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 $f(N) \in O(g(N))$

1

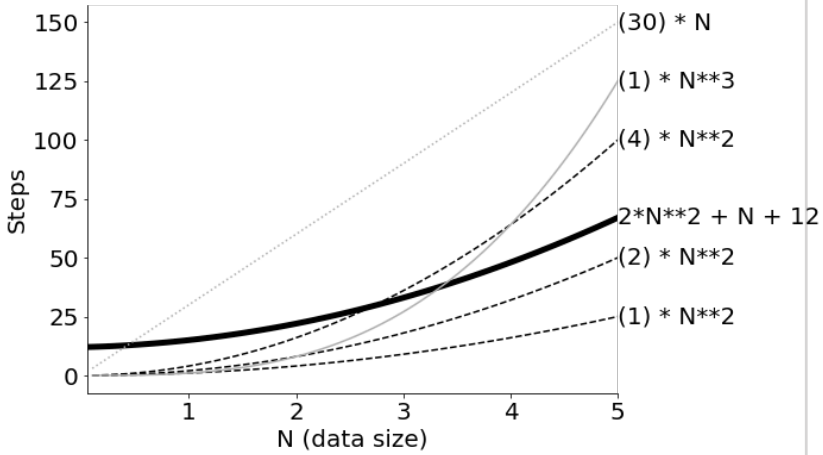
Let $f(N) = 2N^2 + N + 12$

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

If we want to show $f(N) \in O(N^2)$, do you pick 1, 2, or 4 for the C? After you pick C, what you choose for a lower bound on N?

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.



2

Each of the following list operations are either O(1) or O(N), where N is list length. 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.append(x)    L2.extend(L)   x = sum(L)    found = X in L
```

3

```
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=len(L)`. What is **f(N)** in each case?

Worst Case (target is at end of list): f(N) = _____

Best Case (target is at beginning of list): f(N) = _____

Average Case (target in middle of list): f(N) = _____

4

```
# assume L is already sorted, N=len(L)
def 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
```

5

```
# assume L is already sorted, N=len(L)
def 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 steps runs, then f(N) = _____

The complexity of binary search is O(_____)