Efficiency continued

Clicker

```
count = 0
while (count < n)
    statement1
    statement2
    statement3
    statement4
end outer while
```

What is the efficiency of this algorithm in BigO?

A) O(1)
B) O(n)
C) $O(\log_2 n)$
D) $O(n^2)$
E) $O(2^n)$

Pseudocode for a search algorithm…

```
search(list, a)
    N = number of elements in list
    count = 0

    while count < N do the following
        if (list[count] == a)
            return count

        count++

    return -1  // not found

end search function
```

What value does this algorithm return if we are searching for the number 10 in this list?

0	1	2	3	4	5	6
8	18	16	14	11	10	22

A) 10
B) 6
C) 5
D) -1
E) None of the above

Pseudocode for a search algorithm…

```
search(list, a)
    N = number of elements in list
    count = 0

    while count < N do the following
        if (list[count] == a)
            return count

        count++

    return -1  // not found

end search function
```

What value does this algorithm return if we are searching for the number 5 in this list?

0	1	2	3	4	5	6
8	18	16	14	11	10	22

A) 7
B) 6
C) 0
D) -1
E) None of the above

Clicker

```
search(list, a)
    N = number of elements in list
    count = 0

    while count < N do the following
        if (list[count] == a)
            return count

        count++

    return -1  // not found

end search function
```

What is the efficiency of this algorithm in BigO?

A) O(1)
B) O(n)
C) $O(\log_2 n)$
D) $O(n^2)$
E) $O(2^n)$

Pseudocode for a search algorithm…

```
search(list, a)
    N = number of elements in list
    count = 0

    while count < N do the following
        if (list[count] == a)
            return count

        count++

    return -1  // not found

end search function
```

How many times does the while loop repeat if we are searching for the number 22 in this list?

0	1	2	3	4	5	6
8	10	11	14	16	18	22

A) 5
B) 4
C) 0
D) -1
E) None of the above

Challenge…

search(list, a)

Can you come up with a more efficient algorithm to search for a number in a list like the one shown here:

0	1	2	3	4	5	6
8	10	11	14	16	18	22

Challenge…

```
binarySearch(list, a)

    N = number of elements in list
    minIndex =0
    maxIndex = N-1
    while (?                          )
        middle = (minIndex + maxIndex)/2
        currentItem = list[middle]









    return -1 // not found
```

Can you come up with a more efficient algorithm to search for a number in a list like the one shown here:

0	1	2	3	4	5	6
8	10	11	14	16	18	22

Challenge…

```
binarySearch(list, a)

    N = number of elements in list
    minIndex =0
    maxIndex = N-1
    while (?)
        middle = (minIndex + maxIndex)/2
        currentItem = list[middle]
        if (currentItem == a)
            return middle
        else if (currentItem > a)
            maxIndex = middle-1
        else
            minIndex = middle+1

    return -1 // not found
```

Can you come up with a more efficient algorithm to search for a number in a list like the one shown here:

0	1	2	3	4	5	6
8	10	11	14	16	18	22

Challenge…

```
binarySearch(list, a)

    N = number of elements in list
    minIndex =0
    maxIndex = N-1
    while (minIndex <= maxIndex)
        middle = (minIndex + maxIndex)/2
        currentItem = list[middle]
        if (currentItem == a)
            return middle
        else if (currentItem > a)
            maxIndex = middle-1
        else
            minIndex = middle+1

    return -1 // not found
```

Can you come up with a more efficient algorithm to search for a number in a list like the one shown here:

0	1	2	3	4	5	6
8	10	11	14	16	18	22

Challenge…

```
binarySearch(list, a)

    N = number of elements in list
    minIndex =0
    maxIndex = N-1
    while (minIndex <= maxIndex)
        middle = (minIndex + maxIndex)/2
        currentItem = list[middle]
        if (currentItem == a)
            return middle
        else if (currentItem > a)
            maxIndex = middle-1
        else
            minIndex = middle+1

    return -1 // not found
```

What is the efficiency of this algorithm in BigO?

