CSE 12 – Basic Data Structures and Object-Oriented Design
Lecture 11

Greg Miranda & Paul Cao, Winter 2021

This lecture is being recorded

Announcements

- Quiz 11 due Wednesday @ 8am

- Survey 5 due Friday @ 11:59pm

- PA4 due Wednesday @ 11:59pm

Topics

- Questions on Lecture 11?

- Big Theta

- Sorting

Questions on Lecture 11?

O is an upper bound $(\leq)$

Ω is a *lower bound* $(\geq)$

best / average / worst

Algorithm

Steps

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

$f(n)$

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

We say a function f(n) is **"big-omega"** of another function g(n), and write f(n) ∈ **Ω**(g(n)), if there are positive constants c and n_0 such that:

- f(n) ≥ c g(n) for all n ≥ n_0.
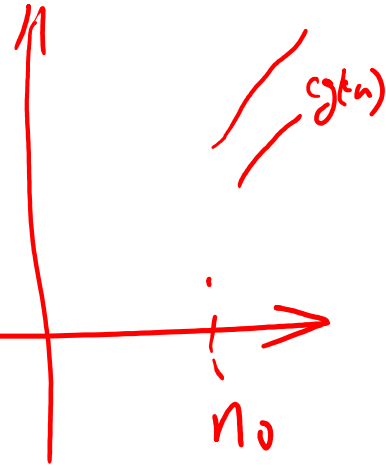
n is large

In other words, for large n, can you multiply g(n) by a constant and have it always be smaller than or equal to f(n)
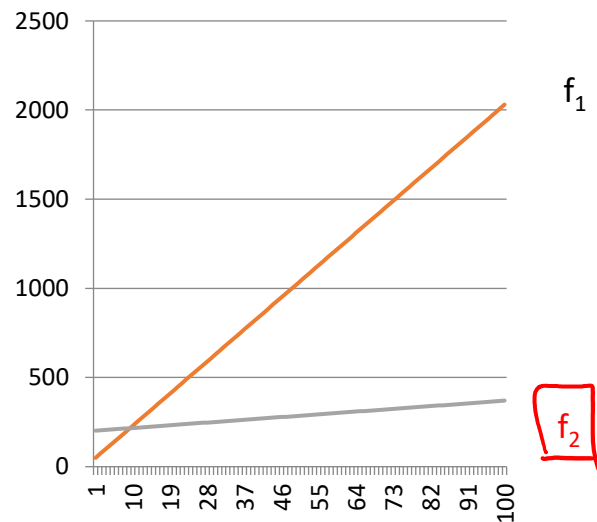
-f(n)

-g(n)

n_0

$$f_2 \in \Omega(f_1)$$

$n \in \Omega(n)$ ✓ n is large

$n \in O(n)$ ✓ $f_2 \geq \frac{1}{100} f_1$
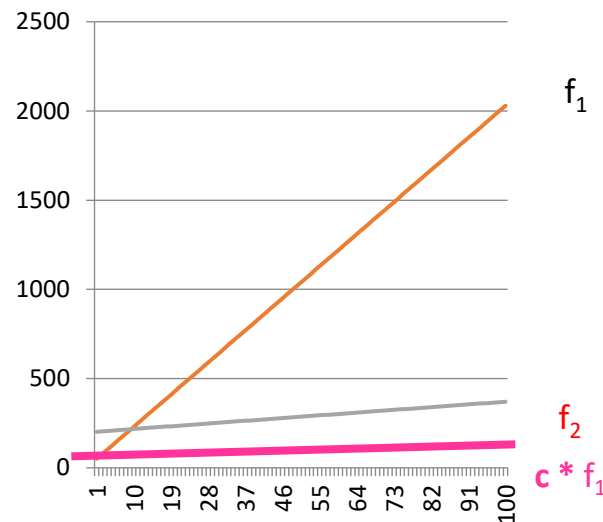
A. TRUE

B. FALSE

Why or why not?

