https://goo.gl/oM4jRM
ITI1120 Review Session

[Study Sheet]

Big Oh Notation

Author: Faizaan Chishtie

Today’s Session:

- What is Big Oh?
- Examples
- Test Taking Strategies
- Practice

Definition

Worst case runtime of an algorithm

- Primitive comparisons/swaps/arithmetic et al. O(1)
- Loops/Recursion are O(something) depending on dependence on input
-

We express complexity using **big-O notation**.

For a problem of size N:

- a constant-time algorithm is "order 1": O(1)
- a linear-time algorithm is "order N": O(N)
- a quadratic-time algorithm is "order N squared": $O(N^2)$

Note that the big-O expressions do not have constants or low-order terms. This is because, when N gets large enough, constants and low-order terms don't matter (a constant-time algorithm will be faster than a linear-time algorithm, which will be faster than a quadratic-time algorithm).

What is Big Oh?

1

log(n)

n

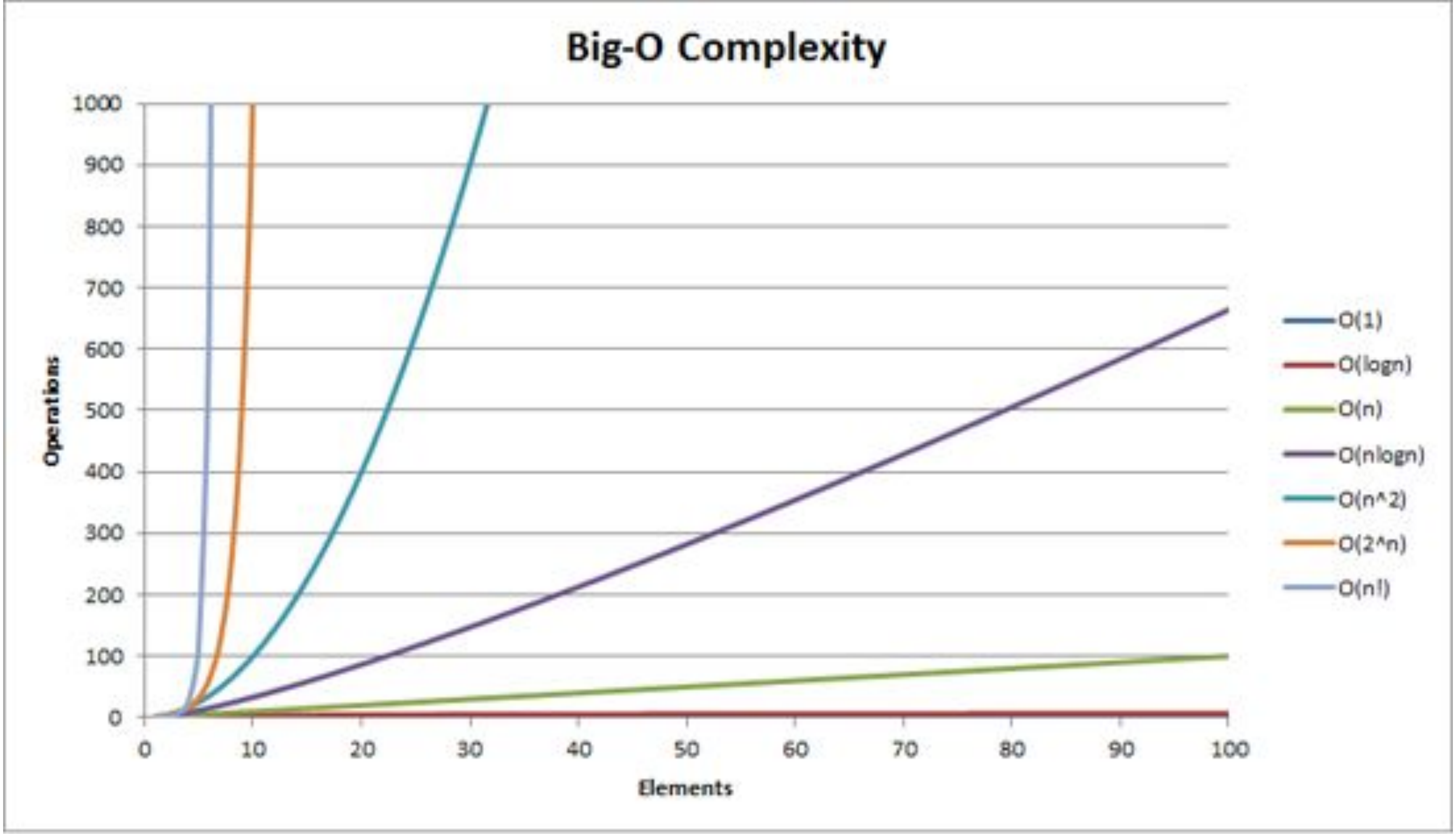
O

nlogn

n^2

2^n

n!


Big-O Complexity
Operations
Elements
1000
900
800
700
600
500
400
300
200
100
0
0
10
20
30
40
50
60
70
80
90
100
O(1)
O(logn)
O(n)
O(nlogn)
O(n^2)
O(2^n)
O(n!)

O(1)

```
def func(n):
    if(n == 1):
        return n - 1
    return n+2
def func(n):
    for i in range(3): #same range no matter the input
    return
```

O(log(n))

```
def binary_search(alist, start, end, key):
    """Search key in alist[start... end - 1]."""
    if not start < end:
        return -1

    mid = (start + end)//2
    if alist[mid] < key:
        return binary_search(alist, mid + 1, end, key)
    elif alist[mid] > key:
        return binary_search(alist, start, mid, key)
    else:
        return mid
```

O(n)

```
def func(n):
    s = 0
    for i in range(n): #in range n!
        s += i
    return s
```

O(n^2)

```
def func(n):
    s = 0
    for i in range(n):
        for j in range(n): #two range functions dependant on n
            s += j
    return s
```

Be **careful!**

Look for range loops that are dependant on n.

Constant range loops, even for i in range(1000000), are O(1)

O(2^n)

Fibonacci sequence:

```
def fibonacci(n):
    if(n <= 1):
        return n
    else:
        return(fibonacci(n-1) + fibonacci(n-2))
```

number of recursive calls is 2 for each iteration of n.

Test Taking Strategies

Recognize

Plan

Do

Contact

fchis052@uottawa.ca