is the final result.
Common error: take this result (NM) and multiply it by N again. WRONG!

Check that when adding the detailed count, we get the same time complexity

Case 2: dependent + NO change of variable

```
for(i=1; i<=N; i=i+1) {
    res = linear_search(nums1, i, 7); // Θ(i)
    printf("index = %d\n", res);
}
```

- 1. $TC_{1iter}(i) = \Theta(i)$
- 2. $\sum_i \Theta(i) = \sum_i i$ (TC_{1iter} is dependent on loop variable i)
- 3. Change of variable is NOT needed. (Values of i are consecutive: 1,2,3,...,N)
- 4. General solution: $\sum_i i = \sum_{i=1}^N i = 1 + 2 + 3 + \dots + N = \frac{N(N+1)}{2} = \frac{N^2}{2} + \frac{N}{2}$
- 5. $\Theta(N^2)$ ****This answer cannot have i or T in it**

Iteration of loop	i	$TC_{1instr} = \Theta(i)$	Sample detailed count:
1 st	1	1	$7*1 + 12$
2 nd	2	2	$7*2 + 12$
3 rd	3	3	$7*3 + 12$
...
i th	i	i	$7*i + 12$
...
(N-1) th	N-1	N-1	$7*(N-1) + 12$
N th	N	N	$7*N + 12$

Total: $1+2+3+\dots+i+\dots+N = N*(N+1)/2 = \Theta(N^2)$

Adding all the terms in the TC_{1iter} column gives the time complexity for all the code. Common error: take this result and multiply it by N again. WRONG!

Check that when adding the detailed count, we get the same time complexity

Assume that the function `linear_search(int num1[], int T, int val)` has time complexity $\Theta(T)$, with detailed count: $7 * T + 9$. When called with i for T it becomes $7*i+9 = \Theta(i)$

Case 3: independent + change of variable

```
for(i=1; i<=N; i=i*2){
    res = linear_search(nums1,M,7); // Θ(M)
    printf("index = %d\n", res);
}
```

1. $TC_{1iter}(i) = \Theta(M)$
2. $\sum_i \Theta(M) = \sum_i M$ (TC_{1iter} independent of loop variable i)
3. Change of variable: needed. (Values of i are NOT consecutive.)

Values of i : 1, 2, 4, 8, 16, ..., i , ... $i_{last} \leq N$

$i = 2^e$;

$2^p = i_{last} \leq N$. Use $2^p = N \Rightarrow p = \log_2 N \Rightarrow$ about $\approx \log_2 N$ loop repetitions (there are $1+p$ repetitions, but we can use the dominant term, p , and $p = \log_2 N$, thus it does about

4. General solution: $\sum_i M = \sum_{e=0}^p M = (1+p)M = pM + M = M \log_2 N + M$

OR:

Because at step 2 we found that TC_{1iter} is independent of i , instead of summation, \sum , we can use: (loop repetitions) * $TC_{1iter} \Rightarrow (1 + \log_2 N) * \Theta(M) = \Theta(M \log_2 N)$

5. $\Theta(M \log_2 N)$ ****This answer cannot have i , T , e , or p in it**

Assume that the function `linear_search(int num1[], int T, int val)` has time complexity $\Theta(T)$, with detailed count: $7 * T + 9$. When called with M for T it becomes $7 * M + 9 = \Theta(M)$

Iteration of loop	e	i	$TC_{1instr} = \Theta(M)$	Sample detailed count:
1 st	0	1 ($=2^0$)	M	$7 * M + 12$
2 nd	1	2 ($=2^1$)	M	$7 * M + 12$
3 rd	2	4 ($=2^2$)	M	$7 * M + 12$
...	
	e	i ($=2^e$)	M	$7 * M + 12$
...	
p th	p-1		M	$7 * M + 12$
last (p+1) th	p = $\log_2 N$	$i_{last} (=2^p)$ $2^p = i_{last} \leq N$	M	$7 * M + 12$

Total: $M + M + M + \dots + M + \dots + M = M * \log_2 N = \Theta(M * \log_2 N)$
Adding all the terms in the TC_{1iter} column gives the time complexity for all the code.
Common error: take this result and multiply it by N or $\log_2 N$. WRONG!

