

Algorithm performance

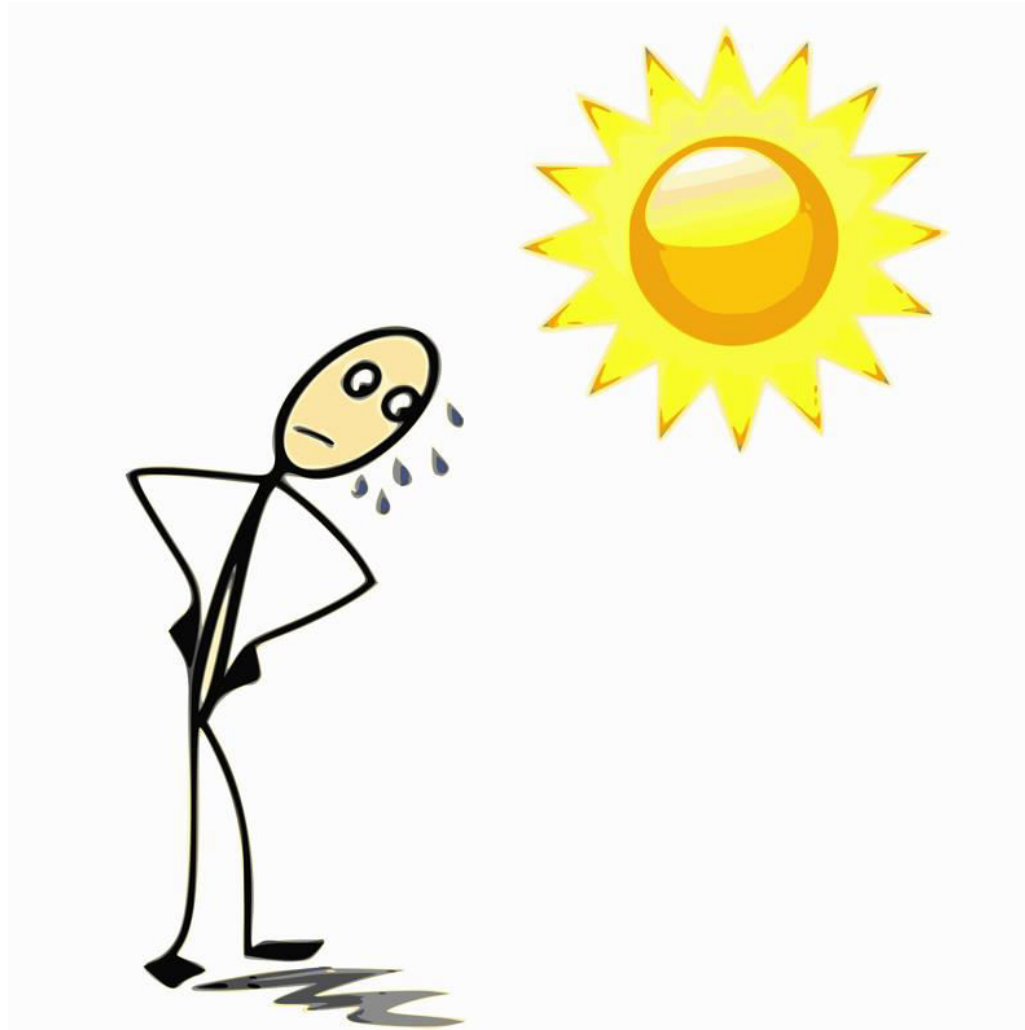


Asymptotic analysis

By the end of this video you will be able to...

- Explain why asymptotic analysis is useful
- Calculate the big-O class of a code snippet





Initialization time



Initialization time

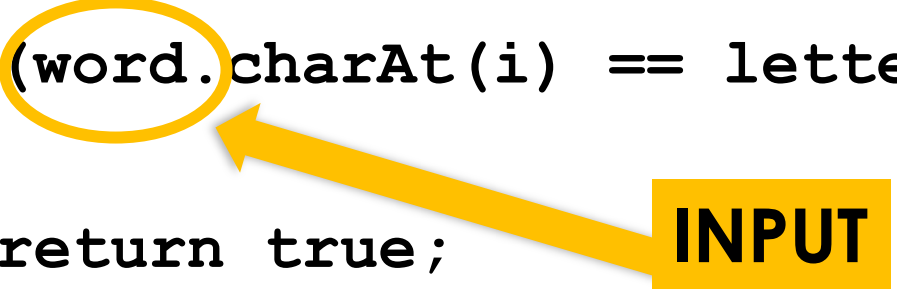
**Implementations of
specific operations**

Focus on how performance scales

If input is **twice** as big,
how many **more operations** do we need?

```
if (word.charAt(i) == letter)
{
    return true;
}
```

```
if (word.charAt(i) == letter)
{
    return true;
}
```

A yellow oval highlights the word parameter in the charAt method call. A yellow arrow points from a yellow box labeled 'INPUT' to the word parameter.

INPUT


```
if (word.charAt(i) == letter)
{
    return true;
}
```

```
if (word.charAt(i) == letter)
{
    return true;
}
```

Constant time

```
int count = 0;
for (int i = 0; i < word.length(); i++)
{
    count ++;
}
```

```
int count = 0;
for (int i = 0; i < word.length(); i++)
{
    count ++;
}
```



INPUT of SIZE n

```
int count = 0; 1
for (int i = 0; i < word.length(); i++)
{
    count ++;
}
```

```
int count = 0;
```

```
for (int i = 0; i < word.length(); i++)
```

```
{
```

```
    count ++;
```

```
}
```

1

1

1

n times

```
int count = 0;
```

```
for (int i = 0; i < word.length(); i++)
```

```
{
```

```
    count ++;
```

```
}
```

1

1

n times

1

$1 + (1 + 3n + 1)$

count

i

n

```
int count = 0;
```

```
for (int i = 0; i < word.length(); i++)
```

```
{
```

```
    count ++;
```

```
}
```

1

1

n times

1

$3n + 3$


```
int count = 0;
for (int i = 0; i < word.length(); i++)
{
    count ++;
}
```

Linear time

$$f(n) = O(g(n))$$

means

Eventually,
 **$f(n)$ and $g(n)$ grow in same way
as their input grows**
up to constants