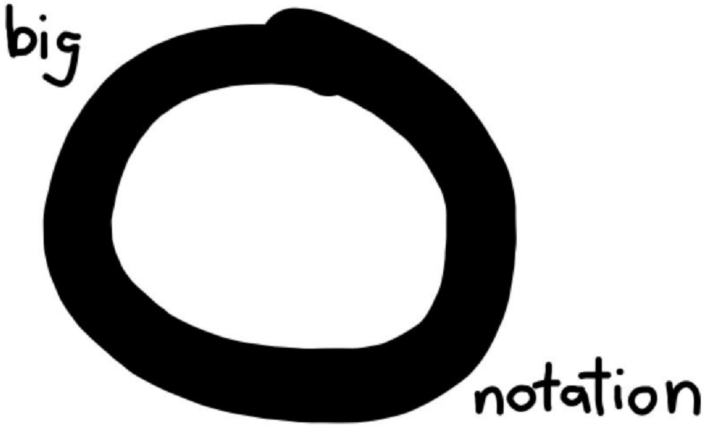


by habib

Outline

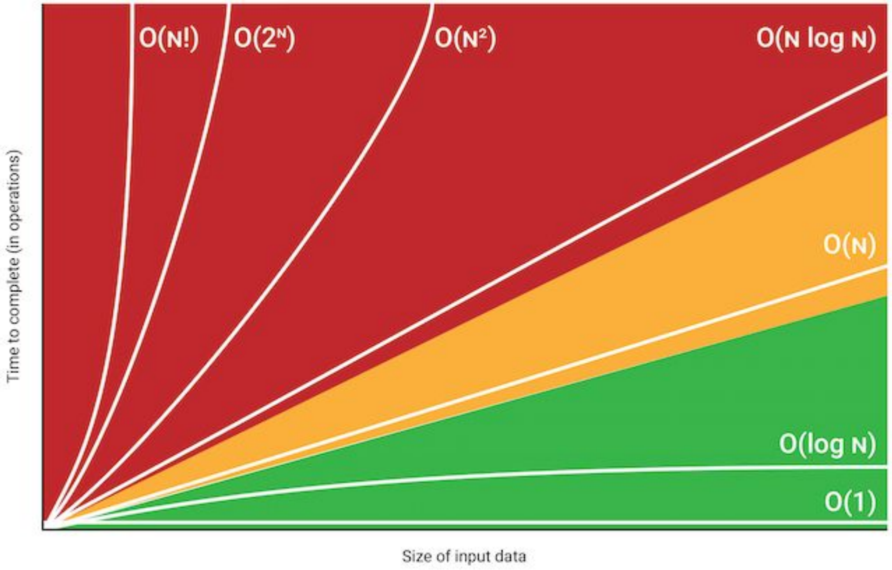
- Why we care
- What it be
- How we calculate it
- Examples
- Future readings

Why we care

- CLRS: “Using O-notation, we can often describe the running time of an algorithm merely by inspecting the algorithm’s overall structure”
- Cracking the Coding interview: “Big O time is the language and metric we use to describe the efficiency of algorithms. Not understanding it thoroughly can really hurt you in developing an algorithm”
- Not everyone’s computer is the same!
    - If program runs well on your good computer, might not work as well on someone’s much slower computer
- Basically tells us how our algorithms run, and how they will scale with different conditions

What it be

- Big O, big omega, big theta
  - time complexity
  - space complexity
- Represented by O(s) where s = some equation describing # of operations
- n usually refers to input size
- O(n) can be faster than O(1) but on average it's slower!