Check that when adding the detailed count, we get the same time complexity

Case 4: dependent + change of variable

```
for(i=1; i<=N; i=i*2){
    res = linear_search(nums1, 3*i, 7); // Θ(i)
    printf("index = %d\n", res);
}
```

1. $TC_{1iter}(i) = \Theta(i)$
2. $TC_{1iter}(i) = \Theta(i)$ is dependent of loop variable $i \Rightarrow \sum_i \Theta(i) = \sum_i i$
3. Change of variable: needed. (Values of i are NOT consecutive.)

Values of i : 1, 2, 4, 8, 16, ..., i , ... $i_{last} \leq N$

$i = 2^e$;

$2^p = i_{last} \leq N$. Use $2^p = N \Rightarrow p = \log_2 N \Rightarrow \log_2 N$ loop repetitions

4. General solution: $\sum_i \Theta(i) = \sum_i i = \sum_{e=0}^p 2^e = \frac{2^{p+1}-1}{2-1} = 2 * 2^p - 1 = 2 * 2^{\log_2 N} - 1 = 2N - 1 = \Theta(N)$

5. $\Theta(N)$ ****This answer cannot have i , T , e , or p in it**

Iteration of loop	e	i	$TC_{1instr} = \Theta(i)$	Sample detailed count:
1 st	0	1 ($=2^0$)	1	$7*(3*1) + 12$
2 nd	1	2 ($=2^1$)	2	$7*(3*2) + 12$
3 rd	2	4 ($=2^2$)	4	$7*(3*4) + 12$
...	
	e	i ($=2^e$)	2^e	$7*(3*2^e) + 12$
...	
	p-1		2^{p-1}	$7*(3*2^{p-1}) + 12$
last	p = $\log_2 N$	$i_{last} (=2^p)$ $2^p = i_{last} \leq N$	2^p	$7*(3*2^p) + 12$

Total: $1+2+4+8+16+\dots+2^e+\dots+2^p = \dots = 2N-1 = \Theta(N)$

Adding all the terms in the TC_{1iter} column gives the time complexity for all the code.

Common error: take this result and multiply it by N or $\log_2 N$ again. WRONG!

Check that when adding the detailed count, we get the same time complexity

Assume that the function `linear_search(int num1[], int T, int val)` has time complexity $\Theta(T)$, with detailed count: $7 * T + 9$. When called with $3*i$ for T it becomes $7*(3*i)+9 = \Theta(i)$

Summary

// Assume that the function `linear_search(int num1[], int T, int val)` has time complexity $\Theta(T)$

```
for(i=1; i<=N; i=i+1) {
    res = linear_search(nums1, M, 7); //  $\Theta(M)$ 
    printf("index = %d\n", res);
}
```

$TC_{1iter}(i) = \Theta(M)$, independent of $i \Rightarrow$ ok with loop repetitions
 i takes consecutive values, no change of variable needed

General sol: $\sum_i \Theta(M) = \sum_{i=1}^N \Theta(M) = N * \Theta(M) = \Theta(MN)$

Solution for this special case: Loop repetitions: N

$$N * \Theta(M) = \Theta(MN)$$

```
for(i=1; i<=N; i=i*5) {
    res = linear_search(nums1, M, 7); //  $\Theta(M)$ 
    printf("index = %d\n", res);
}
```

$TC_{1iter}(i) = \Theta(M)$, independent of $i \Rightarrow$ ok with loop repetitions
 i does NOT take consecutive values \Rightarrow **need change of variable:**

Values of i : 1, 5, 25, 125, ... , $i=5^e$, ... , $5^p = i_{last} \leq N \Rightarrow p = \log_5 N$

General solution: $\sum_i \Theta(M) = \sum_{e=0}^p \Theta(M) = p * \Theta(M) = \Theta(Mp) = \Theta(M \log_5 N)$

Solution for this special case: Loop repetitions : $p = \log_5 N$

$$p * TC_{1iter} = (\log_5 N) * \Theta(M) = \Theta(M \log_5 N)$$

```
for(i=1; i<=N; i=i+1) {
    res = linear_search(nums1, i, 7); //  $\Theta(i)$ 
    printf("index = %d\n", res);
}
```

