Algorithms

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

Today's Lecture

- Asymptotic Analysis
- Dominance
- Asymptotic Bounds and Input Data Cases
- Analyzing Code
- Space Complexity

- From a speed perspective, which category does this vehicle belong to?

???

Category 1
Sports cars

Category 2
Full size cars

Category 3
Economy cars

Categorizing Cars by Speed

- From a speed perspective, which category does this vehicle belong to?

Answer

It belongs to category 2 (full size cars)

Category 1
Sports cars

Category 2
Full size cars

Category 3
Economy cars

Categorizing Cars by Speed

- In the previous example we put a car into a category based on speed.
- By figuring out the category a car belongs to we can get a general idea about how fast it can drive.
- If we are told that a car belongs to the sports car category, we may not know its exact characteristics, but we do have a general idea about its capabilities.

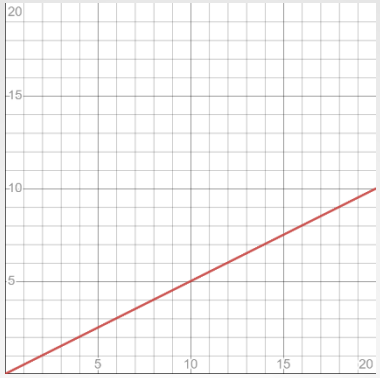
Categorizing Cars by Speed

- Asymptotic analysis allows us to put algorithms into categories just like we did with the cars.
- We just want to be able to identify an algorithm's speed in terms of a general category.
- If we are told that an algorithm belongs to a certain category, we may not know its exact characteristics, but we do have a general idea about its capabilities.

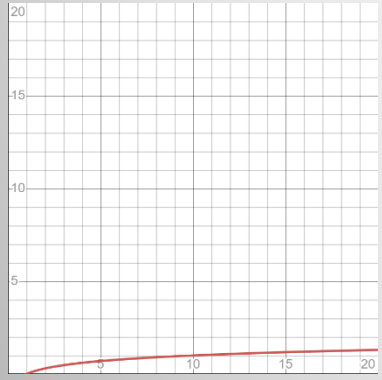
Categorizing Cars by Speed vs Asymptotic Analysis

- Which category does the following function belong to?

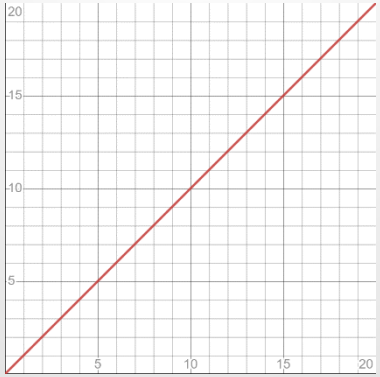
???



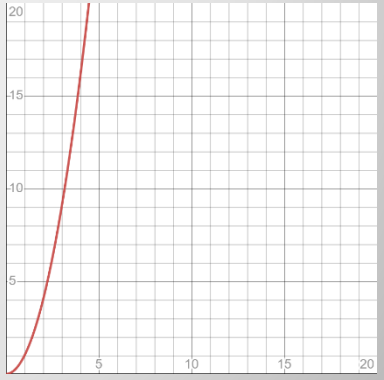
Category 1
Log



Category 2
Linear

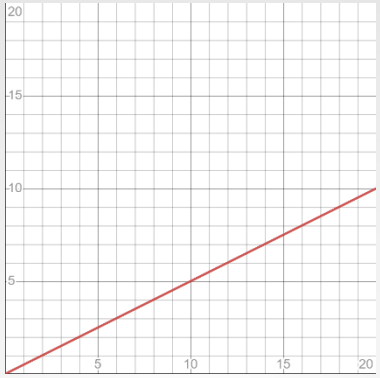


Category 3
Quadratic



Categorizing Functions

- Which category does the following function belong to?



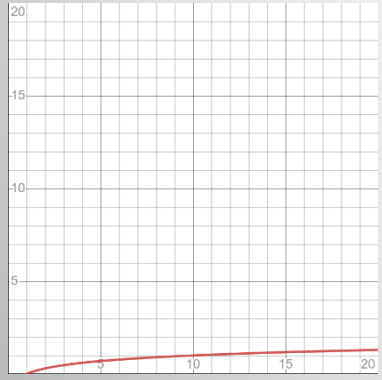
Answer

It belongs to category 2 (linear)

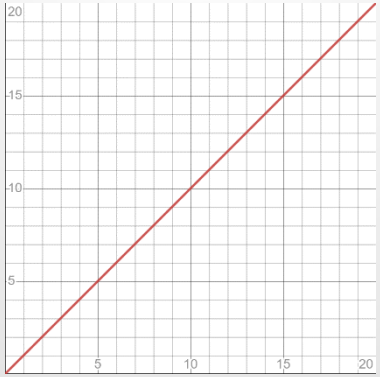
f(n) = ½ * n

This function grows linearly. It is not exactly the same as other linear functions, but it does grow similarly to them as n gets large (as opposed to log or quadratic functions).

