AP Computer Science A: *Informal Code Analysis*

CodeHS

This Lesson Will Cover:

- Informal run time comparisons of iterative statements
- Statement execution count

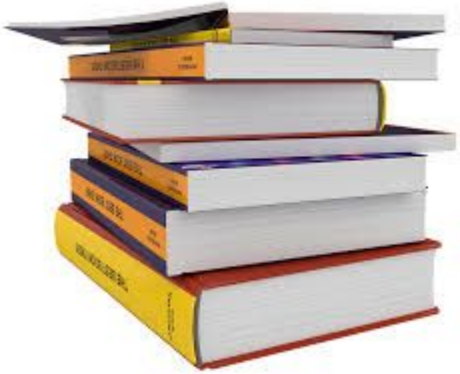
CodeHS

Algorithm

Algorithms are a step-by-step process that solves a problem.

Everyday Algorithms

We use algorithms everyday!



```
makePBJ()
{
     takeBread();
     spreadJelly();
     spreadPeanutButter();
     if(likeToasted)
     {
          toast();
     }
}
```

```
readBook()
{
     openBook();
     while(pagesUnread)
     {
          readpage();
          takeNotes();
          flipPage();
     }
}
```

What Makes a Good Algorithm?

So what makes a good algorithm?

What Makes a Good Algorithm?

So what makes a good algorithm?

- **Correctness**
- **Efficiency** (run time and memory requirements)
- **Easy to understand** by someone else

Correctness

Algorithms should work as intended:

```
/*This program will print the
loop counter 100 times*/

for(int i=1; i <= 10; i++)
{
   System.out.println(i);
}
```

Does this code snippet work as intended??

Correctness

Algorithms should work as intended:

```
/*This program will print the
loop counter 100 times*/

for(int i=1; i <= 10; i++)
{
   System.out.println(i);
}
```

If a program doesn't solve the problem it was intended to solve, then it's not a good algorithm!

Correctness

Algorithms should work as intended:

```
/*This program will print the
loop counter 100 times*/

for(int i=1; i <= 100; i++)
{
   System.out.println(i);
}
```

Efficiency

Algorithms need to be efficient.

Efficiency is important because it affects program **speed**, and program **cost.**

Efficiency cost

Running a computer program uses **CPU Processing Time**.

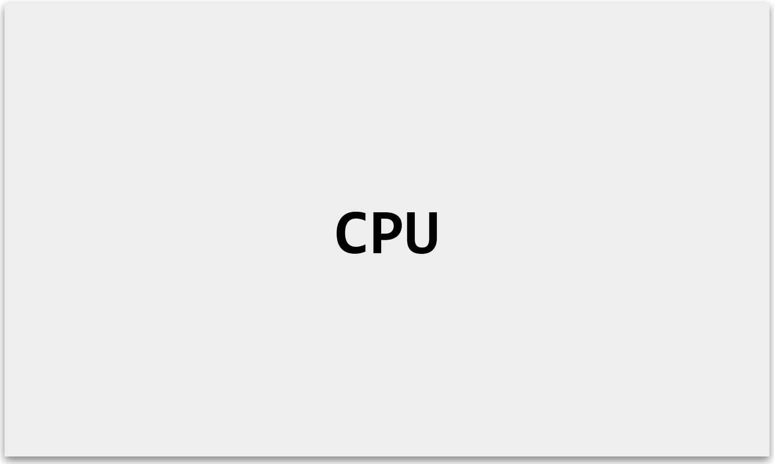
Processing Time is the amount of the time the computer spends processing program instructions.

Efficiency cost

A computer has a limit to its **CPU usage** at any given time.