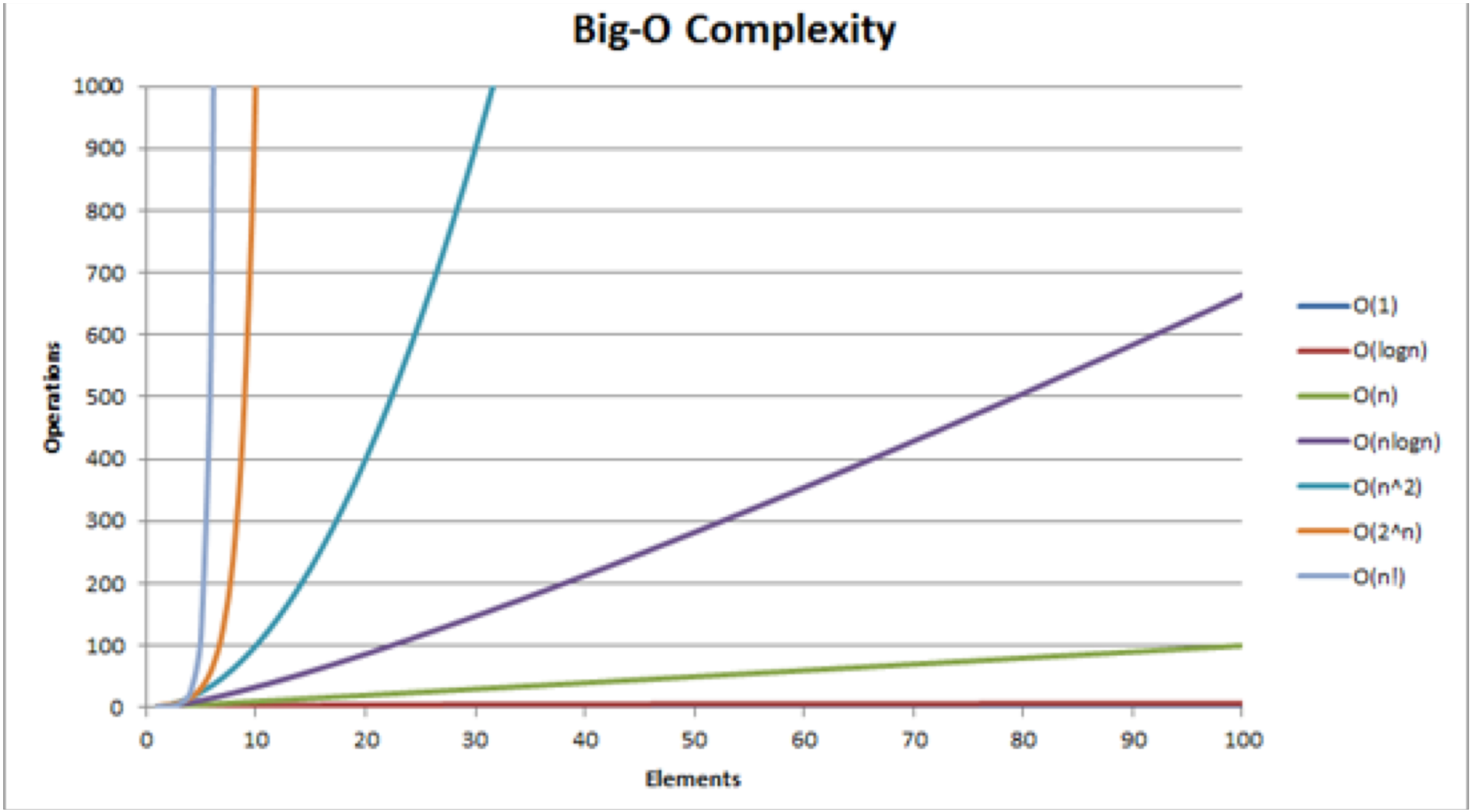
A) O(1)

B) O(n)

C) $O(\log_2 n)$

D) $O(n^2)$

E) $O(2^n)$



http://www.daveperrett.com/articles/2010/12/07/comp-sci-101-big-o-notation/

clicker

- Which algorithm would you use if you were asked to write a search function for a given list

A. Binary Search
B. Linear Search
C. It depends on if the list is sorted or not

Discuss…

- If you were given an unsorted list of numbers, describe an algorithm to put them in sorted order.

Big O?

```
count = 0
while (count < n)
   repeat = 0
   while (repeat < count)
        statement1
        statement2
        statement3
        statement4
   end inner while
end outer while
```

A) O(1)
B) O(n)
C) $O(\log_2 n)$
D) $O(n^2)$
E) $O(2^n)$

Big O?

```
function1(n)
    count = 0
    while (count < n)
      call function2(count)
    end of while
end of function1

function2(n)
    count = 0
    while (count < n)
      statement1
    end of while
end of function2
```

A) O(1)
B) O(n)
C) $O(\log_2 n)$
D) $O(n^2)$
E) $O(2^n)$

Efficiency – Big O?

```
indexOfSmallest(data, size, start)
    smallest = start
    for i = start+1 to i=size (not inclusive)
        if(data[i] < data[smallest])
            smallest = i
return smallest
```

A) O(1)
B) O(n)
C) $O(\log_2 n)$
D) $O(n^2)$
E) $O(2^n)$

Sorting…

```
selectionSort(data, size)
    for i = 0 to i=size-1 (not inclusive)
        indexSmallest = indexOfSmallest(data, size, i)
        swap(data[i], data[indexSmallest])
```

Efficiency – Big O?

A) O(1)
B) O(n)
C) $O(\log_2 n)$
D) $O(n^2)$
E) $O(2^n)$

```
selectionSort(data, size)
    for i = 0 to i=size-1 (not inclusive)
        indexSmallest = indexOfSmallest(data, size, i)
        swap(data[i], data[indexSmallest])
swap(element1, element2)
    temp = element1,
    element1 = element2
    element2 = temp
```