In other words, for large n, can you multiply f_1 by a positive constant and have it always be smaller than f_2

f(n) ∈ **O**(g(n)), if there are positive constants c and n_0 such that $f(n) \leq c *$ g(n) for all $n \geq n_0$.

f(n) ∈ **Ω**(g(n)), if there are positive constants c and n_0 such that $f(n) \geq c *$ g(n) for all $n \geq n_0$.

- Obviously $f_1 \in \Omega(f_2)$ because $f_1 > f_2$ (after about n=10, so we set n_0 = 10)
  - f_2 is clearly a ***lower bound*** on f_1 and that's what big-Ω is all about
- But $f_2 \in \Omega(f_1)$ as well!
  - We just have to use the "**c**" to adjust so f_1 that it moves below f_2
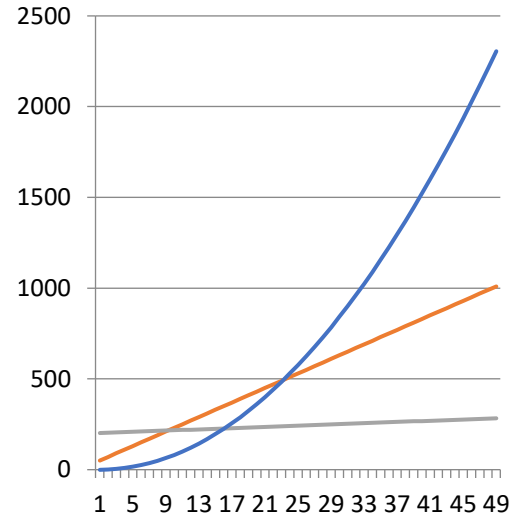

f_1

f_2

c * f_1

f(n) ∈ **O**(g(n)), if there are positive constants c and n_0 such that f(n) ≤ **c** * g(n) for all n ≥ n_0.

f(n) ∈ **Ω**(g(n)), if there are positive constants c and n_0 such that f(n) ≥ **c** * g(n) for all n ≥ n_0.

$$f_3 \in \Omega(f_1)$$

A. TRUE

B. FALSE

Why or why not?

f_3

f_1

f_2

$$f_3 > \boxed{c} f_1$$

$$f_2 \in \Omega(f_1)$$

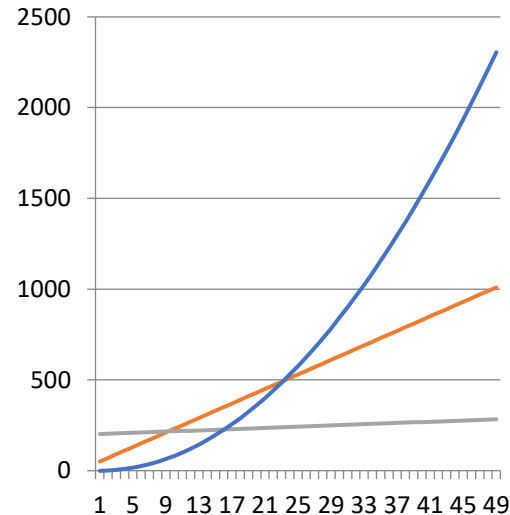f(n) ∈ **O**(g(n)), if there are positive constants c and n_0 such that f(n) ≤ **c** * g(n) for all n ≥ n_0.

f(n) ∈ **Ω**(g(n)), if there are positive constants c and n_0 such that f(n) ≥ **c** * g(n) for all n ≥ n_0.