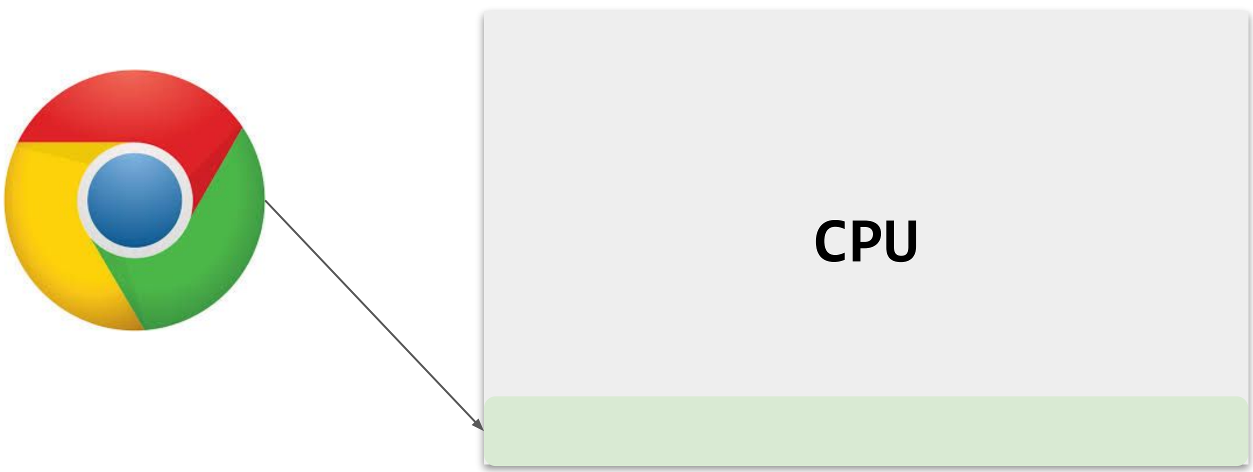
Efficiency cost

A computer has a limit to its **CPU usage** at any given time.