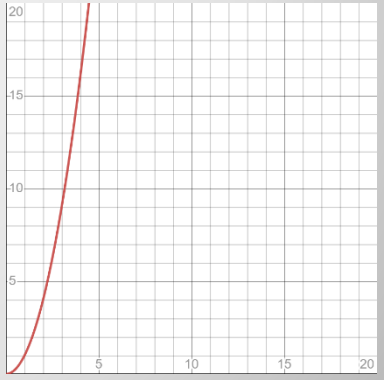
Category 1
Log



Category 2
Linear

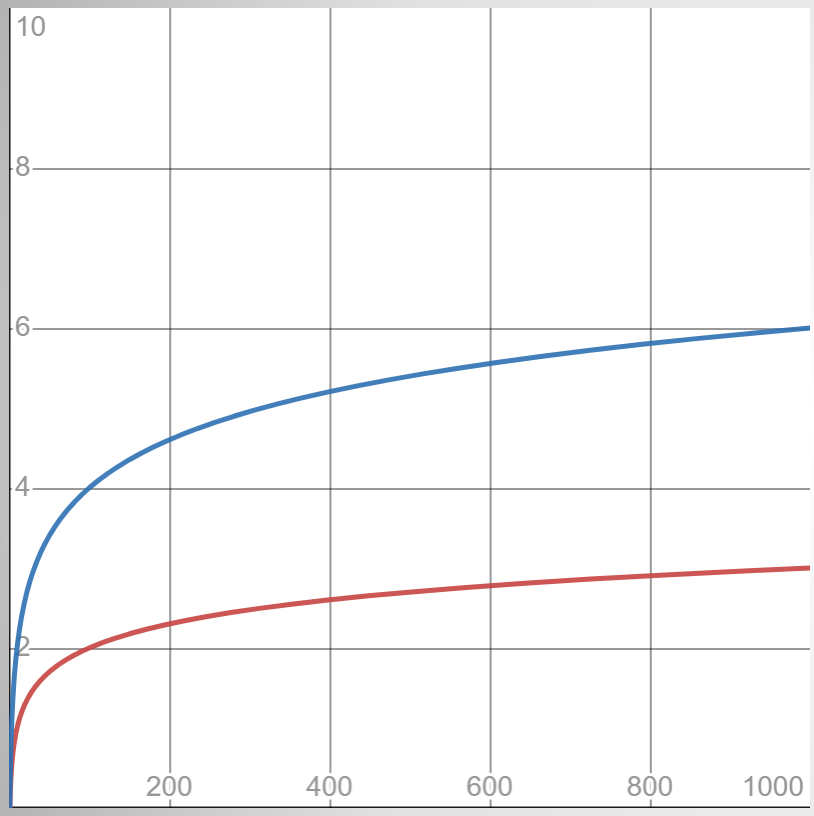


Category 3
Quadratic



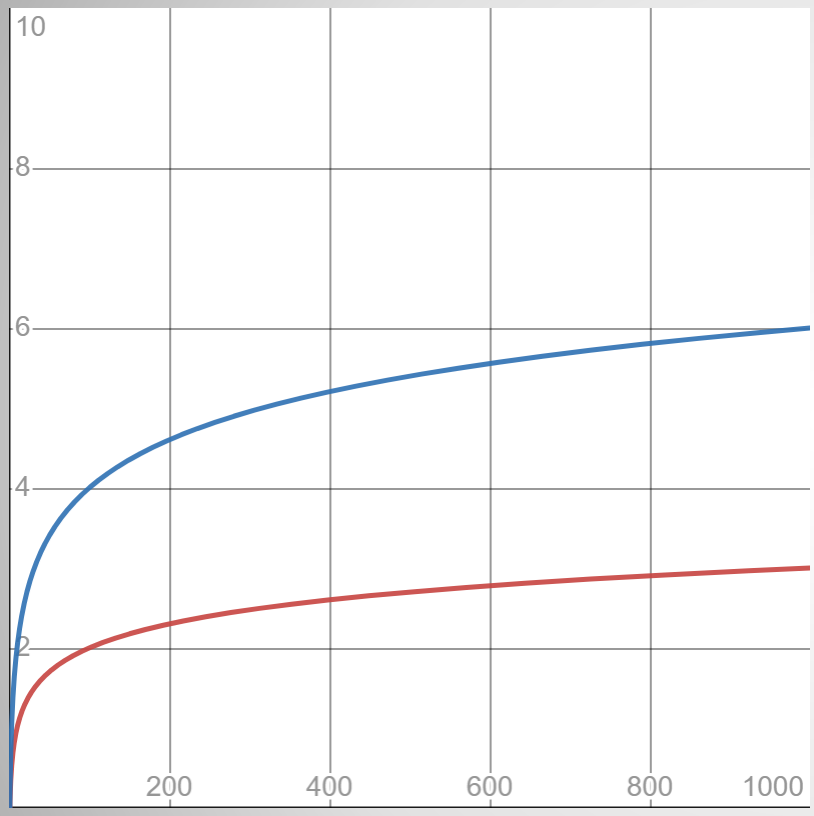
Categorizing Functions

- Which function returns higher values as n grows?



Comparing Functions

- Which function returns higher values as n grows?



2 log n

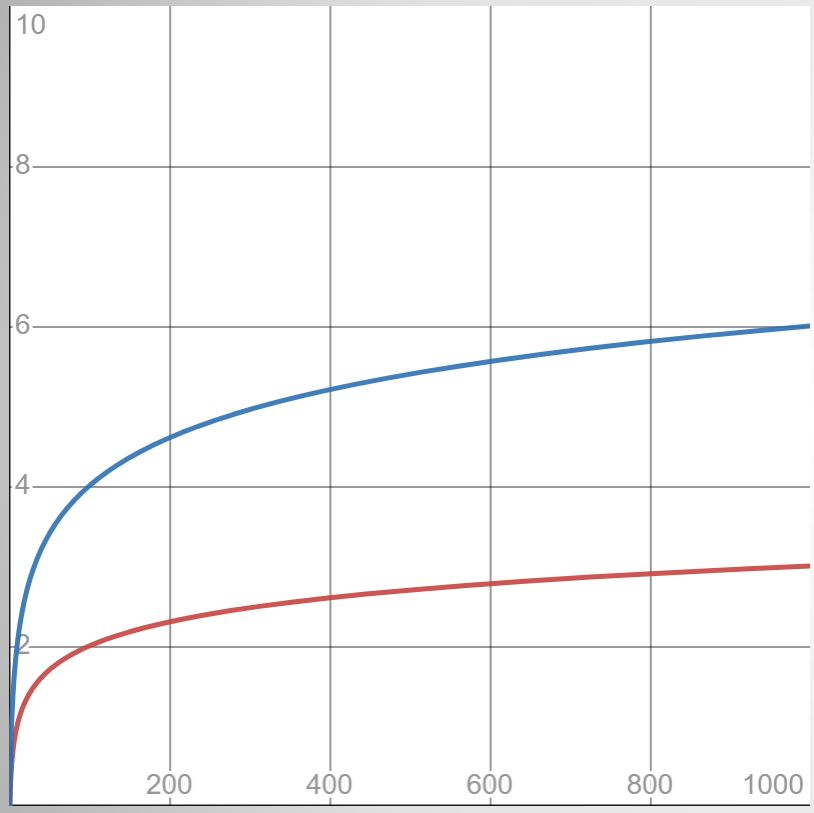
log n

Answer

2 log n returns "higher" values as n grows

Comparing Functions

- What constant value (say c) can we multiply log n by to make it return higher values than 2 log n?



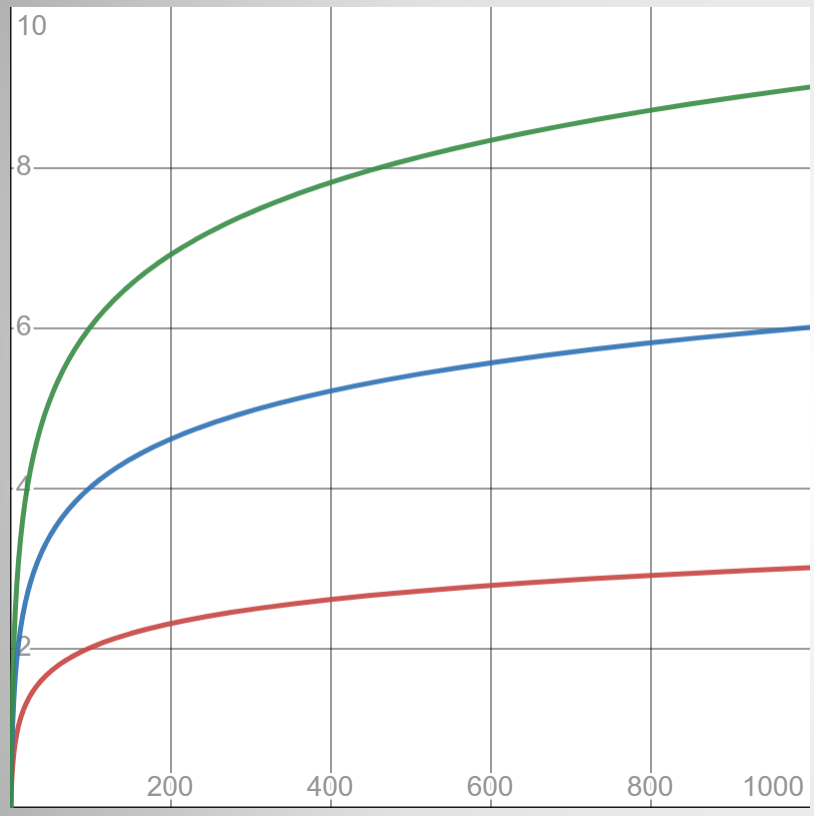
2 log n

c log n

Set c to a value that will make this line go above the blue line.

