

Algorithms

Arthur Hoskey, Ph.D.
Farmingdale State College
Computer Systems Department

- Go over important dates

Important Dates

- Overview of Big O Notation

Today's Lecture

- **Computational Complexity** – Area of Computer Science that focuses on the amount of resources required to do something.

Computational Complexity

Amount of resources required
(time/space are resources)

Sub
Areas

Computational Complexity Theory

Focused on the **complexity of problems (sorting, searching etc...)**. Classifies problems according to their resource usage.

Analysis of Algorithms

Focused on the **complexity of an explicit algorithm**. Given a specific algorithm what are its resource requirements.



We will do a small amount of
work here today

Computational Complexity

- Some algorithms will perform faster than others.
- It is important to be able to compare different algorithms so we can choose the best one for the problem at hand.
- How can we compare algorithms???

Comparison of Algorithms

- Can we use the **actual time** (say in milliseconds) it takes for a program to run to compare algorithms?

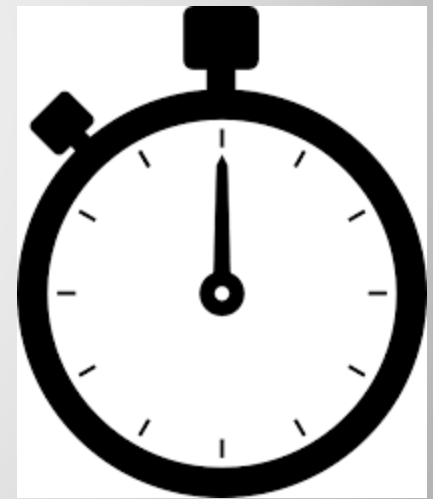


Comparison of Algorithms

- Can we use the **actual time** (say in milliseconds) it takes for a program to run to compare algorithms?

NO. Not a good comparison

1. Computer speeds can vary.
2. If algorithms are run on the same computer the load that the OS is dealing with at any instant in time can vary.



Comparison of Algorithms

- We need a higher-level approach to compare algorithms.

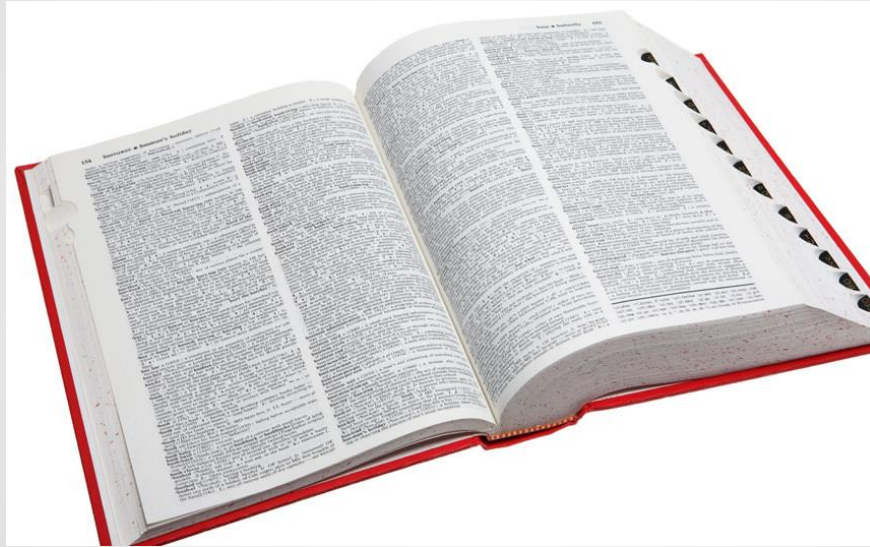
Better Algorithm Comparison

1. Isolate a particular operation that is fundamental to the algorithm.
2. Then count the number of times that this operation is performed.

This eliminates variables such as computer speeds and OS load.

Higher-Level Approach

- Now we will go over an example of organizing a dictionary in two different ways...



Dictionary Organization Example

- A dictionary associates words with definitions.
- Word → Definition.
- For example...

Word	Definition
Student	A person formally engaged in learning, especially one enrolled in a school or college.
Computer	A programmable electronic device designed to accept data, perform prescribed mathematical and logical operations at high speed, and display the results of these operations. Mainframes, desktop and laptop computers, tablets, and smartphones are some of the different types of computers.

Dictionary Organization Example

Pretend two people are creating a dictionary and each uses their own organization scheme to store the entries.

Method 1 – Put the words in **alphabetical order**.

Method 2 – Put the words in **random order**.

What metric should we use to compare the two algorithms?