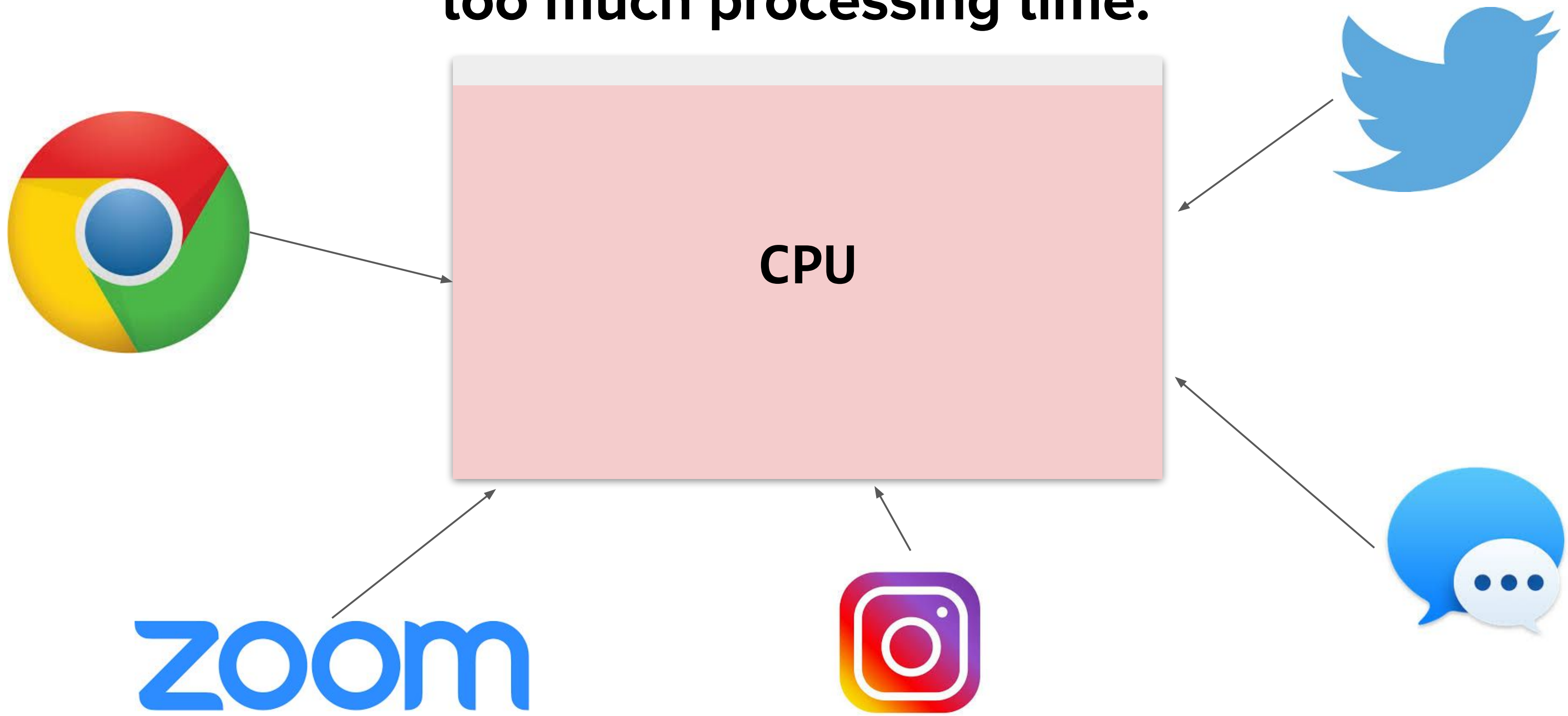
Efficiency cost

When applications and programs need to use processing time, it takes a percentage of the CPU usage.