$$f_1 \in \Omega(f_3)$$

A. TRUE

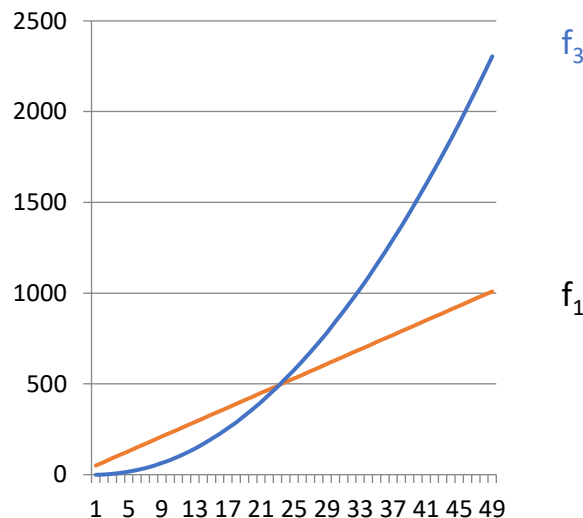B. FALSE

Why or why not?

$$\frac{1}{2}n^2 \in \Omega(n^2)$$

$$\frac{1}{2}n^2 + 10n \in \Omega(n^2)$$

A : T

B : F

2500
2000
1500
1000
500
0

1 5 9 13 17 21 25 29 33 37 41 45 49

f_3

f_1

f_2

$f_3 \in \Omega(f_1)$ *but* f_1 not $\in \Omega(f_3)$

- There is no way to pick a c that would make an $O(n^2)$ function (f_3) stay below an $O(n)$ function (f_1).

Summary

Big-O

- **Upper bound** on a function
- $f(n) \in O(g(n))$ means that we can expect f(n) will always be **under** the bound g(n)
  - But we don't count n up to some starting point n_0
  - And we can "cheat" a little bit by moving g(n) up by multiplying by some constant c

Big-Ω

- **Lower bound** on a function
- $f(n) \in \Omega(g(n))$ means that we can expect f(n) will always be **over** the bound g(n)
  - But we don't count n up to some starting point n_0
  - And we can "cheat" a little bit by moving g(n) down by multiplying by some constant c

Big-θ $(==)$

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

$\exists \; n_0, c_1, c_2$

s.t.

- **Tight bound** on a function.
- If $f(n) \in O(g(n))$ *and* $f(n) \in \Omega(g(n))$, then $f(n) \in \theta(g(n))$.
- Basically it means that $f(n)$ and $g(n)$ are interchangeable
- Examples:
  - $3n+20 = \theta(10n+7)$

  - $5n^2 + 50n + 3 = \theta(5n^2 + 100)$

$O(n! \; n^n)$

$c_1 g(n) \leq f(n) \leq c_2 g(n)$

when $n \geq n_0$

$f(n) = 3n+20$

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

$f(n) \in \Theta(n^2) \; \times$
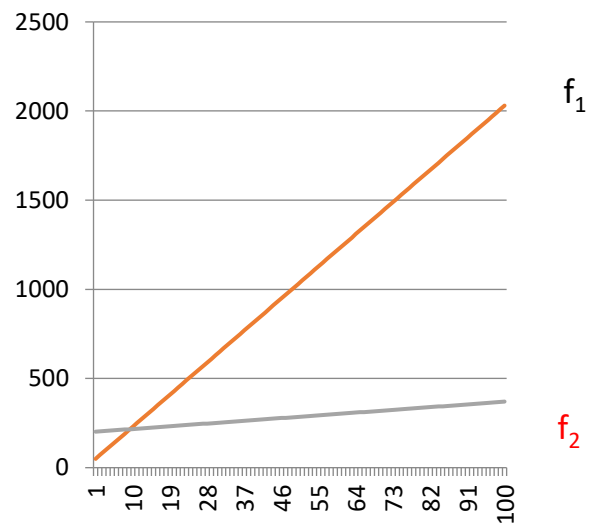
$f(n) \in O(n) \checkmark$