Manipulating Functions

- What constant value (say c) can we multiply log n by to make it return higher values than 2 log n?



3 log n (c=3)

2 log n

log n

Answer

Yes! We can set c to 3.

3 log n returns higher values than 2 log n as n grows.

Manipulating Functions

- Now on to asymptotic analysis…

Asymptotic Analysis

- **Asymptotic Efficiency of Algorithms** - Look at input sizes large enough to make only the order of growth of the running time relevant.
- Focus on how the running time of an algorithm increases with the size of the input in the limit, as the size of the input increases without bound.
- Most of the time an algorithm that is asymptotically more efficient will be the best choice for all but very small inputs.

Taken from: Introduction to Algorithms, 3rd edition, by Cormen, Leiserson, Rivest, and Stein, 2009.

Asymptotic Analysis

Asymptotic Notation

- We will cover the following three asymptotic notations.
- **Big O**. Upper bound. g(n) is an upper bound of f(n).

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

- **Big Omega**. Lower bound. g(n) is a lower bound of f(n).

$f(n) \in \Omega(g(n))$

- **Big Theta**. Upper and lower bound. g(n) is both and upper and lower bound of f(n).

$f(n) \in \Theta(g(n))$

The element of symbol ($\in$) is used above because $O()$, $\Omega()$, and $\Theta()$ are sets of functions.

Big O

- **Big O**. Bounded above.

$$O(g(n)) = \{ f(n): there\ exists\ positive\ constants\ c, n_0\ such\ that\ 0 \leq f(n) \leq cg(n) for\ all\ n > n_0 \}$$

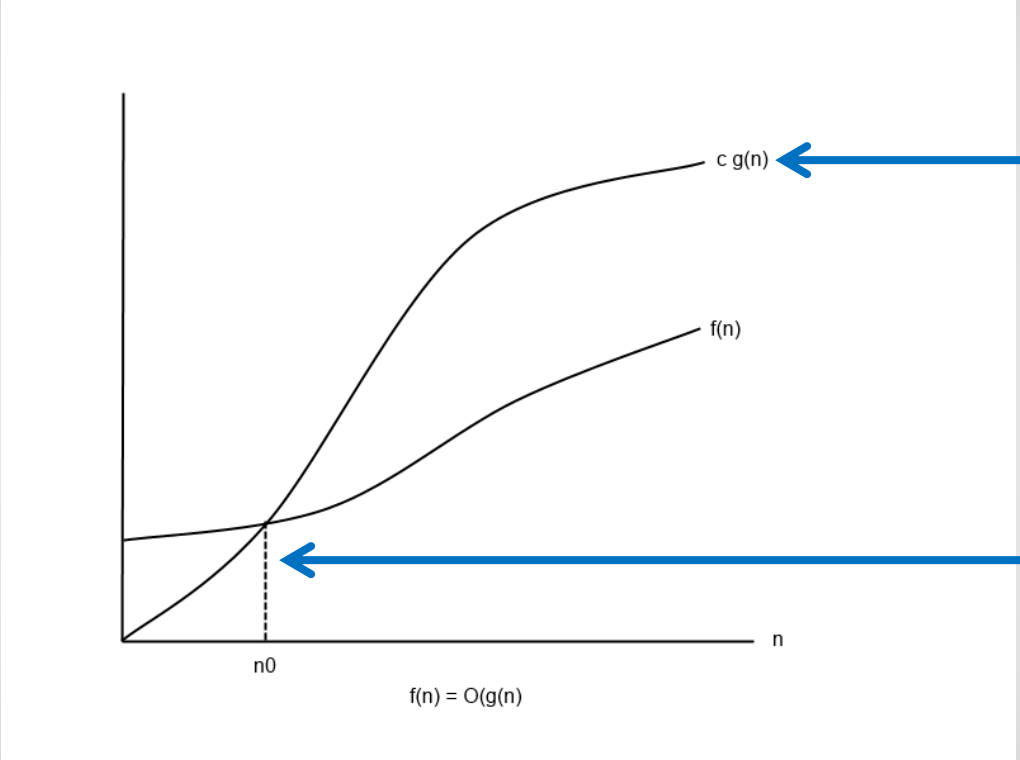
- **Show Upper Bound. To show that f(n) is an element of O(g(n)) we must find constants c and n_0 to make the above true.**

- The next slide shows a graph containing c and n_0…

- **Big O**. Bounded above.

$$O(g(n)) = \{ f(n): there\ exists\ positive\ constants\ c, n_0\ such\ that\ 0 \leq f(n) \leq \boldsymbol{c}g(n) for\ all\ n > \boldsymbol{n_0} \}$$

Goal
Find values of c and n_0 that make the relation true (any positive constants are fine)



Bounded Above

Beyond n_0 f(n) stays BELOW the bound

Taken from: Introduction to Algorithms, 3rd edition, by Cormen, Leiserson, Rivest, and Stein, 2009.

Big O

Sample O()

- **Show: 2n ∈ O(n)**
- Show that this relation holds: $0 \leq f(n) \leq \boldsymbol{c}g(n)\ for\ all\ n > \boldsymbol{n_0}$
- Given 2n ∈ O(n) what are f(n) and g(n)?

f(n) g(n)

So f(n)=2n, g(n)=n

- Plugin in f(n) and g(n) and then find positive constants c and n_0 such that the relation is true:

$$0 \leq f(n) \leq \boldsymbol{c}g(n)\ for\ all\ n > \boldsymbol{n_0}$$

Replace f(n) and g(n) in relation → $0 \leq 2n \leq \boldsymbol{c}n\ for\ all\ n > \boldsymbol{n_0}$

What value of c will make cn GREATER than 2n?

What value of n_0 will guarantee that 2n is ALWAYS less than cn?

Show f(n) is O(g(n))

Sample O() (continued)

- **Show: 2n ∈ O(n)**

  f(n) = 2n

  g(n) = n

Show the relation: $0 \leq f(n) \leq \boldsymbol{c}g(n)\ for\ all\ n > n_0$

Fill in f(n) and g(n): $0 \leq \boldsymbol{2n} \leq \boldsymbol{cn}\ for\ all\ n > n_0$

Will c=1 work? $0 \leq 2n \leq \boldsymbol{1n}\ for\ all\ n > n_0$

Is there a value for n_0 where the relation holds?

To find n_0, try some values of n to see what happens as n gets larger:

n	2n	1n
1	2	**1**
10	20	**10**
100	200	**100**
1000	2000	**1000**
10000	20000	**10000**

1n is always less. BAD!

c=1 DOES NOT WORK!!!

As n increases 2n will always be greater than 1n. There is no value of n where 2n becomes greater than 1n. So, there is no value of n_0 that will make the relation true.

