


Estruturas de Dados / Programação 2
Complexidade de um Programa de Computador

Márcio Ribeiro
marcio@ic.ufal.br
twitter.com/marciomribeiro

n prime?

1

```
for i = 2 to n - 1
  if i divides n
    n is not prime
```

(n - 2) times

2

```
for i = 2 to sqrt(n)
  if i divides n
    n is not prime
```

(√n - 1) times

1ms for a division...

n	1	2
11	9ms	2ms
101	99ms	9ms
1000003 = $10^6 + 3$	= 10^6ms = 10^3sec = 16.66min	= √($10^6 + 3$) - 1 = 10^3ms = 1sec
10000000019 = $10^{10} + 19$	= 10^{10}ms = 10^7sec = **115 days**	= 10^5ms = 100sec = **1.66min**

1

```
for i = 2 to n - 1
  if i divides n
    n is not prime
```

(n - 2) times

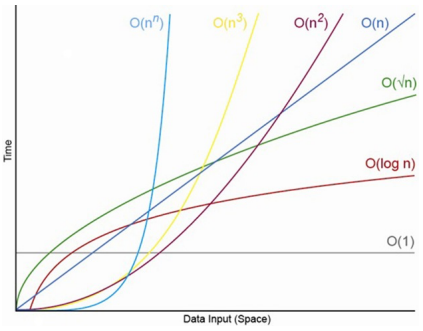
O(n)

2

```
for i = 2 to sqrt(n)
  if i divides n
    n is not prime
```

(√n - 1) times

O(√n)



How to analyze?

Model Machine

- Single processor
- Sequential execution
- 1 unit time for arithmetical and logical operations
- 1 unit time for assignments and returns

Assumptions

- Each statement takes the same unit of time
- We ignore constant multiples
- Statements within a loop will execute as many times as the **maximum** permitted by the loop control

```
int sum_of_list(list, n)
{
  int total = 0;
  for (i = 0; i < n; i++)
    total = total + list[i];
  return total;
}
```

Cost	N. of times
1 (c_1)	1
2 (c_2)	n + 1
2 (c_3)	n
1 (c_4)	1

t(n) = 1 + 2(n + 1) + 2n + 1
t(n) = 4n + 4

t(n) = cn + c'

$c = c_2 + c_3$

$c' = c_1 + c_2 + c_4$

Statements

- Remember: we ignore constant multiples

```
Statement 1
Statement 2
…
Statement n
```

```
Total time = t(s1) + t(s2) + … + t(sn)
```

```
If each statement is "simple", then the time
for each statement is constant, and the total
time is also constant: O(1)
```

If statements

- Time equal to the larger of its two branches
- If sequence 1 is O(n) and sequence 2 is O(1), the worst case time for the whole if-then-else statement would be O(n)

```
if (condition) {
  sequence of statements 1
} else {
  sequence of statements 2
}
```

For loops

- The loop executes "n" times, so the sequence of statements also executes "n" times
- Since we assume the statements are O(1), the total time for the for loop is "n", which is O(n) overall.

```
for (i = 1; i <= n; i++) {
  something that is O(1)
}
```

1 + 1 + 1 + … + 1
(n times)

Nested for loops

- Inner loop iterates "k" times; Simplifying assumption that inner loop iterates "n" times: $O(n^2)$. Overestimation… but this has no effect on the final estimate!

```
for (k = 1; k <= n; k++) {
  for (i = 1; i <= k; i++) {
    something that is O(1)
  }
}
```

1 (first time) +
2 (second time) +
…
+ n (last time)

$$\sum_{i=1}^{n} i = n(n+1)/2$$

which still is $O(n^2)$

What about this one?

```
for (i = 1; i <= n; i++) {
  for (j = 1; j <= i; j++) {
    for (k = 1; k <= i; k++) {
      something that is O(1)
    }
  }
}
```

- For each value of "i", the two inner loops contribute to i^2

$$1^2 + 2^2 + 3^2 + \ldots + n^2 \qquad \sum_{i=1}^{n} i^2 = n(n+1)(2n+1)/6$$

which still is $O(n^3)$

Exercises

1)
```
for (i = 0; i < n; i++) {
  something that is O(1)
}
for (j = 0; j < m; j++) {
  something that is O(1)
}
```

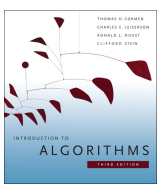
2)
```
for (i = 0; i < n; i++) {
  for (j = 0; j < n; j++) {
    something that is O(1)
  }
}
for (k = 0; k < n; k++) {
  something that is O(1)
}
```

Exercises

3)
```
for (i = 0; i < n; i++) {
  for (j = n; j > i; j--) {
    something that is O(1)
  }
}
```

4)
```
for (i = 0; i < n; i++) {
  min = i;
  for (j = i + 1; j < n; j++)
    if (s[j] < s[min]) min = j;
  swap(&s[i], &s[min]);
}
```

References



Chapters 1, 2, and 3

Chapter 2