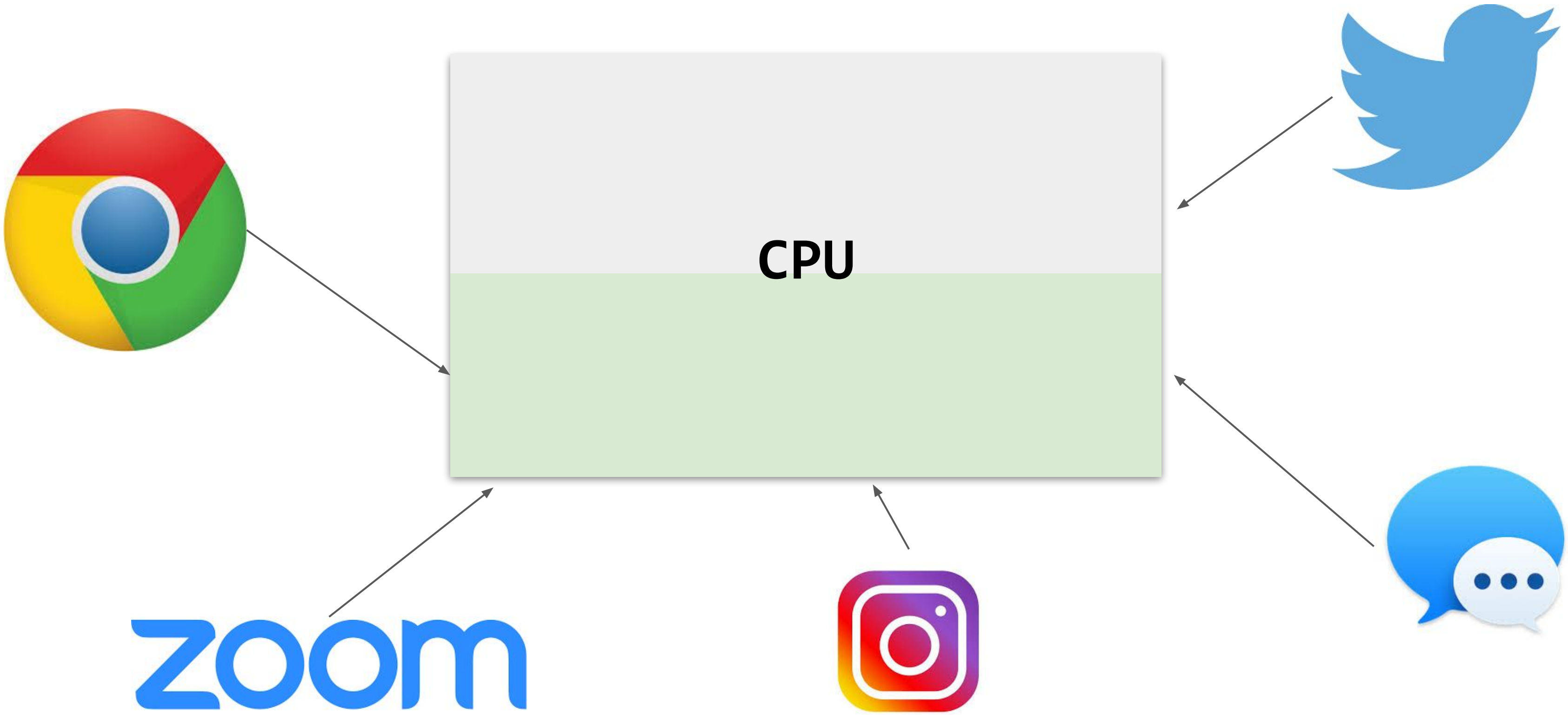
Efficiency cost

Computers experience a slowdown when CPU usage reaches its upper limits, which occurs when applications use too much processing time:

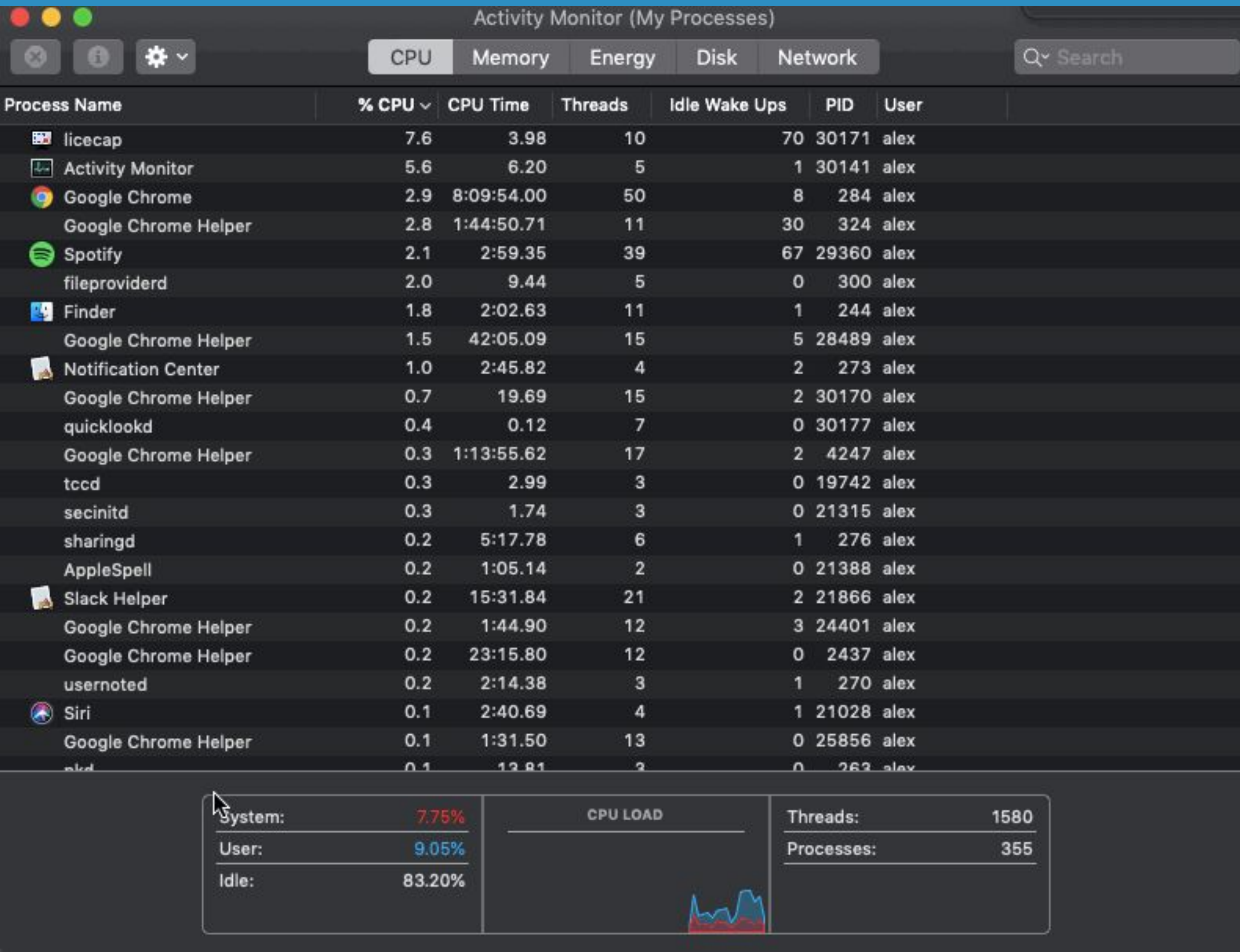


Efficiency cost

More efficient programs use less Processing Time, which can decrease overall CPU usage:



Monitoring CPU Usage



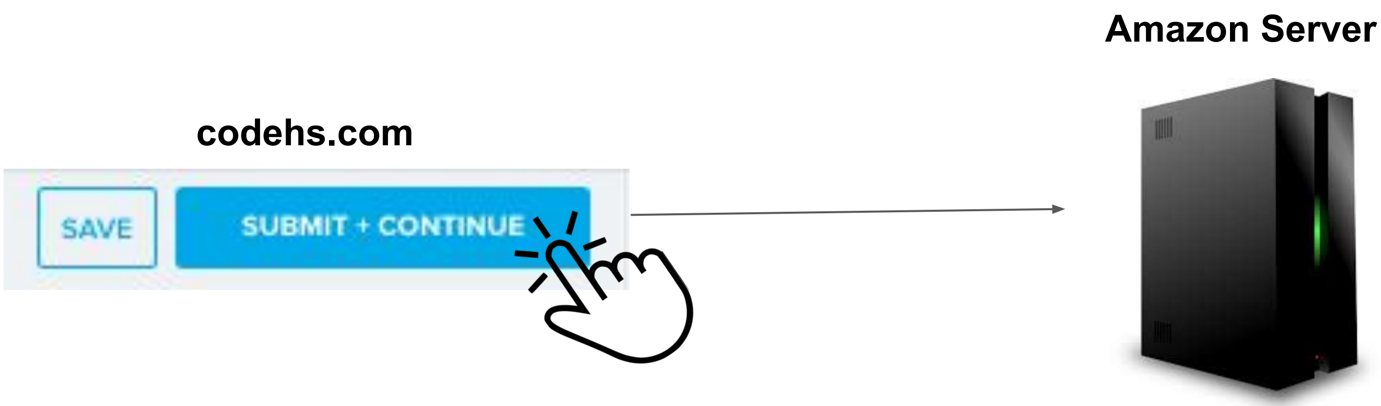
We can actually monitor how our computers are using CPU processing time and usage!

Efficiency cost

Program **speed** also affects program **cost**!

Efficiency cost

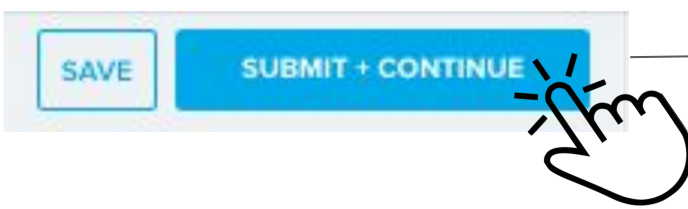
Program **speed** also affects program **cost**!



Efficiency cost

Program **speed** also affects program **cost**!

codehs.com



Amazon Server

Server runs program to save your code, and sends code to your teacher!

Efficiency cost

Program **speed** also affects program **cost**!

Servers cost money to run. The more CPU usage, the more some servers cost

Amazon Server

Efficiency cost

Program **speed** also affects program **cost**!

Amazon Server

Limiting processing time reduces CPU usage, and can save money on server costs!

Efficiency

An efficient algorithm is one that makes economical use of processing time and computer memory.

Efficiency

An efficient algorithm is one that makes economical use of processing time and computer memory.

The major questions that we want to ask about an algorithm are:

How long does it take to perform this task?

Is there another approach that will get the answer more quickly?

Which Algorithm is Most Efficient?

Which of these algorithms is most efficient?

#1:

You say your birthday and ask if anyone in the room has the same birthday as you.

If anyone has the same birthday, they say “yes.”

Which Algorithm is Most Efficient?

Which of these algorithms is most efficient?