Show f(n) is O(g(n))

Sample O() (continued)

- **Show: 2n ∈ O(n)**

  f(n) = 2n

  g(n) = n

  Show the relation: $0 \leq f(n) \leq \boldsymbol{c}g(n)\ for\ all\ n > n_0$

  Fill in f(n) and g(n): $0 \leq \boldsymbol{2n} \leq \boldsymbol{cn}\ for\ all\ n > n_0$

  Will c=3 work? $0 \leq 2n \leq \boldsymbol{3n}\ for\ all\ n > n_0$

  To find n_0, try some values of n to see what happens as n gets larger:

n	2n	3n
1	2	**3**
10	20	**30**
100	200	**300**
1000	2000	**3000**
10000	20000	**30000**

YES, c=3 WORKS

As n increases 3n will always be greater than 2n. We can now choose an n_0. For example, n_0 = 10.

2n ∈ O(n) is true since there are constants c and n_0 that make the relation true.

- **Answer: c=3, n_0=10**
- There are many other values of c and n_0 that will work.

Show f(n) is O(g(n))

Summary - Showing f(n) ∈ O(g(n))

- To show that something is O() you must find constants c and n_0 that make the relation below true:

$$O(g(n)) = \{ f(n): there\ exists\ positive\ constants\ c, n_0\ such\ that \\ 0 \leq f(n) \leq \boldsymbol{c}g(n) for\ all\ n > \boldsymbol{n_0} \}$$

General Outline of Approach

1. Replace f(n) and g(n) with the specific functions.
2. Choose a value for c.
3. Check values of n to see what happens as n gets very large.
4. If there is a value of n where f(n) is always less than cg(n) then f(n) ∈ O(g(n)). Choose n_0 such that f(n) is always less than cg(n). If there are NO values of c and n_0 that make the relation true, then f(n) is NOT an element of O(g(n).
5. Your final answer should be the values of c and n_0 that make the relation true.

Showing O() Upper Bound

Inclass Exercise (Big O)

Answer the following?

- Is $3n^2 \in O(n^2)$?

In Class Exercise

Proof by Contradiction

- Assume you have a statement you want to prove, say P.
- With proof by contradiction, we first assume what we want to prove is not true. We then show that this results in something that is not possible.
- Outline of proof by contradiction (¬ stands for negation).
  - Assume the negation of P is true, ¬P
  - Find a contradiction (something that is not possible). This could mean we reach a conclusion that contradicts our original assumption or a conclusion that contradicts something we know of that is true. Use logical steps to reach the conclusion (for example, algebraic manipulations).
  - Therefore, our assumption cannot be true which means the negation of our original assumption is true. So, ¬¬P which implies P.

Show f(n) is not O(g(n))

Sample O()

- **Show: n^2 is NOT ∈ O(n)**
- Use proof by contradiction.
- We want to show that values of c and n_0 cannot exist.

- First, assume the opposite is true. ¬(**n^2 is NOT ∈ O(n)**) is the same as $n^2 \in O(n)$
- If $n^2 \in O(n)$ then there are c and n_0 that make the following true: $0 \leq n^2 \leq cn \; for \; all \; n > \boldsymbol{n_0}$
- Use algebra to simplify the relation (divide by n in this case).
- We get the following: $0 \leq n \leq c \; for \; all \; n > \boldsymbol{n_0}$
- The above will fail for values of n > c. THIS IS A CONTRADICTION! Remember, n will go to infinity, so n is guaranteed to exceed c at some point (c is constant). When n exceeds c, we have a contradiction of our original assumption.
- This contradiction means that our assumption $n^2 \in O(n)$ cannot be true. So ¬($n^2 \in O(n)$) must be true.
- Therefore, **n^2 is NOT ∈ O(n)**.

Show f(n) is not O(g(n))

Inclass Exercise (Big O)

Answer the following?

- Is $n^3 \in O(n^2)$?

In Class Exercise

Big Omega

- **Big Omega**. Bounded below.

$$\Omega(g(n)) = \{ f(n): there\ exists\ positive\ constants\ c, n_0\ such\ that\ 0 \leq cg(n) \leq f(n)\ for\ all\ n > n_0 \}$$

cg(n) and f(n) are switched when compared to O()

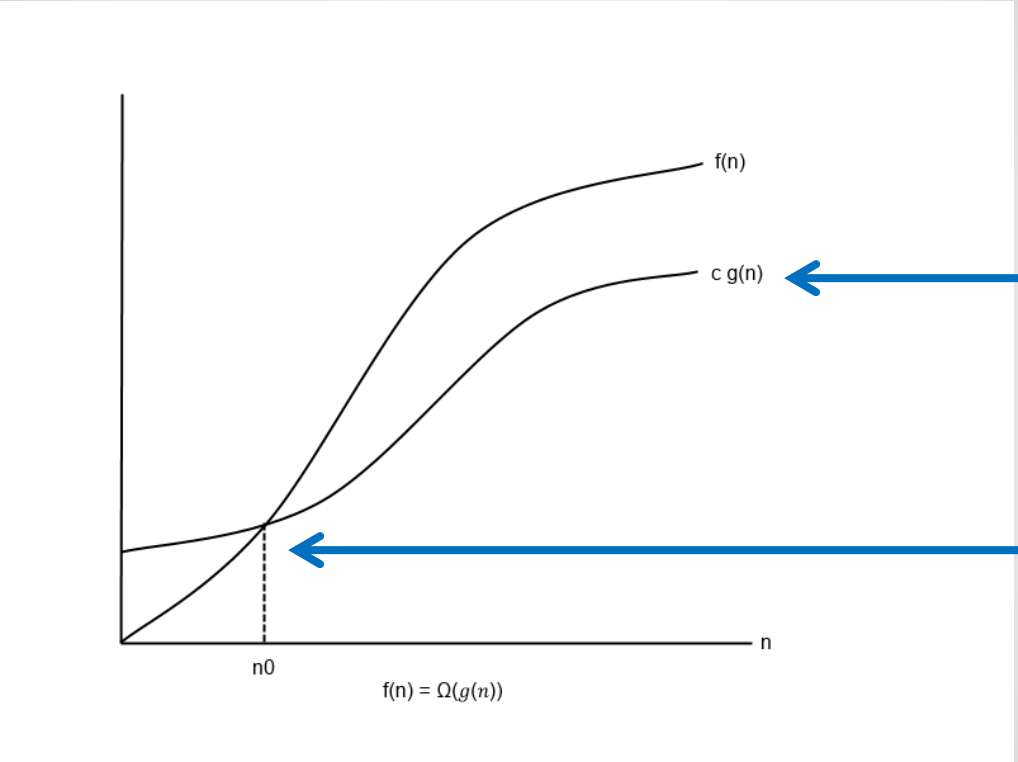
- **Show Lower Bound. To show that f(n) is an element of Ω(g(n)) we must find constants c and n_0 to make the above true.**

- The next slide shows a graph containing c and n_0…

- **Big Omega**. Bounded below.

$$\Omega(g(n)) = \{ f(n): there\ exists\ positive\ constants\ c, n_0\ such\ that\ 0 \leq cg(n) \leq f(n)\ for\ all\ n > n_0 \}$$

Goal
Find values of c and n_0 that make the relation true (any positive constants are fine)



Bounded Below

Beyond n_0 f(n) stays ABOVE the bound

Taken from: Introduction to Algorithms, 3rd edition, by Cormen, Leiserson, Rivest, and Stein, 2009.