Dictionary Organization Example

Algorithm Comparison

- One operation we could count is the **number of comparisons** necessary to find an element.
- How many entries would we have to "look at" in the dictionary before we found the one that we wanted?

Dictionary Organization Example

SMALL GROUP EXERCISE

Pretend two people are writing a dictionary and each uses their own organization scheme to store the entries.

Method 1 – Put the words in **alphabetical order**.

Method 2 – Put the words in **random order**.

In the **worst case**, which method is faster when **finding** a word's definition using **comparison** as the metric? Why?

Discuss what is happening for each scheme.

Dictionary Organization Example

Method 1 – Put the words in alphabetical order.

Method 2 – Put the words in random order.

Method 1 is faster (find operation).

Another Question

If there were **16** entries in the dictionary and the target entry is the **last one** then how many comparisons are needed in the **worst** case for each scheme?

Dictionary Organization Example

Method 1 – Put the words in alphabetical order.

Method 2 – Put the words in random order.

Method 1 is faster (find operation).

Another Question

If there were **16** entries in the dictionary and the target entry is the last one then how many comparisons are needed in the **worst** case for each scheme?

ANSWER

Method 1 – Less than 16

Method 2 - 16 Comparisons are needed

Dictionary Organization Example

Method 1 – Put the words in alphabetical order.

Method 2 – Put the words in random order.

- We just compared the algorithms to see which one does a **Find()** faster.

What other operation could we compare?

Dictionary Organization Example

Method 1 – Put the words in alphabetical order.

Method 2 – Put the words in random order.

- Add New Entry. Now compare the speed it takes to add a new entry. This is an **Insert() operation**.

Which algorithm is faster when inserting?

Dictionary Organization Example

Method 1 – Put the words in alphabetical order.

Method 2 – Put the words in random order.

Which algorithm is faster when inserting?

ANSWER:

- **Method 2** is faster in this case (insert operation). All you have to do is put it at the end.

Another Question

- *What “**extra work**” do you have to do to insert data using the Method 1 algorithm?*

Dictionary Organization Example

Method 1 – Put the words in alphabetical order.

Method 2 – Put the words in random order.

- Method 1 requires that you first **FIND** the insertion place then you can insert the new entry. You cannot just put the entry at the end.

Dictionary Organization Example

Method 1 – Put the words in alphabetical order.

Method 2 – Put the words in random order.

- *If there were **16** entries how many comparisons would it take to find the correct insertion place using the Method 2 algorithm?*

Dictionary Organization Example

Method 1 – Put the words in alphabetical order.

Method 2 – Put the words in random order.

- *If there were **16** entries how many comparisons would it take to find the correct insertion place using the Method 2 algorithm?*

ANSWER

- *0 comparisons. Just put it at the end.*

Dictionary Organization Example

Performance on Find and Insert (Method 2)

Method 2 – Put the words in random order.

Assuming **16** entries and the target is at the end for a **Find()**:

Algorithm	Runtime
Find()	16 comparisons
Insert()	0 comparisons

Performance on Find and Insert (Method 2)

Method 2 – Put the words in random order.

Assuming **5000** entries and the target is at the end for a **Find()**:

Algorithm	Runtime
Find()	5000 comparisons
Insert()	0 comparisons

Note

Insert() remains **constant** no matter how many entries there are in the dictionary using the **random order** organization scheme.

Performance on Find and Insert (Method 2)

Method 2 – Put the words in random order.

Assuming **5000** entries and the target is at the **middle** point for a **Find()**:

Algorithm	Runtime
Find()	??? comparisons
Insert()	0 comparisons

Performance on Find and Insert (Method 2)

Method 2 – Put the words in random order.

Assuming **5000** entries and the target is at the **middle** point for a **Find()**:

Algorithm	Runtime
Find()	2500 comparisons
Insert()	0 comparisons

Performance on Find and Insert (Method 2)

Method 2 – Put the words in random order.

Assuming **5000** entries and the target is at the **beginning of the dictionary** while performing a Find():

Algorithm	Runtime
Find()	??? comparisons
Insert()	0 comparisons

Performance on Find and Insert (Method 2)

Method 2 – Put the words in random order.

Assuming **5000** entries and the target is at the **beginning of the dictionary** while performing a Find():

Algorithm	Runtime
Find()	1 comparison
Insert()	0 comparisons

Performance on Find and Insert (Method 2)

Method 2 – Put the words in random order.

Assume **5000** entries and you do not know where the target is. What is the **average** number of comparisons:

Algorithm	Runtime
Find()	??? comparisons
Insert()	0 comparisons

Performance on Find and Insert (Method 2)

Method 2 – Put the words in random order.