$TC_{1iter}(i) = \Theta(i)$, dependent of $i \Rightarrow$ must use summation

i takes consecutive values, no change of variable needed

General sol: $\sum_i \Theta(i) = \sum_{i=1}^N \Theta(i) = \sum_{i=1}^N i = \frac{N(N+1)}{2} = \Theta(N^2)$

```
for(i=1; i<=N; i=i*5) {
    res = linear_search(nums1, i, 7); //  $\Theta(i)$ 
    printf("index = %d\n", res);
}
```

$TC_{1iter}(i) = \Theta(i)$, dependent of $i \Rightarrow$ must use summation

i does NOT take consecutive values \Rightarrow **need change of variable:**

Values of i : 1, 5, 25, 125, ... , $i=5^e$, ... , $5^p = i_{last} \leq N \Rightarrow p = \log_5 N$

General solution : $\sum_i \Theta(i) = \sum_i i = \sum_{e=0}^p 5^e = \frac{5^{p+1}-1}{5-1} =$

$$\frac{5*5^p-1}{4} = \Theta(5^p) = \Theta(5^{\log_5 N}) = \Theta(N)$$

Good example: change of variable with i^2 in TC_{1iter}

```
// assume that the time complexity of foo(int t, int M) is  $\Theta(t^2M)$   
for(i=0; i<N; i=i+4)  
    foo(i, M);
```

In the call to foo, i was passed for t =>
use $\Theta(i^2M)$, NOT $\Theta(t^2M)$

for-i: $TC_{1iter}(i) = \Theta(i^2M)$ Dependent ()

change of var. i: 0, 4, 8, 12, ... $i=4e$, $i_{last} = 4p = N \Rightarrow p = N/4$

$$\sum_i \Theta(i^2M) = \sum_{e=0}^{N/4} [(4e)^2M] = 16M \sum_{e=0}^{N/4} e^2 = 16M \frac{\frac{N}{4}(\frac{N}{4}+1)(2\frac{N}{4}+1)}{6} = \Theta(N^3M)$$

$\Theta(N^3M)$

replace i^2 with $(4e)^2$

$$\sum_{k=1}^N k^2 = \frac{N(N+1)(2N+1)}{6}$$

Time complexity of nested loops – special cases solution

- Solve the innermost loop and use its time complexity in the calculation of the $TC_{1iter}()$ of the next outer loop and so on. Note that we solve t first, then k, then i.

```
for(i=1; i<=N; i++)  
    for(k=1; k<=M; k=k*2)  
        for(t=0; t<=i; t=t+3)  
            printf("C");
```

for-t: $TC_{1iter}(t) = \Theta(1)$ independent
t: 0,3,6,9,... $t = 3e, 3p=i \Rightarrow p = i/3$ repetitions

$$i/3 * \Theta(1) = \Theta(i)$$

for-k: $TC_{1iter}(k) = \Theta(i)$ independent
k: 1,2,4,8,... $k = 2^e, 2^p=M \Rightarrow p = \log_2 M$ repetitions

$$\log_2 M * \Theta(i) = \Theta(i * \log_2 M)$$

for-i: $TC_{1iter}(i) = \Theta(i * \log_2 M)$ Dependent
change of variable NOT needed i: 1 to N

$$\sum_{i=1}^N \Theta(i \log_2 M) = \log_2 M \sum_{i=1}^N i = \log_2 M \frac{N(N+1)}{2} = \Theta(N^2 \log_2 M)$$

$$\Theta(N^2 \log_2 M)$$

If the constant for the dominant term was needed, we would keep the constants for all the dominant terms that have a constant: $i/3$ and $N^2/2 \Rightarrow$ the constant for all the code: $1/6$ (this excludes the constant coming from the detailed count)