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

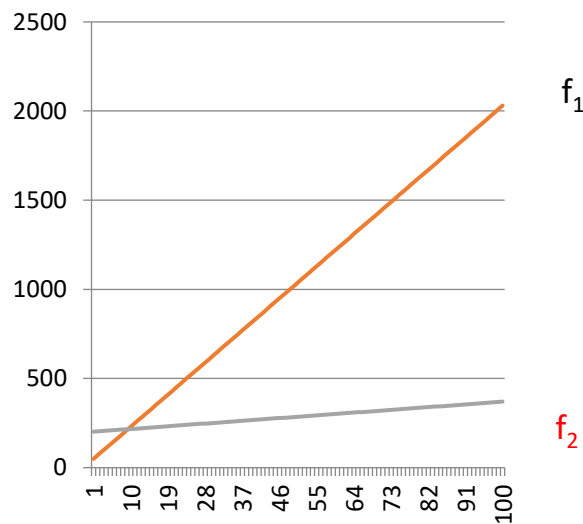$f_1 \in \Theta(f_2)$

A. TRUE

B. FALSE

Why or why not?

$f_1 \in \Theta(f_2)$

A. TRUE

B. FALSE

Why or why not?

Since f_1 is $O(f_2)$ and $\Omega(f_2)$, it is also $\Theta(f_2)$ (this is the definition of big-Theta)
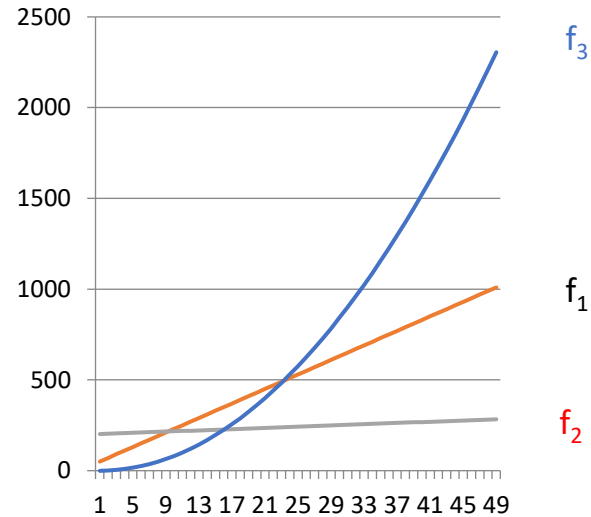
$f_1 \in \Theta(f_3)$

A. TRUE

B. FALSE

Why or why not?

$f_1 \in \Omega(f_3)$ ✗

$f_1 \in O(f_3)$ ✓

Big-θ and sloppy usage

- Sometimes people say, "This algorithm is $O(n^2)$" when it would be more precise to say that it is $\theta(n^2)$
  - They are intending to give a tight bound, but use the looser "big-O" term instead of the "big-θ" term that actually means tight bound
  - Not wrong, but not as precise
- I don't know why, this is just a cultural thing you will encounter among computer scientists

Count how many times each line executes, then say which
O( ) **statement(s) is(are) true**.

Line #

```
int maxDifference(int[] arr){          1
    max = 0;                            2
    for (int i=0; i<arr.length;  i++) {  3
        for (int j=0; j<arr.length; j++) { .  4
            if (arr[i] - arr[j] > max)    5
                max = arr[i] - arr[j];    6
        }                                7
    }                                    8
    return max;                          8
}                                        9
```

tightest :$O(n^2)$

any : $\in$
(ABD)

A. $f(n) \in O(2^n)$ D. $f(n) \in O(n^3)$

B. $f(n) \in O(n^2)$ E. Other/none/more

C. $f(n) \in O(n)$ *(assume n = arr.length)*

Count how many times each line executes, then say which
Θ( ) statement(s) is(are) true.

