

Worksheet: Complexity Analysis

1

```
def search(L, target):
    for x in L:
        if x == target: #line A
            return True
    return False
```

assume this is asked unless otherwise stated

Let $f(N)$ be the number of times line A executes, with $N = \text{len}(L)$. What is $f(N)$ in each case?

- Worst Case (target is at end of list): $f(N) = \underline{\hspace{2cm}}$
- Best Case (target is at beginning of list): $f(N) = \underline{\hspace{2cm}}$
- Average Case (target in middle of list): $f(N) = \underline{\hspace{2cm}}$

A **step** is any unit of work with bounded execution time (it doesn't keep getting slower with growing input size).

We classify algorithm complexity by classifying the **order of growth** of a function $f(N)$, where f gives the number of steps the algorithm must perform for a given input size.

Big O definition: if $f(N) \leq C * g(N)$ for large N values and some fixed constant C , then $f(N) \in O(g(N))$

2

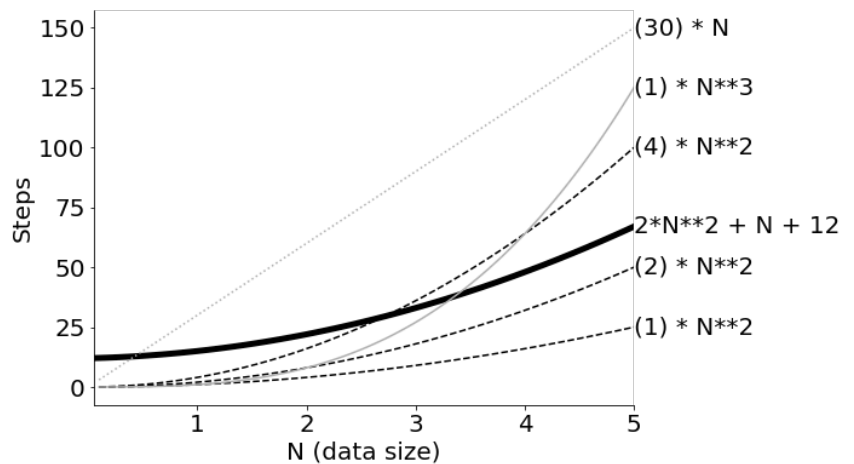
Let $f(N) = 2N^2 + N + 12$

If we want to show $f(N) \in O(N^3)$, what is a good lower bound on N ? Let's have $C=1$.

To show $f(N) \in O(N^2)$, do we pick 1, 2, or 4 for the C ? After picking C , what should we choose for N 's lower bound?

What is more informative to show?
 $f(N) \in O(N^3)$ or $f(N) \in O(N^2)$?

Somebody claims $f(N) \in O(N)$, offering $C=30$ and $N > 0$. Suggest an N value to counter their claim.



3

```
nums = [...]
first100sum = 0
for x in nums[:100]:
    first100sum += x
print(first100sum)
```

If we increase the size of `nums` from 20 items to 100 items, the code will probably take _____ times longer to run.

If we increase the size of `nums` from 100 to 1000, will the code take longer? Yes / No

The complexity of the code is $O(\underline{\hspace{2cm}})$, with $N = \text{len}(\text{nums})$.

4

Each of the following list operations are either $O(1)$ or $O(N)$, where N is $\text{len}(L)$. Circle those you think are $O(N)$.

- `L.insert(0, x)`
- `L.pop(0)`
- `x = L[0]`
- `x = max(L)`
- `x = len(L)`
- `L.append(x)`
- `L.pop(-1)`
- `L2.extend(L)`
- `x = sum(L)`
- `found = x in L`

5

```
L = [...]
for x in L:
    avg = sum(L) / len(L)
    if x > 2*avg:
        print("outlier", x)
```

What is the big O complexity?

Is there a way to optimize the code?

6

```
A = [...]
B = [...]
```

```
for x in A:
    for y in B:
        print(x*y)
```

how would you define the variable(s) to describe the size of the input data?

The complexity of code is $O(\text{_____})$

7

assume L is already sorted, $N=\text{len}(L)$

```
def binary_search(L, target):
    left_idx = 0 # inclusive
    right_idx = len(L) # exclusive
    while right_idx - left_idx > 1:
        mid_idx = (right_idx + left_idx) // 2
        mid = L[mid_idx]
        if target >= mid:
            left_idx = mid_idx
        else:
            right_idx = mid_idx

    return right_idx > left_idx and L[left_idx] == target
```

how many times does this step run when $N = 1$? $N = 2$? $N = 4$? $N = 8$?

If $f(N)$ is the number of times this step runs, then $f(N) = \text{_____}$

The complexity of binary search is $O(\text{_____})$

8

```
s1 = tuple("...") # could be any string
s2 = tuple("...")
```

```
# version A
import itertools
```

```
matches = False
for p in itertools.permutations(s1):
    if p == s2:
        matches = True
```

```
# version B
s1 = sorted(s1)
s2 = sorted(s2)
matches = (s1 == s2)
```

assumed sorted is $O(N \log N)$

what is the complexity of version A? $O(\text{_____})$

what is the complexity of version B? $O(\text{_____})$

9

```
def selection_sort(L):
    for i in range(len(L)):
        idx_min = i
        for j in range(i, len(L)):
            if L[j] < L[idx_min]:
                idx_min = j
        # swap values at i and idx_min
        L[idx_min], L[i] = L[i], L[idx_min]
```

if this runs $f(N)$ times, where $N=\text{len}(L)$, then $f(N) = \text{_____}$

The complexity of selection sort is $O(\text{_____})$

```
nums = [2, 4, 3, 1]
selection_sort(nums)
print(nums)
```