How we calculate it (ground rules)

- Best, worst, and expected case
  - really just care about the worst and expected case
  - worst: choose biggest pivot repeatedly in quick sort
- Drop non-dominant terms
  - O(2n) = O(n)
  - O(n^3 + n^2 + n^1) = O(n^3)
- Each line is one operation
  - Add if operation A then B
  - Multiply if operation A executed B times

```python
## add case
arr_a = [1, 2, 3, 4, 5]
arr_b = [5, 4, 3, 2, 1]
count = 0
for elem_a in arr_a:
    count += 1
    print(elem_a)

for elem_b in arr_b:
    count+= 1
    print(elem_b)
print(count)
```

```python
## multiply case
arr_a = [1, 2, 3, 4, 5]
arr_b = [5, 4, 3, 2, 1]
count = 0
for elem_a in arr_a:
    for elem_b in arr_b:
        print(elem_a, ",", elem_b)
        count += 1
print(count)
```

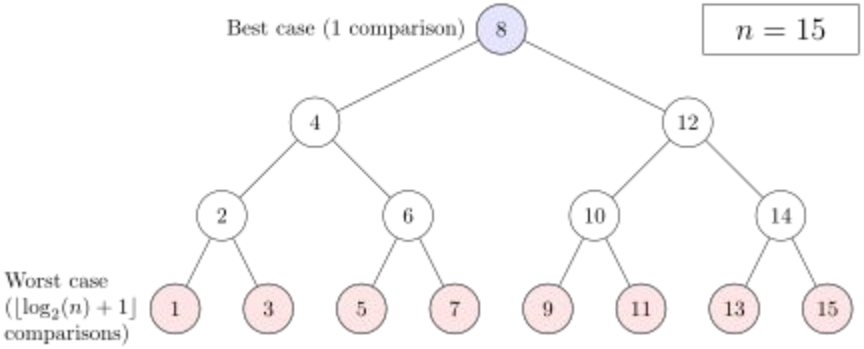
How we calculate it (space)

- Same idea as runtime
- Talk about this when we deal with recursion OR dealing with extra space (maps, stacks, arrays, etc...)
- One recursive call -> one level to system stack

```python
def multiplication(multiplicand, multiplier):
    if multiplier <= 0:
        return 0
    return multiplicand + multiplication(multiplicand, multiplier - 1)
print(multiplication(2,3))
```

How we calculate it (log runtimes)

- Dividing input size by half each step leads to log(n) run time
- Examples:
    - Binary search has O(log(n)) runtime
    - Merge sort has O(n * log(n)) runtime



Examples (what are these run times?)

big_o.py > ...

```python
my_arr = [5, 4, 1, 2, 7]

target = 5
two_nums = []

for i in range(len(my_arr)-1):
    for j in range(i+1, len(my_arr)):
        if my_arr[i] + my_arr[j] == target:
            two_nums.append(my_arr[i])
            two_nums.append(my_arr[j])

print(two_nums)
```

```python

two_nums = []
check = set()
for i in range(len(my_arr)):
    if my_arr[i] in check:
        two_nums.append(my_arr[i])
        two_nums.append(target-my_arr[i])
    else:
        check.add(target-my_arr[i])
print(two_nums)

```

Examples (what are these run times?) cont.

```python
## printing numbers with a twist
arr_a = [1, 2, 3, 4, 5, 6, 7]
arr_store = []

for i in range(len(arr_a)):
    for j in range(len(arr_a) // 2):
        arr_store.append(arr_a[j] + arr_a[i])
        print(arr_a[i], arr_a[j] + arr_a[i])

print(arr_store)
```

```python
def fib(number):
    if number <= 1:
        return 1
    return fib(number - 1) + fib(number - 2)

print(fib(5))
```

Examples

- Which one of these has a run time of O(N)?
    - O(N + M)
    - O(2^N)
    - O(N + N/2)
    - O(N + log(N))
    - O(2N + 3)

- What’s the run time of the factorial function?

```python
def factorial(number):
    if n < 0:
        return -1
    elif n == 0:
        return 1
    else:
        return factorial(n - 1) * n

print(factorial(4)) # 4! = 4 * 3 * 2 * 1 = 24
```

Examples (gimme gimme)

```python
odd_arr= [1, 3, 5, 7, 9, 11, 13]

for element in odd_arr:
    for i in range(5):
        print(element * i)
    print("-----")
```

Future readings

- CLRS - Big algorithm textbook from the late 80’s
- Cracking the coding interview - helpful review of data structures and algorithms
- [Abdul Bari algorithms series]