Line #

```
int maxDifference(int[] arr){          1

    max = 0;                           2

    for (int i=0; i<arr.length;  i++) {   3

        for (int j=0; j<arr.length; j++) {   4

            if (arr[i] - arr[j] > max)    5

                max = arr[i] - arr[j];    6

        }                              7

    }                                  8

    return max;                        9

}
```

A. $f(n) \in \theta(2^n)$ D. $f(n) \in \theta(n^3)$

B. $f(n) \in \theta(n^2)$ E. Other/none/more

C. $f(n) \in \theta(n)$ *(assume n = arr.length)*

Count how many times each line executes, then say which
Θ( ) statement(s) is(are) true.

```
int sumTheMiddle(int[] arr){
    int range = 100;
    int start = arr.length/2 - range/2;
    int sum = 0;
    for (int i=start; i<start+range;  i++)
    {
        sum += arr[i];
    }
    return max;
}
```

A. $f(n) \in \theta(2^n)$ D. $f(n) \in \theta(1)$

B. $f(n) \in \theta(n^2)$ E.   None of these

C. $f(n) \in \theta(n)$ *(assume n = arr.length)*

Count how many times each line executes, then say which
Θ( ) statement(s) is(are) true.

```
int sumTheMiddle(int[] arr){
    int range = arr.length/100;
    int start = arr.length/2 - range/2;
    int sum = 0;
    for (int i=start; start+range;  i++)
    {
        sum += arr[i];
    }
    return sum;
}
```

A. f(n) ∈ θ(2ⁿ)   D. f(n) ∈ θ(1)

B. f(n) ∈ θ(n²)   E.   None of these

C. f(n) ∈ θ(n)   *(assume n = arr.length)*

With *worst case* analysis, which algorithm has the better (smaller) Big-Θ bound?

```
boolean find( String[] theList, String toFind ) {
   for ( int i = 0; i < theList.length; i++ ) {
      if ( theList[i].equals(toFind) )
         return true;
   }
   return false;
}

boolean fastFind( String[] theList, String toFind ) {
   return false;
}
```

A. find
B. fastFind
C. They are the same

With *best case* analysis, which algorithm has the better Big-Θ bound?

```
boolean find( String[] theList, String toFind ) {
  for ( int i = 0; i < theList.length; i++ ) {
    if ( theList[i].equals(toFind) )
      return true;
  }
  return false;
}

boolean fastFind( String[] theList, String toFind ) {
  return false;
}
```

A. find
B. fastFind
C. They are the same

# Practical sorting algorithms: Selection sort

Pseudocode: selectionSort
- *While the size of the unsorted part is greater than 1*
  - *find the position of the smallest element in the unsorted part*
  - *move this smallest element to the last position in the sorted part*
  - *increase the size of the sorted part and decrement the size of the unsorted part*

https://www.youtube.com/watch?v=Ns4TPTC8whw

| 12 | 4 | 9 | 3 | 15 | 8 | 19 | 2 |
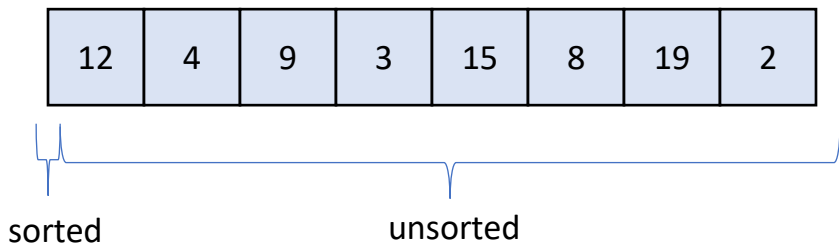
sorted   unsorted

Insertion Sort: The Picture

(a) Initial configuration for insertion sort. The input array is logically split into a sorted part (initially containing one element) and an unsorted part.
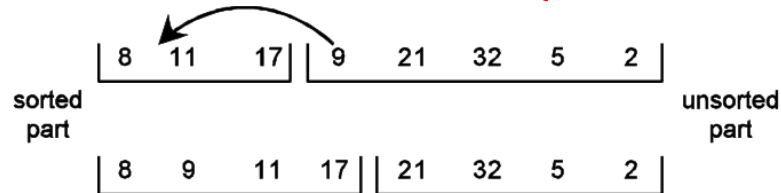
sorted part | 17 | 8   11   9   21   32   5   2 | unsorted part
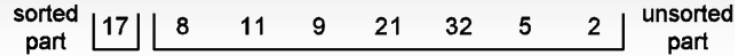
(b) The array after the first value from the unsorted part has been inserted in its proper position in the sorted part (first pass).

