



big



notation

by habib

# Outline

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





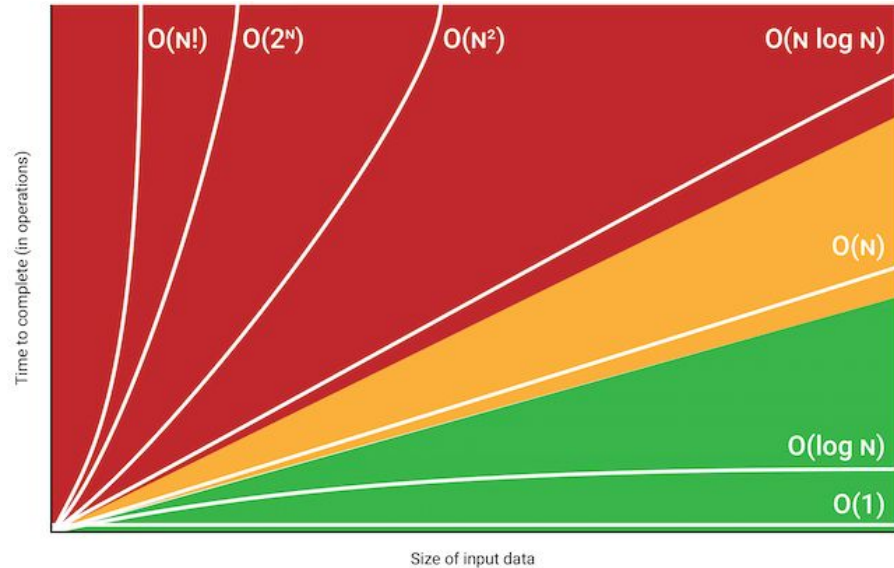
# Why we care



- 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





# 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

```
3  ## add case
4  arr_a = [1, 2, 3, 4, 5]
5  arr_b = [5, 4, 3, 2, 1]
6  count = 0
7  for elem_a in arr_a:
8      count += 1
9      print(elem_a)
10
11 for elem_b in arr_b:
12     count += 1
13     print(elem_b)
14 print(count)
15
```

```
10
17 ## multiply case
18 arr_a = [1, 2, 3, 4, 5]
19 arr_b = [5, 4, 3, 2, 1]
20 count = 0
21 for elem_a in arr_a:
22     for elem_b in arr_b:
23         print(elem_a, ",", elem_b)
24         count += 1
25 print(count)
26
```



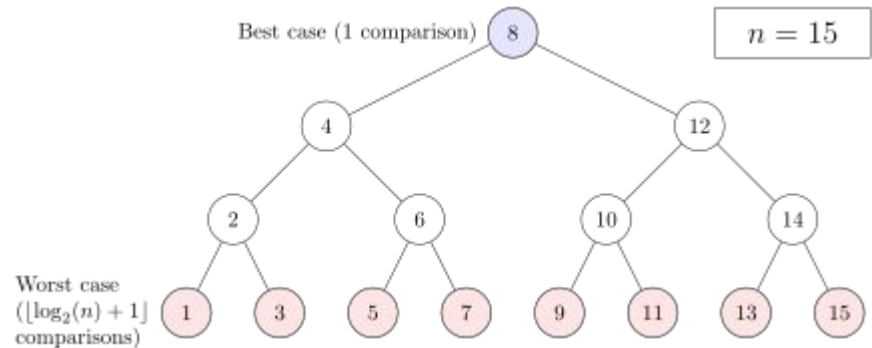
## How we calculate it (space)

- 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

```
55 def multiplication(multiplicand, multiplier):  
56     if multiplier <= 0:  
57         return 0  
58     return multiplicand + multiplication(multiplicand, multiplier - 1)  
59 print(multiplication(2,3))  
60
```

# 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 > ...
1  my_arr = [5, 4, 1, 2, 7]
2
3  target = 5
4  two_nums = []
5
6  for i in range(len(my_arr)-1):
7      for j in range(i+1, len(my_arr)):
8          if my_arr[i] + my_arr[j] == target:
9              two_nums.append(my_arr[i])
10             two_nums.append(my_arr[j])
11
12  print(two_nums)
13
```

```
38
39  two_nums = []
40  check = set()
41  for i in range(len(my_arr)):
42      if my_arr[i] in check:
43          two_nums.append(my_arr[i])
44          two_nums.append(target-my_arr[i])
45      else:
46          check.add(target-my_arr[i])
47  print(two_nums)
48
49  # print(two_nums)
```





## Examples (what are these run times?) cont.

```
61  ## printing numbers with a twist
62  arr_a = [1, 2, 3, 4, 5, 6, 7]
63  arr_store = []
64
65  for i in range(len(arr_a)):
66      for j in range(len(arr_a) // 2):
67          arr_store.append(arr_a[j] + arr_a[i])
68          print(arr_a[i], arr_a[j] + arr_a[i])
69
70  print(arr_store)
```

```
69  def fib(number):
70      if number <= 1:
71          return 1
72      return fib(number - 1) + fib(number - 2)
73
74  print(fib(5))
75
```



# 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?

```
76
77 def factorial(number):
78     if n < 0:
79         return -1
80     elif n == 0:
81         return 1
82     else:
83         return factorial(n - 1) * n
84
85 print(factorial(4)) # 4! = 4 * 3 * 2 * 1 = 24
86
```



## Examples (gimme gimme)

```
88 odd_arr= [1, 3, 5, 7, 9, 11, 13]
89
90 for element in odd_arr:
91     for i in range(5):
92         print(element * i)
93     print("-----")
94
```



## 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](#)