Assume **5000** entries and you do not know where the target is. What is the **average** number of comparisons:

Algorithm	Runtime
Find()	2500 comparisons
Insert()	0 comparisons

Performance on Find and Insert (Method 2)

Method 2 – Put the words in random order.

Assume you have **n** entries. Also, assume the **target is at the end**. Now how many comparisons are necessary?

Algorithm	Runtime
Find()	??? comparisons
Insert()	??? comparisons

Performance on Find and Insert (Method 2)

Method 2 – Put the words in random order.

Assume you have **n** entries. Also, assume the **target is at the end**. Now how many comparisons are necessary?

Algorithm	Runtime
Find()	n comparisons
Insert()	0 comparisons

Performance on Find and Insert (Method 2)

Method 2 – Put the words in random order.

Assume you have **n** entries and you don't know where the target is. What is the **average** runtime for each?

Algorithm	Runtime
Find()	??? comparisons
Insert()	??? comparisons

Performance on Find and Insert (Method 2)

Method 2 – Put the words in random order.

Assume you have **n** entries and you don't know where the target is. What is the **average** runtime for each?

Algorithm	Runtime
Find()	<u>n/2</u> comparisons
Insert()	0 comparisons

On average you will have to look through **half** the entries.

Performance on Find and Insert (Method 2)

Algorithm Comparison uses something called **Big O notation**.

Assume you have **n** entries or data items.

A runtime of **O(n)** means you have to "look at" each item in the collection once. n comparisons are required.

So if there are 100 entries then n is 100. If there are 5000 entries then n is 5000.

The number of comparisons will vary depending on how many items there are in the collection.

Algorithm Comparison - Big O

Assume you have n entries or data items.

$O(1)$ means you have to "look at" one entry (kind of). It really means the **amount of work does not change**.

This is called "**constant**" time. The number of things to do remains constant no matter how many items there are.

IMPORTANT

A $O(1)$ running time **will not change** no matter what the value of n is.

Algorithm Comparison - Big O

Big O notation is used as an ***approximation or estimate*** of how long an algorithm will take to run.

It is not meant to be an exact number.

It is a worst-case estimation.

Algorithm Comparison - Big O

What if we went to buy a car. While buying the car we also bought a sandwich and a soda while we were waiting. Assume the following costs of the items that we just bought.

Item	Cost
Car	\$25000
Sandwich	\$10
Soda	\$1

Approximately how much did we spend?



Calculating Big O

What if we went to buy a car. While buying the car we also bought a sandwich and a soda while we were waiting. Assume the following costs of the items that we just bought.

Item	Cost
Car	\$25000
Sandwich	\$10
Soda	\$1

Approximately how much did we spend?

Answer


Approximately \$25000

Calculating Big O

- When calculating the Big O complexity class for a function you can "ignore" the small terms.
- You only need to look at the "big" terms.
- Look for the terms that dominate the calculation.
- For example...

Calculating Big O Complexity Class

Find the Big O complexity class for $f(n)$.

$$f(n) = n + 1 + 1$$


Terms

**Look for the
“largest”
term**

Calculating Big O Complexity Class

Find the Big O complexity class for $f(n)$.

$$f(n) = n + 1 + 1$$



Terms

**Look for
the largest
term**

**The n term dominates the rest of the
equation so you can ignore the 1
terms**

Answer

$$f(n) \in O(n)$$



**$O(n)$ is the set of all functions where
the n term dominates. The particular
 $f(n)$ we show above is a member of
this set of functions.**

Calculating Big O Complexity Class

Find the Big O complexity class for $f(n)$.

$$f(n) = n/2 + 1 + 1$$

Calculating Big O Complexity Class

Find the Big O complexity class for $f(n)$.

$$f(n) = n/2 + 1 + 1$$

Answer

$$f(n) \in O(n)$$

Note

You can ignore constants in equations. The term $n/2$ is $n * 1/2$. The $1/2$ can be ignored.

Calculating Big O Complexity Class

- What are the run times for the following operations on an array?

Finding max?

Calculating total of all elements?

Calculating Big O

How do some common Big O complexity classes vary as n increases?

n	$O(n)$	$O(n^2)$	$O(1)$	$O(\log_2 n)$
1				
2				
4				
8				
16				
32				
64				
128				
256				
512				
1024				

???