| 17 | 8   11   9   21   32   5   2 |
sorted part   unsorted part

| 8   17 | 11   9   21   32   5   2 |
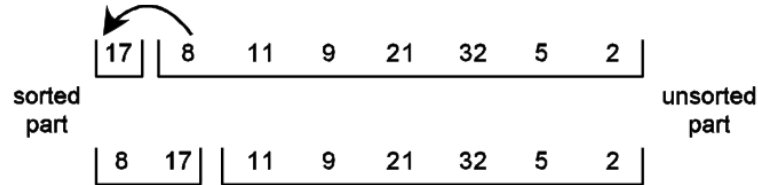
(c) The array after the first value from the unsorted part has been inserted in its proper position in the sorted part (second pass).

| 8   17 | 11   9   21   32   5   2 |
sorted part   unsorted part

| 8   11   17 | 9   21   32   5   2 |
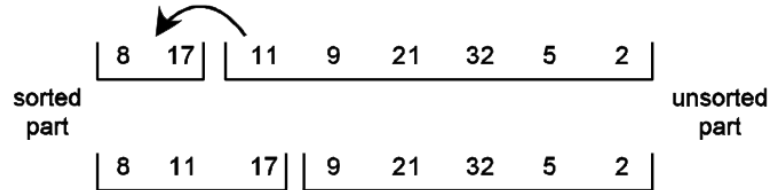
(d) The array after the first value from the unsorted part has been inserted in its proper position in the sorted part (third pass).

| 8   11   17 | 9   21   32   5   2 |
sorted part   unsorted part

| 8   9   11   17 | 21   32   5   2 |

https://www.youtube.com/watch?v=ROalU379l3U

Selection Sort – what does it print out?

$$O(n^2) \qquad \Theta(n^2)$$

```java
import java.util.Arrays;
public class Sort {
public static void sortA(int[] arr) {
  for(int i = 0; i < arr.length; i += 1) {
    System.out.print(Arrays.toString(arr) + " -> ");
    int minIndex = i;
    for(int j = i; j < arr.length; j += 1) {
      if(arr[minIndex] > arr[j]) { minIndex = j; }
    }
    int temp = arr[i];
    arr[i] = arr[minIndex];
    arr[minIndex] = temp;
    System.out.println(Arrays.toString(arr));
  }
}
}
```

n

Sort.sortA(new int[]{ 53, 83, 15, 45, 49 });

[53, 83, 15, 45, 49] ->

$$\sum_{i=0}^{n-1} \left(\sum_{j=i}^{n-1} 1 \right) + C$$

$$\frac{n(n+1)}{2}$$

$1 + 2 + 3 + \cdots + n$

$$\sum_{i=0}^{n-1} \left(n - i + C \right)$$

$$n - 0 + n - 1 + \cdots + n - (n-1)$$

Insertion Sort – what does it print out?

```
import java.util.Arrays;
public class Sort {
public static void sortB(int[] arr) {
 for(int i = 0; i < arr.length; i += 1) {
   System.out.print(Arrays.toString(arr) + " -> ");
   for(int j = i; j > 0; j -= 1) {
     if(arr[j] < arr[j-1]) {
       int temp = arr[j-1];
       arr[j-1] = arr[j];
       arr[j] = temp;
     }
   }
   System.out.println(Arrays.toString(arr));
 }
}
}
```

Sort.sortB(new int[]{ 53, 83, 15, 45, 49 });

[53, 83, 15, 45, 49] ->

$$\sum_{i=0}^{n-1} \left(\sum_{j=i}^{1} 1 \right) \in O(n^2)$$