Big Omega

Sample Ω()

- **Show: 3n ∈ Ω(n)**

- Given 3n ∈ Ω(n) what are f(n) and g(n)?

f(n) g(n)

So f(n)=3n, g(n)=n

- We must find positive constants c and n_0 such that the following holds:

$$0 \leq \boldsymbol{c}g(n) \leq f(n) \; for \; all \; n > \boldsymbol{n_0}$$

Replace f(n) and g(n) in relation →

$$0 \leq \boldsymbol{c}n \leq 3n \; for \; all \; n > \boldsymbol{n_0}$$

What value of c will make cn LESS than 3n?

What value of n_0 will guarantee that 3n is always greater than cn?

Show f(n) is Ω(g(n))

Sample Ω() (continued)

- **Show: 3n ∈ Ω(n)**

  f(n) = 3n

  g(n) = n

Show the relation: $0 \leq cg(n) \leq f(n)\ for\ all\ n > n_0$

Fill in f(n) and g(n): $0 \leq cn \leq 3n\ \ for\ all\ n > n_0$

Will c=5 work? $0 \leq 5n \leq 3n\ \ for\ all\ n > n_0$

Is there a value for n_0 where the relation holds?

To find n_0, try some values of n to see what happens as n gets larger:

n	5n	3n
1	**5**	3
10	**50**	30
100	**500**	300
1000	**5000**	3000
10000	**50000**	30000

5n is always above. BAD!

c=5 DOES NOT WORK!!!

As n increases 5n will always be greater than 3n. There is no value of n where 5n becomes LESS than 3n. So, there is no value of n_0 that will make the relation true.

Show f(n) is Ω(g(n))

Sample Ω() (continued)

- **Show: 3n ∈ Ω(n)**

  f(n) = 3n

  g(n) = n

  Show the relation: $0 \leq cg(n) \leq f(n)\ for\ all\ n > n_0$

  Fill in f(n) and g(n): $0 \leq c\boldsymbol{n} \leq \boldsymbol{3n}\ for\ all\ n > n_0$

  Will c=2 work? $0 \leq \boldsymbol{2}n \leq 3n\ for\ all\ n > n_0$

  To find n_0, try some values of n to see what happens as n gets larger:

n	2n	3n
1	**2**	3
10	**20**	30
100	**200**	300
1000	**2000**	3000
10000	**20000**	30000

YES, c=2 WORKS

As n increases 2n will always be less than 3n. We can now choose an n_0. For example, n_0 = 10.

3n ∈ Ω(n) is true since there are constants c and n_0 that make the relation true.

- **Answer: c=2, n_0=10**
- There are many other values of c and n_0 that will work.

Show f(n) is Ω(g(n))

Inclass Exercise (Big Omega)

Answer the following?

- Is $3n^2 \in \Omega(n^2)$?
- Is $3n^2 \in \Omega(n)$?
- Is $3n^2 \in \Omega(n^3)$?

In Class Exercise

- **Big Theta**. Bounded both above and below.

$$\Theta(g(n)) = \{ f(n): there\ exists\ positive\ constants\ c1, c2, n_0\ such\ that\ 0 \leq c1g(n) \leq f(n) \leq c2g(n) for\ all\ n > n_0 \}$$

Lower bound

Upper bound

- **Show Both Upper and Lower Bounds. To show that f(n) is an element of Θ(g(n)) we must find constants c1, c2, and n_0 to make the above true.**

- The next slide shows a graph containing c1, c2, and n_0…

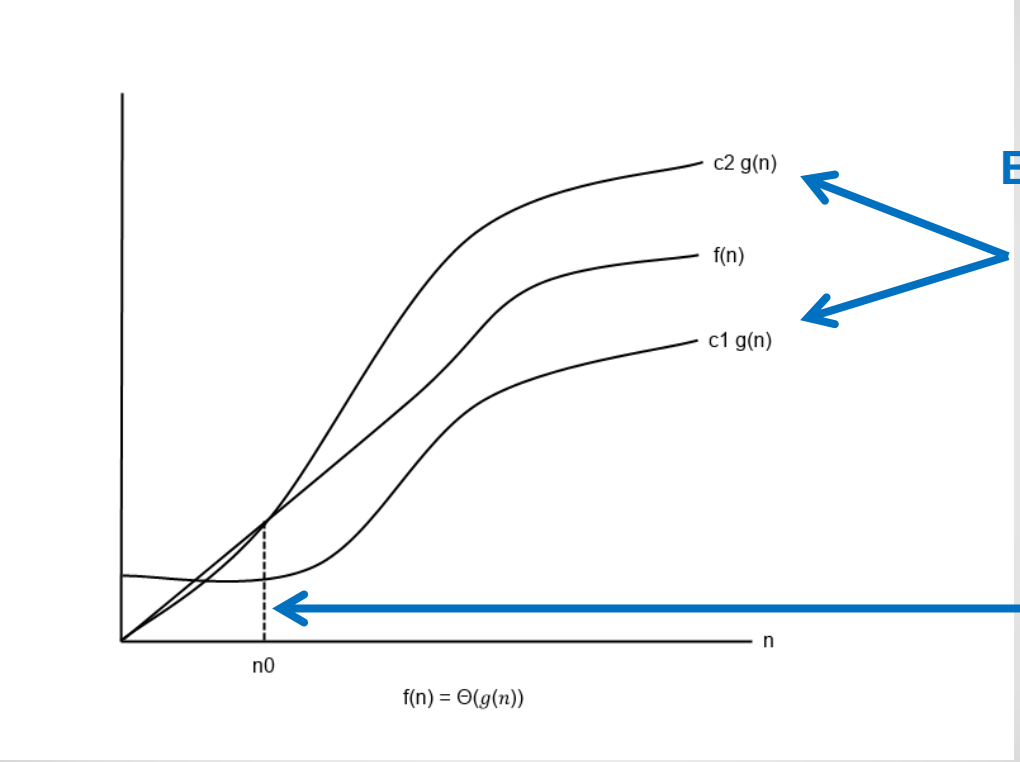
Big Theta

- **Big Theta**. Bounded both above and below.

$$\Theta(g(n)) = \{ f(n): there\ exists\ positive\ constants\ c1, c2, n_0\ such\ that\ 0 \leq c1g(n) \leq f(n) \leq c2g(n) for\ all\ n > n_0 \}$$

Goal

Find values of c1, c2, and n_0 that make the relation true (any positive constants are fine)



Bounded Above and Below

Beyond n_0 f(n) stays BETWEEN both bounds

Taken from: Introduction to Algorithms, 3rd edition, by Cormen, Leiserson, Rivest, and Stein, 2009.

Big Theta

Sample Θ()

- **Show: 4n ∈ Θ(n)**

- Given 4n ∈ Θ(n) what are f(n) and g(n)?

f(n) g(n)

So f(n)=4n, g(n)=n

- We must find positive constants c1, c2, and n_0 such that the following holds:

$$0 \leq c1g(n) \leq f(n) \leq c2g(n) for\ all\ n > n_0$$

Replace f(n) and g(n) in relation →

$$0 \leq \boldsymbol{c1}\ n \leq 4n \leq \boldsymbol{c2}\ n\ for\ all\ n > n_0$$

What value of n_0 will guarantee that 4n is BETWEEN both bounds?