Time complexity of nested loops – keep multiplication constant for dominant term

- Solve the innermost loop and use its time complexity in the calculation of the $TC_{1iter}()$ of the next outer loop and so on. Note that we solve t first, then k, then i.

```
for(i=1; i<=N; i++)  
    for(k=1; k<=M; k=k*2)  
        for(t=0; t<=i; t=t+5)  
            printf("C");
```

for-t: $TC_{1iter}(t) = 3 \text{ instructions}$ (from: $t \leq i$, $\text{printf}("C"); t=t+5$)
independent
t: 0,5,10,15,... $t = 5e$, $5p=i \Rightarrow p = i/5$ repetitions

$$i/5 * 3 = \Theta(3i/5)$$

for-k: $TC_{1iter}(k) = \Theta(3i/5)$ independent
k: 1,2,4,8,... $k = 2^e$, $2^p=M \Rightarrow p = \log_2 M$ repetitions

$$\log_2 M * \Theta(3i/5) = \Theta((3/5) * i * \log_2 M)$$

for-i: $TC_{1iter}(i) = \Theta((3/5) * i * \log_2 M)$ Dependent
change of variable NOT needed i: 1 to N

$$\sum_{i=1}^N \Theta((3/5) \log_2 M) = (3/5) \log_2 M \sum_{i=1}^N i = (3/5) \log_2 M \frac{N(N+1)}{2} = \Theta((3/5) (1/2) N^2 \log_2 M)$$

$$\Theta\left(\frac{3}{10}\right) N^2 \log_2 M$$

The multiplication constant for the dominant term is: $\frac{3}{10}$

Time Complexity of Sequential Loops

// Write the Θ (Theta) time complexity

```
for (i=left; i<right; i++)  
    printf("%d, ", A[i]);  
for (i=0; i<p; i++) {  
    for (k=0; k<r; k++)  
        printf("B");  
}
```

$\Theta(M)$
where $M = \text{right-left}+1$

$\Theta(pr)$

$\Theta(M+pr)$

The actual size of the data being processed by the first loop is $\text{right-left}+1$.

In applications (e.g. binary search or merge sort) this quantity often results in a fraction (e.g. half) of the amount of data in the previous iteration.

If multiple variable appear in the time complexity, there may be more than one dominant term.

Example 1 the time complexity of the above code

Example 2: $27n^4m + 6n^3 + 7nm^2 + 100 = \Theta (n^4m + nm^2)$

Insertion Sort Time Complexity

i	Inner loop time complexity:		
	Best : 1	Worst: i	Average: i/2
1	1	1	1
2	1	2	2/2
...	...		
N-2	1	N-2	(N-2)/2
N-1	1	N-1	(N-1)/2
Total	(N-1)	$[N * (N-1)]/2$	$[N * (N-1)]/4$
Order of magnitude	$\Theta(N)$	$\Theta(N^2)$	$\Theta(N^2)$
Data that produces it.	Sorted	Sorted in reverse order	Random data

```
void insertion_sort(int A[],int N){
    int i,k,key;
    for (i=1; i<N; i++)
        key = A[i];
        // insert A[i] in the
        // sorted sequence A[0...i-1]
        k = i-1;
        while (k>=0) and (A[k]>key)
            A[k+1] = A[k];
            k = k-1;
        }
        A[k+1] = key;
    }
```

Insertion sort is adaptive

=> $O(N^2)$

'Total' instructions in worst case:

$$T(N) = (N-1) + (N-2) + \dots + 2 + 1 = \\ = [N * (N-1)]/2 \rightarrow \Theta(N^2)$$

Note that the N^2 came from the summation, NOT because 'there is an N in the inner loop' (NOT because $N * N$).

O will be explained in detail later. It says that the algorithm take at most order of N^2 .

See the Khan Academy for a discussion on the use of $O(N^2)$:

<https://www.khanacademy.org/computing/c/algorithm/a/insertion-sort/a/insertion-sort>

Extra topic

NOT required Spring 2021

Calculate exact number of iterations of a loop as a function of the variable(s) that control the loop

Formula for values of i and exact calculation of number of loop iterations – Example 1

```
for (i=0; i<=N; i=i+3)
    printf("A");
```

i takes values: **0,3,6,9,12,... ≤ N**

We notice that these are **consecutive multiples of 3** so we will explicitly show that by writing i as a function of another variable:

i = 3*e

Where **e takes values: 0,1,2,3,...,p**

Here we use **p** to refer to that last multiple of 3 that is ≤ N.