#2:

You tell Anika your birthday and ask if she has the same birthday.

If she says “no,” you tell Matt your birthday and ask if him if he has the same birthday.

You repeat this process for each person in the room.

Which Algorithm is Most Efficient?

Which of these algorithms is most efficient?

#3:

You only ask questions of Anika, who only asks questions of Matt, who only asks questions of Isabelle, and so on.

You tell Anika your birthday, and ask if she has the same birthday; if she says "no," you ask her to find out if Matt has the same birthday.

Anika asks Matt and tells you his answer. If his answer is "no," you ask Anika to find out about Isabelle. Anika asks Matt to find out about Isabelle, etc.

Which Algorithm is Most Efficient?

#1 is the most efficient:
You say your birthday and ask if anyone in the room has the same birthday as you. If anyone has the same birthday, they say "yes."

This is the most efficient algorithm because you only need to ask the question once! It will take the least amount of TIME.

Measuring Algorithm Performance

It is important to be able to measure, or at least make an educated guess, about the space and time needed to execute an algorithm.

This will allow you to compare alternative approaches to a problem you need to solve.

Measuring Algorithm Performance

The absolute running time of an algorithm can’t be predicted since this depends on the:

- Programming language
- Computer
- Other programs running at the same time
- The quality of the operating system
- and many other factors.

Statement Execution Count

We need a machine-independent way to estimate an algorithm’s running time.

CodeHS

Statement Execution Count

The **Statement Execution Count** is the number of times a statement is executed by the program.

Statement Execution Count

A executed statement might refer to:

- an arithmetic operation (`+ , *`)
- an assignment (`int value = 2`)
- a test (`x == 0`)
- an input/output

Example Algorithm

The **Statement Execution Count** is the number of times a statement is executed by the program.

```
public void computeSum ()
{
    int sum = 0;
    int n = 10;

    for (int i = 0; i < n; i++)
    {
        sum += i;
    }

    System.out.println(sum);
}
```

Example Algorithm

The **Statement Execution Count** is the number of times a statement is executed by the program.

Execution Count

1

```
public void computeSum ()
{
    int sum = 0;
    int n = 10;

    for (int i = 0; i < n; i++)
    {
        sum += i;
    }

    System.out.println(sum);
}
```

First instruction executes once!

Example Algorithm

The **Statement Execution Count** is the number of times a statement is executed by the program.

Execution Count

2

```
public void computeSum ()
{
    int sum = 0;
    int n = 10;

    for (int i = 0; i < n; i++)
    {
        sum += i;
    }

    System.out.println(sum);
}
```

First instruction executes once!

Second instruction executes once!

Example Algorithm

The **Statement Execution Count** is the number of times a statement is executed by the program.

Execution Count

2 + n

```
public void computeSum ()
{
    int sum = 0;
    int n = 10;

    for (int i = 0; i < n; i++)
    {
        sum += i;
    }

    System.out.println(sum);
}
```

First instruction executes once!

Second instruction executes once!

Third instruction executes n times!

Example Algorithm

The **Statement Execution Count** is the number of times a statement is executed by the program.

Execution Count

$\boxed{2 + n + 1}$

```
public void computeSum ()
{
    int sum = 0;
    int n = 10;

    for (int i = 0; i < n; i++)
    {
        sum += i;
    }

    System.out.println(sum);
}
```

First instruction executes once!

Second instruction executes once!

Third instruction executes n times!

Fourth instruction executes once!

Example Algorithm

The **Statement Execution Count** is the number of times a statement is executed by the program.

```
public void computeSum ()
{
    int sum = 0;
    int n = 10;

    for (int i = 0; i < n; i++)
    {
        sum += i;
    }

    System.out.println(sum);
}
```

Execution Count

`3 + n`

The algorithm executed **3 + n** instructions where n is the value initialized in the second execution statement.

Control Flow Statements

While tracking individual statements is relatively straightforward, control flow statements can make this process more difficult.

CodeHS

If you have...`if-then-else` Statements

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

OR

For conditional statements, either sequence 1 will execute, or sequence 2 will execute. So the count will include one of the 2 statement counts.