What value of c1 will make c1n LESS than 4n?

What value of c2 will make c2n GREATER than 4n?

Show f(n) is Θ(g(n))

Sample Θ() (continued)

- **Show: 4n ∈ Θ(n)**

  f(n) = 4n

  g(n) = n

Is there a value for n_0 where the relation holds?

Show the relation: $0 \le c1\ g(n) \le f(n) \le c2\ g(n)\ for\ all\ n > n_0$

Fill in f(n) and g(n): $0 \le c1\ \boldsymbol{n} \le \boldsymbol{4n} \le c2\ \boldsymbol{n}\ for\ all\ n > n_0$

Will c1=5,c2=8 work? $0 \le \boldsymbol{5}n \le 4n \le \boldsymbol{8}n\ for\ all\ n > n_0$

To find n_0, try some values of n to see what happens as n gets larger:

n	5n	4n	8n
1	**5**	4	**8**
10	**50**	40	**80**
100	**500**	400	**800**
1000	**5000**	4000	**8000**
10000	**50000**	40000	**80000**

BAD lower bound

GOOD upper bound

c1=5, c2=8 DOES NOT WORK!!!

As n increases 5n will always be greater than 4n. There is no value of n where 5n becomes LESS than 4n. So, there is no value of n_0 that will make the relation true.

Show f(n) is Θ(g(n))

Sample Θ() (continued)

- **Show: 4n ∈ Θ(n)**

  f(n) = 4n

  g(n) = n

  Show the relation: $0 \le c1\ g(n) \le f(n) \le c2\ g(n)\ for\ all\ n > n_0$

  Fill in f(n) and g(n): $0 \le c1\ \boldsymbol{n} \le \boldsymbol{4n} \le c2\ \boldsymbol{n}\ for\ all\ n > n_0$

  Will c1=3,c2=8 work? $0 \le \boldsymbol{3}n \le 4n \le \boldsymbol{8}n\ for\ all\ n > n_0$

  To find n_0, try some values of n to see what happens as n gets larger:

n	3n	4n	8n
1	**3**	4	**8**
10	**30**	40	**80**
100	**300**	400	**800**
1000	**3000**	4000	**8000**
10000	**30000**	40000	**80000**

YES, c1=3, c2=8 WORKS

As n increases 3n will always be BETWEEN both bounds. We can now choose an n_0. For example, n_0 = 10.

4n ∈ Θ(n) is true since there are constants c1, c2, and n_0 that make the relation true.

- **Answer: c1=3, c2=8, n_0=10**
- There are many other values of c1, c2, and n_0 that will work.

Show f(n) is Θ(g(n))

Inclass Exercise (Big Theta)

Answer the following?

- Is $2n \in \Theta(n^2)$?
- Is $2n \in O(n^2)$? ⟵ **Yes, that is big O**
- Is $3n \in \Theta(n)$?

In Class Exercise

- **Big O**. Bounded above.

$$O(g(n)) = \{ f(n): there\ exists\ positive\ constants\ c, n_0\ such\ that\ 0 \leq f(n) \leq cg(n) for\ all\ n > n_0 \}$$

- **Big Omega**. Bounded below.

$$\Omega(g(n)) = \{ f(n): there\ exists\ positive\ constants\ c, n_0\ such\ that\ 0 \leq cg(n) \leq f(n)\ for\ all\ n > n_0 \}$$

- **Big Theta**. Bounded both above and below.

$$\Theta(g(n)) = \{ f(n): there\ exists\ positive\ constants\ c1, c2, n_0\ such\ that\ 0 \leq c1g(n) \leq f(n) \leq c2g(n) for\ all\ n > n_0 \}$$

Taken from: Introduction to Algorithms, 3rd edition, by Cormen, Leiserson, Rivest, and Stein, 2009.

Asymptotic Notation Summary

- Check math fundamentals slides if you need a review of basic limits…

Limits

- Now we will move on to dominance…

Dominance

Limits and Dominance

- If the denominator overwhelms or dominates the numerator it will cause the limit to go to 0.
- In the following example, if f(n) dominates g(n) then the limit will go to 0.

$$\lim_{n \to \infty} (g(n)/f(n))$$

- For example…

Find Dominance Using Limit

- Assume the following:

$f(n) = n^2$

$g(n) = n$

- Find the limit:

$$\lim_{n\to\infty}(g(n)/f(n))$$

- Substitute functions:

$$\lim_{n\to\infty}(n/n^2)$$

n^2 dominates n because this limit goes to 0

- Simplify:

$$\lim_{n\to\infty}(1/n)$$

- Result:

$$\lim_{n\to\infty}(1/n) = 0$$

Dominance and Simplifying

- We can use the fact that one function dominates another to further simplify expressions when doing asymptotic analysis.
- Assume $f(n)=n^2+n$
- We can simplify this expression to the following since n^2 dominates n: $f(n) = n^2$

- If we are not sure about if one term dominates another, we can plug those terms into the limit and do the calculation.
- Put the term you think should dominate in f(n) and the other in g(n).
- If the following limit goes to 0 then f(n) dominates g(n)

$$\lim_{n\to\infty}(g(n)/f(n))$$

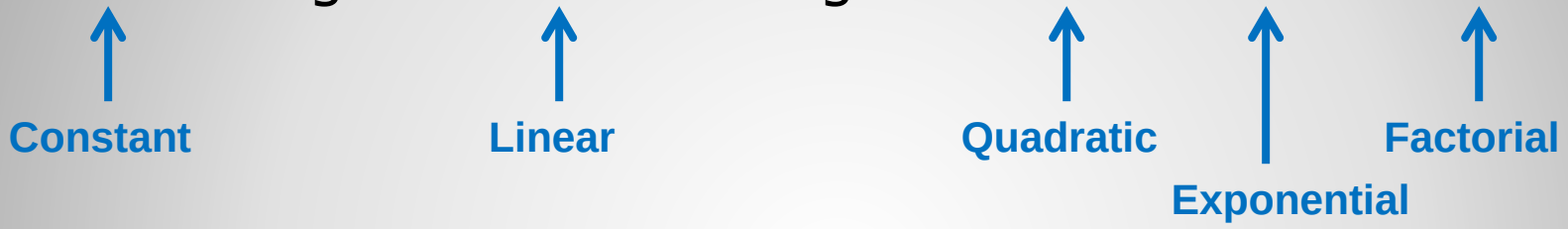
Much Less Than and Much Greater Than

- The << symbol in mathematics can be used to indicate that one function is much less than another function.
- For example: $n << n^2$
- This says that n is much less than n^2. Basically, n is dominated by n^2.
- The >> symbol means much greater than.

Selected Dominance Relationships

- Here is a hierarchy of selected functions showing dominance relationships.

- $1 << \log n << n << n \log n << n^2 << 2^n << n!$



- 1 stands for any constant function.
- log n stands for the base 2 log (base 2 log is $\log_2 n$).
- n log n is n times log n
- The is not an exhaustive list.