Common Big O Complexity Classes

How do some common Big O complexity classes vary as n increases?

n	O(n)	O(n ²)	O(1)	O(log ₂ n)
1	1	1	1	0
2	2	4	1	1
4	4	16	1	2
8	8	64	1	3
16	16	256	1	4
32	32	1024	1	5
64	64	4096	1	6
128	128	16384	1	7
256	256	65536	1	8
512	512	262144	1	9
1024	1024	1048576	1	10

Common Big O Complexity Classes

Problem

Order the columns from "fastest" to "slowest".

n	$O(n)$	$O(n^2)$	$O(1)$	$O(\log_2 n)$
1	1	1	1	0
2	2	4	1	1
4	4	16	1	2
8	8	64	1	3
16	16	256	1	4
32	32	1024	1	5
64	64	4096	1	6
128	128	16384	1	7
256	256	65536	1	8
512	512	262144	1	9
1024	1024	1048576	1	10

Common Big O Complexity Classes

Answer (fastest to slowest)
 $O(1)$, $O(\log n)$, $O(n)$, $O(n^2)$

n	$O(1)$	$O(\log_2 n)$	$O(n)$	$O(n^2)$
1	1	0	1	1
2	1	1	2	4
4	1	2	4	16
8	1	3	8	64
16	1	4	16	256
32	1	5	32	1024
64	1	6	64	4096
128	1	7	128	16384
256	1	8	256	65536
512	1	9	512	262144
1024	1	10	1024	1048576

Common Big O Complexity Classes

What is the Big O complexity class for the following function?

$$f(n) = 20n + n^2 + 1$$

Which term dominates (grows the most)?


Which terms could represent the car, sandwich, and soda?



Calculating Big O

What is the Big O complexity class for the following function?

$$f(n) = 20n + n^2 + 1$$



Sandwich Car Soda

Answer

$$f(n) \in O(n^2)$$

Note

**n^2 is the largest term.
Other terms can be
ignored.**

Calculating Big O

SMALL GROUP EXERCISE

What are the Big O complexity classes of the following functions:

- $f(n) = 3 \cdot \log n + 100000 \cdot 3$
- $f(n) = 6 \cdot n + n \cdot n + \log n + 2000$
- $f(n) = n \cdot \log n + n + 88$

Practice – Find Complexity Class

- Here is another way of thinking about big O...

Big O Complexity Classes

Put the cars into their appropriate category.

- Race car
- SUV
- Economy car

?



?



?



?



Big O Complexity Classes

Put the cars into their appropriate category.

- Race car
- SUV
- Economy car

Economy



Race



SUV



Economy



Big O Complexity Classes

- In the previous example we placed cars into different categories.
- There are many cars within each category.
- The cars within each category may have slight differences but they are very similar and have a lot in common.
- With big O, we are also trying to put things into categories.
- Big O analysis is trying to determine which category a particular function belongs to (as opposed to a car in the previous example).

Big O Complexity Classes

Put the functions into their appropriate big O categories.

- $O(1)$
- $O(n)$
- $O(n^2)$

?

$$f(n)=3n^2+n$$

?

$$f(n)=200n+1$$

?

$$f(n)=2n+1$$

?

$$f(n)=10$$

?

$$f(n)=8n+4$$

?

$$f(n)=4n^2+2n$$

Big O Complexity Classes

Put the functions into their appropriate big O categories.

- $O(1)$
- $O(n)$
- $O(n^2)$

$$O(n^2)$$
$$f(n)=3n^2+n$$

$$O(n)$$
$$f(n)=200n+1$$

$$O(n)$$
$$f(n)=2n+1$$

$$O(1)$$
$$f(n)=10$$

$$O(n)$$
$$f(n)=8n+4$$

$$O(n^2)$$
$$f(n)=4n^2+2n$$

Big O Complexity Classes

- There are many functions within each big O category.
- The functions within each category may have slight differences but they are very similar and have a lot in common.
- From the previous example:
 - The $O(n^2)$ category had the following functions in it:
 - $f(n)=3n^2+n$
 - $f(n)=4n^2+2n$
 - The $O(n)$ category had the following functions in it:
 - $f(n)=200n+1$
 - $f(n)=2n+1$
 - $f(n)=8n+4$
 - The $O(1)$ category had the following functions in it:
 - $f(n)=10$

Big O Complexity Classes

- By doing big O analysis and placing functions into general categories we can determine something about its relative performance.
- In the car example, if we know a car belongs to the SUV category, we have some idea about its performance relative to the other categories. The car is likely faster than an economy car but slower than race car.
- Similarly with functions, if we know a function belongs to the $O(n)$ category then we know something about its performance. It is faster than functions in $O(n^2)$ but slower than functions in $O(1)$.

Big O Complexity Classes

- **End of Slides**

End of Slides