If you have...for Loops

```
for (int i = initial; i < n; i++)
{
    sequence of statements
}
```

If the increment of a for loop is ++, the execution count is (n - initial)

If you have...for Loops

```
for (int i = initial; i < n; i++)
{
    sequence of statements
}
```

If the increment of a for loop is ++, the execution count is (n - initial)

initial	n	Execution count
0	10	10 - 0 = 10
4	8	8 - 4 = 4

If you have...Nested Loops

```
for (int i = initial; i < n; i++)
{
    for (j = init2; j < m; j++)
    {
        sequence of statements
    }
}
```

For nested loops, the execution count is (n - initial) * (m - init2)

If you have...Nested Loops

```
for (int i = initial; i < n; i++)
{
    for (j = init2; j < m; j++)
    {
        sequence of statements
    }
}
```

For nested loops, the execution count is (n - initial) * (m - init2)

initial	n	init2	m	Execution count
0	10	0	5	(10 - 0) * (5 - 0) = 50
4	8	1	4	(8 - 4) * (4 - 1) = 12

Comparing Algorithms

Comparing statement execution counts is one way to informally determine which algorithm is **more efficient**.

Algorithms with **lower** execution counts are likely to be more efficient.

Practice Problem One

What is the execution count for this algorithm?

```
for (int n = 50; n > 0; n = n / 2)
{
   System.out.print(n);
}
```

Practice Problem One

What is the execution count for this algorithm?

```
for (int n = 50; n > 0; n = n / 2)
{
   System.out.print(n);
}
```

Execution Count

6

50	25	12	6	3	1

Practice Problem Two

What is the execution count for this algorithm?

```
for (int i = 0; i < 10; i++)
{
   if (i == 2)
   {
      i++;
   }

}
```

Practice Problem Two

What is the execution count for this algorithm?

```
for (int i = 0; i < 10; i++)
{
    if (i == 2)
    {
        i++;
    }

}
```

Increases i by 1, so the next increment is 4!

0	1	2	4	5	6	7	8	9

Execution Count

9

Practice Problem Three

Which algorithm is more efficient?

```
public void findChar(String string, char key)
{
     for(int i = 0; i < string.length(); i ++)
     {
          char character = string.charAt(i);
          if (character == key)
          {
               System.out.println("Found!");
               break;
          }
     }
}
```

```
public void findChar(String string, char key)
{
     for(int i = 0; i < string.length(); i ++)
     {
          char character = string.charAt(i);
          if (character == key)
          {
               System.out.println("Found!");
          }
     }
}
```

Practice Problem Three

Which algorithm is more efficient?

```
public void findChar(String string, char key)
{
	for(int i = 0; i < string.length(); i ++)
	{
		char character = string.charAt(i);
		if (character == key)
		{
			System.out.println("Found!");
			break;
		}
	}
}
```

```
public void findChar(String string, char key)
{
	for(int i = 0; i < string.length(); i ++)
	{
		char character = string.charAt(i);
		if (character == key)
		{
			System.out.println("Found!");
		}
	}
}
```

Practice Problem Three

findChar(“CodeHS”, ‘e’)

```
public void findChar(String string, char key)
{
     for(int i = 0; i < string.length(); i ++)
     {
          char character = string.charAt(i);
          if (character == key)
          {
               System.out.println(“Found!”);
               break;
          }
     }
}
```

Stops the for loop after ‘e’ is found!

Execution Count

4

```
public void findChar(String string, char key)
{
     for(int i = 0; i < string.length(); i ++)
     {
          char character = string.charAt(i);
          if (character == key)
          {
               System.out.println(“Found!”);
          }
     }
}
```

Continues running even after finding ‘e’

Execution Count

6

Now It's Your Turn!

CodeHS

Concepts Learned this Lesson

Term	Definition
Algorithm	Step-by-step process that solves a problem.
Statement execution count	The number of times a statement is executed by the program.
Big-O Notation	A way to represent how long an algorithm will take to execute. It helps to determine how efficient different approaches to solving a problem are.

Standards Covered

- (LO) CON-2.H Compute statement execution counts and informal run-time comparison of iterative statements.
- (EK) CON-2.H.1 A statement execution count indicates the number of times a statement is executed by the program.