- Note – The following are also true for >>:

$n! >> 2^n >> n^2 >> n \log n >> n >> \log n >> 1$

- Now we will move on to asymptotic bounds and input data cases…

Asymptotic Bounds and Input Data Cases

Cases

- **Input Cases** - We are also interested in how an algorithm performs with different cases of input. The amount of work an algorithm will need to do can vary according to the input data.
  - **Best Case Input** – How does it perform with the best possible set of input data.
  - **Average Case Input** – How does it perform with an average set of input data.
  - **Worst Case Input** – How does it perform with the worst possible set of input data.

Bound != Input Case

- Asymptotic bounds indicate what happens as the input size moves towards infinity.
- We can calculate asymptotic bounds separately for the best, average, and worst cases of input data.
- Big O is sometimes confused with "worst case" but they are not the same thing.
- Big O can be applied to best, average, and worst cases of input data (big O is just an upper bound).

Bounds != Input Case

- Analyzing code…

Analyzing Code

Math Review - log

- Log is exponent!

- $\log_a b = x$

- Same as above but in exponential form:

$a^x = b$

- We want to find the power that a must be raised to that will result in b.
- For example:

$\log_2 8 = x$

$2^x = 8$

- Solve for x. 2 raised to what power is 8?

Answer: 3.

Math Review – log

Math Review - log

- As n grows larger the result of $\log_2 n$ grows slowly.

n	Log of n	Result
2	$\log_2 2$	1
4	$\log_2 4$	2
8	$\log_2 8$	3
16	$\log_2 16$	4
32	$\log_2 32$	5
64	$\log_2 64$	6
128	$\log_2 128$	7
256	$\log_2 256$	8
512	$\log_2 512$	9
1024	$\log_2 1024$	10

n is up to 1024 but $\log_2$ 1024 is only up to 10

- Assume n=1024.
- How many times do you need to divide n in half to make it get to 1?

Math Review - log

- Assume n=1024.
- How many times do you need to divide n in half to make it get to 1?

Answer

- If n is 1024 and you divide it by 2 you get 512.
- If n is 512 and you divide it by 2 you get 256.
- If n is 256 and you divide it by 2 you get 128.
- If n is 128 and you divide it by 2 you get 64.
- If n is 64 and you divide it by 2 you get 32.
- If n is 32 and you divide it by 2 you get 16.
- If n is 16 and you divide it by 2 you get 8.
- If n is 8 and you divide it by 2 you get 4.
- If n is 4 and you divide it by 2 you get 2.
- If n is 2 and you divide it by 2 you get 1.

10 divisions

The log function tells us how many times we divide by the base to get down to 1. For example: $\log_2 1024 = 10$

Math Review - log

- Assume n=64.
- How many times do you need to divide n by 4 to make it get to 1?

Answer

- If n is 64 and you divide it by 4 you get 16.
- If n is 16 and you divide it by 4 you get 4.
- If n is 4 and you divide it by 4 you get 1.

3 divisions

- $\log_4 64 = 3$
- $4^3 = 64$

The base is 4 in this example. We divide 64 by 4 a total of 3 times to make it get to 1.

Math Review - log

Math Review – log and exponential

- The log is the inverse of the exponential.
- $2^{10}=2*2*2*2*2*2*2*2*2*2=1024$.
- $\log_2 1024 = 10$
- Now divide 1024 by 2. This will cause one 2 to be removed from 2^{10} which turns it into 2^9.
- $2^{10}=2*2*2*2*2*2*2*2*2*2 = 1024$ →
- $2^9 =2*2*2*2*2*2*2*2*2 = 512$

Divide by 2

- $\log_2 1024=10$ and the $\log_2 512=9$. The log goes down by 1 when dividing by 2.
- Note: The log goes up by 1 when multiplying the number by 2. For example, 512*2=1024 so the log goes up by 1 in this case.

Analyzing Code

- Determine the relationship between the number of instructions executed and the number of items.
- How much "work" gets done with respect to the number of items (n).
- Loops can cause lots of work to be done.
- Methods can cause lots of work to be done.

Analyzing Code - Overview

Analyzing Loops

- Look carefully at loops to see how much work they do (how many times the loop body executes).
- Look at the loop control variable and think about how it is being updated.
- The loop may go significantly more or less than n times depending on how the loop control variable is being updated (it does not have to to use ++ or – when updating).

Analyzing Code - Loops

Analyzing Methods

- Methods will cause a certain amount of work to be done.
- Assume method A() has a time complexity of O(n).
- If method B() calls method A() then method B() will be doing at least O(n) work (possibly more depending on the code in B()).

Analyzing Code – Methods

Analyzing Code

- O(1) - Executed instructions does not depend on the number of items (n) in any way. The number of executed instructions remains constant.
- O(log n) - Executed instructions has a logarithmic relationship with the number of items (n). For example, a binary search. It keeps splitting the number of items in half.
- O(n) – Executed instructions has a linear relationship with the number of items (n). For example, a loop that processes every item.
- $O(n^2)$ - Executed instructions has a quadratic relationship with the number of items (n). For example, nested loops that both depend on the number of items (n).

Analyzing Code - Overview

Analyze Code – Example 1

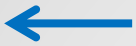
Declare int a
a = 1
Print a

What is the upper bound?

Analyzing Code

Analyze Code – Example 1

```
Declare int a
a = 1
Print a
```



Does NOT depend on the number of items (n)

What is the upper bound? O(1)

- There are 3 instructions below.
- Each instruction is executed once no matter what.
- The number of times each instruction runs does NOT depend on the number of items (n) in any way.

Answer: O(3) → O(3*1) → O(1)

Analyzing Code

Analyze Code – Example 2

```
Declare int n
Declare int i = 0

Read n from keyboard

While (i<n)
   print "I love CS"
   i++
EndWhile
```

What is the upper bound?

Analyzing Code

Analyze Code – Example 2

```
Declare int n
Declare int i = 0

Read n from keyboard
```

O(1)

```
While (i<n)
   print "I love CS"
   i++
EndWhile
```

O(n)

Answer: O(1) + O(n) → O(n)

What is the upper bound? O(n)

Analyzing Code

Analyze Code – Example 3

```
Declare int n
Declare int i = 0

print "I love CS"
print "I love CS"
print "I love CS"
print "I love CS"
print "I love CS"
```

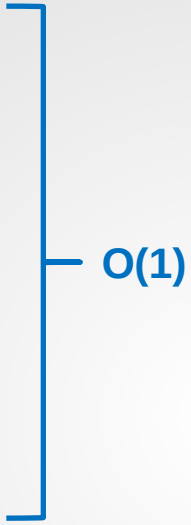
What is the upper bound?

Analyzing Code

Analyze Code – Example 3

```
Declare int n
Declare int i = 0

print "I love CS"
print "I love CS"
print "I love CS"
print "I love CS"
print "I love CS"
```



What is the upper bound? O(1)

Answer: O(1)

- The same instruction is being executed but it gets executed a constant number of times.

Analyzing Code

Analyze Code – Example 4

```
Declare int n
Declare int i = 0

While (i<5)
   print "I love CS"
   i++
EndWhile
```

What is the upper bound?

Analyzing Code

Analyze Code – Example 4

```
Declare int n
Declare int i = 0

While (i<5)
   print "I love CS"
   i++
EndWhile
```

O(1)

O(1)

What is the upper bound? O(1)

Answer: O(1) + O(1) → O(1)

- There is a loop but the number of times the loop body gets executed is constant.
- The loop does not depend on the number of items (n).