The loop executes (the condition is true) for all i = 3e where e takes values: 0,1,2,3,...,p => 1+p total values (because of the 0) => **the loop iterates 1+p times**. (A)

Next we will compute the exact formula for p:

Because of how we chose p we have: **3p ≤ N** but p is an integer and largest with this property => **p = $\lfloor \frac{N}{3} \rfloor$** (B)

From (A) and (B) it follows that the loop executes $1 + p = 1 + \lfloor \frac{N}{3} \rfloor$ times =>

The loop executes exactly: $1 + \lfloor \frac{N}{3} \rfloor$ times.

As a verification step you should check that the formula does give the exact number of loop iterations for a few values of N: 0,1,2,3,4,15,17

e	i=3e
0	0
1	3
2	6
3	9
...	...
e	3e
...	...
p	3p (i _{last} = 3p , i _{last} ≤ N 3p ≤ N => p = $\lfloor N/3 \rfloor$)

Practice: how would you solve: `for (i=3; i<=N; i=i+3) printf("A");`

Formula for values of i and exact calculation of number of loop iterations – Example 1

```
for (i=2; i<=N; i=i+3)
    printf("A");
```

i takes values: **2,5,8,11,14**,.... ≤ N

We notice that these are consecutive multiples of 3 with an offset of 2. We will explicitly show that by writing i as a function of another variable:

$i = 2 + (3 * e)$

Where e takes values: 0,1,2,3,....,p

Here we use p to refer to that last value **$2 + 3p$ that is ≤ N.**

The loop executes (the condition is true) for all $i = 2 + 3e$ where e takes values: 0,1,2,3,....,p => 1+p total values (because of the 0) => the loop iterates 1+p times. (A)

Next we will compute the exact value of p: $2 + 3p \leq N \Rightarrow 3p \leq (N - 2) \Rightarrow p \leq (N - 2) / 3$, but p is an integer and largest with this property => **$p = \left\lfloor \frac{N - 2}{3} \right\rfloor$** (B)

FINAL ANSWER from (A) and (B): The loop executes exactly: **$1 + \left\lfloor \frac{N - 2}{3} \right\rfloor$** times.

As a verification step you should check that the formula does give the exact number of loop iterations for a few values of N: **2,3,4,5,6** (Note that you start with the smallest value of N for which the loop iterates at least one time: 2. You do not use 0 or 1 for N in this verification.)

e	i=2+3e
0	2
1	5
2	8
3	11
...	...
e	i = 2+3e
...	...
p	$i_{last} \leq N$ ($i_{last} = 2 + 3p$)

Formula for values of i and exact calculation of number of loop iterations – Example 3

```
for (i=1; i<=N; i=i*5)
    printf("A") ;
```

i takes values: **1,5,25,125**,..., $\leq N$

We notice that these are consecutive multiples powers of 5 so we will explicitly show that by writing i as a function of another variable:

$i = 5^e$

Where e takes values: 0,1,2,3,...,p

Here we use p to refer to that largest value **5^p that is $\leq N$** .

The loop executes (the condition is true) for all $i = 5^e$ where e takes values: 0,1,2,3,...,p \Rightarrow 1+p total values (because of the 0) \Rightarrow **the loop iterates 1+p times**. (A)

Next we will compute the exact formula for p.

Because of how we picked p we have: **$5^p \leq N$** , where p is an integer and largest with this property.

Take \log_5 on both sides \Rightarrow **$p \leq \log_5 N \Rightarrow p = \lfloor \log_5 N \rfloor$** (B)

From (A) and (B) it follows that the loop executes **$1 + p = 1 + \lfloor \log_5 N \rfloor$** times \Rightarrow

ANSWER: The loop executes exactly: **$1 + \lfloor \log_5 N \rfloor$** times.

As a verification step you should check that the formula does give the exact number of loop iterations for a few values of N: **1,5,25,26,29,30**

e	$i=5^e$
0	1
1	5
2	25
3	125
...	...
e	$i=5^e$
...	...
p	$i_{\text{last}} \leq N$ $(i_{\text{last}} = 5^p) \Rightarrow$ $p = \lfloor \log_5 N \rfloor$