Analyzing Code

Analyze Code – Example 5

```
Declare int n
Declare int i = 0

Read n from keyboard

While (i<5)
   print n
   print "I love CS"
   i++
EndWhile
```

What is the upper bound?

Analyzing Code

Analyze Code – Example 5

```
Declare int n
Declare int i = 0

Read n from keyboard
```

O(1)

```
While (i<5)
   print n
   print "I love CS"
   i++
EndWhile
```

O(1)

What is the upper bound? O(1)

Answer: O(1) + O(1) → O(1)

- The value n is printed in the loop, but it does not change how many times the loop executes.

Analyzing Code

Analyze Code – Example 6

```
Declare int n
Declare int a = 0
Declare int b = 0

Read n from keyboard

While (a<n)
   While (b<n)
      print "I love CS"
      b++
   EndWhile
   a++
EndWhile
```

What is the upper bound?

Analyzing Code

Analyze Code – Example 6

```
Declare int n
Declare int a = 0
Declare int b = 0

Read n from keyboard

While (a<n)
   While (b<n)
      print "I love CS"
      b++
   EndWhile
   a++
EndWhile
```

O(1)

$O(n^2)$

What is the upper bound? $O(n^2)$

Answer: $O(1) + O(n^2) \rightarrow O(n^2)$

Analyzing Code

Analyze Code – Example 7

```
Boolean Search(int[] data, int target)
   Declare int startIndex = 0
   Declare int endIndex = data.length-1
   Declare int pivot

   While (startIndex<=endIndex)
      pivot = Floor((startIndex+endIndex)/2)
      if (target == data[pivot])
         return true
      else if (target>data[pivot])
         startIndex=pivot+1
      else
         endIndex=pivot-1
   EndWhile

   return false
```

What is the upper bound?

Analyzing Code

Analyze Code – Example 7

```
Boolean Search(int[] data, int target)
   Declare int startIndex = 0
   Declare int endIndex = data.length-1
   Declare int pivot

   While (startIndex<=endIndex)
      pivot = Floor((startIndex+endIndex)/2)
      if (target == data[pivot])
         return true
      else if (target>data[pivot])
         startIndex=pivot+1
      else
         endIndex=pivot-1
   EndWhile

   return false
```

O(1)

O(log n)

O(1)

What is the upper bound? O(log n)

- Answer: O(1) + O(log n) + O(1) → O(log n)
- The pivot keeps getting divided in half, so it is logarithmic.

Analyzing Code

Analyze Code – Example 8

```
Declare int n
Declare int choice
Declare int i = 0
Enter choice from keyboard

If (choice == 1)
   print "I love CS"
Else
   Enter n from keyboard
   While (i<n)
      print "I love CS"
      i++
   EndWhile
EndIf
```

What is the upper bound?

Analyzing Code

Analyze Code – Example 8

```
Declare int n
Declare int choice
Declare int i = 0
Enter choice from keyboard

If (choice == 1)
   print "I love CS"
Else
   Enter n from keyboard
   While (i<n)
      print "I love CS"
      i++
   EndWhile
EndIf
```

O(1)

O(1)

O(n)

O(n) (whole if statement)

What is the upper bound? O(n)

Answer: O(1) + O(n) → O(n)

- Even though the true part of the if may only run it is possible for the else to run.
- Since the else depends on the number of items, the whole if also depends on it.

Analyzing Code

Analyze Code – Example 9

```
void ShowMessage(int x)
   Declare int i = 0
   While (i<x)
      print "I love CS"
      i++
   EndWhile

Main
   Declare int n
   Enter n from keyboard
   ShowMessage(n)
```

What is the upper bound?

Analyzing Code

Analyze Code – Example 9

```
void ShowMessage(int x)
   Declare int i = 0
   While (i<x)
      print "I love CS"
      i++
   EndWhile
```

O(n)

```
Main
   Declare int n
   Enter n from keyboard
   ShowMessage(n)
```

O(1)

O(n)

What is the upper bound? O(n)

- Answer: O(1) + O(n) → O(n)

- The amount of work to do in the ShowMessage method depends on the value of parameter x.
- Main calls ShowMessage and passes in a value entered by the user (n), so the amount of work that gets done in ShowMessage will depend on n.

Analyzing Code

- Space complexity…

Space Complexity

Space Complexity

- Space Complexity - The total space used by the algorithm.

- Input Space – Memory space used by inputs.
  - Variables used to store input data (these could be collections).
- Auxiliary Space – Any other memory used during execution of the algorithm.
  - Variables used in calculations.
  - Extra collections that store copies of all the original data.

- n is the number of items we are processing.

- Taken from: https://en.wikipedia.org/wiki/Space_complexity

Space Complexity

Space Complexity Example 1

Declare int[] ar
Read n pieces of data into
  ar from a file

For all data in ar (i is loop variable)
  print ar[i]

What is the upper bound on space (give total, input, and auxiliary)?

Space Complexity

Space Complexity Example 1

What is the upper bound on space (give total, input, and auxiliary)?

```
Declare int[] ar
Read n pieces of data into
   ar from a file

For all data in ar (i is loop variable)
    print ar[i]
```

Answer

- Input Space: O(n)
- Auxiliary Space: O(1)

- Total Space: O(n) + O(1) → **O(n)**

Space Complexity

Space Complexity Example 2

What is the upper bound on space (give total, input, and auxiliary)?

```
Declare int[] salary
Declare int[] salaryWithRaise
Read n pieces of data into
   salary from a file

For all data in salary (i is loop variable)
    salaryWithRaise[i] = salary[i] + 10
```

Space Complexity

Space Complexity Example 2

What is the upper bound on space (give total, input, and auxiliary)?

```
Declare int[] salary
Declare int[] salaryWithRaise
Read n pieces of data into
   salary from a file

For all data in salary (i is loop variable)
    salaryWithRaise[i] = salary[i] + 10
```

There is an extra n element array allocated (salaryWithRaise) so O(n) auxiliary space is used

Answer

- Input Space: O(n)
- Auxiliary Space: O(n) + O(1) → O(n)

- Total Space: O(n) + O(n) → **O(n)**

Space Complexity

Space Complexity Example 3

```
void Show(int i, int[] a)
   If (i == a.length)
      return
  print a[i]
  Show(i+1, a)

Main
   Declare int[] ar
   Read n pieces of data into
      ar from a file
   Show(0, ar)
```

What is the upper bound on space (give total, input, and auxiliary)?

Space Complexity

Space Complexity Example 3

```
void Show(int i, int[] a)
   If (i == a.length)
      return
  print a[i]
  Show(i+1, a)

Main
   Declare int[] ar
   Read n pieces of data into
      ar from a file
   Show(0, ar)
```

What is the upper bound on space (give total, input, and auxiliary)?

Show. There are n+1 calls to show (because of recursion). Each recursive call allocates local variables. So, n sets of local variables will be on the call stack when the base case is reached. Calls to Show cost O(n) auxiliary space.

Main. O(n) space for input and O(1) space loop variables.

Answer

- Input Space: O(n)
- Auxiliary Space: O(n) + O(1) → O(n)
- Total Space: O(n) + O(n) → **O(n)**

Space Complexity

- **End of Slides**

End of Slides