AddisCoders 2026 Week 2 Day 10 Lab B

Lecturer: Huy Nguyen August 7, 2026

1. Counting Operations

To find the Big-O runtime of a program, we take two steps:

(1) Count the number of operations, in terms of n.

(2) Remove all multiplicative constants, and keep only the highest-order term.

Here are some examples of step (2):

$5n + 3 \longrightarrow O(n)$	drop the +3, then drop the 5
$100n + 2 \longrightarrow O(n)$	a big constant is still just a constant
$2n^2 + 50n + 7 \longrightarrow O(n^2)$	n^2 is the highest-order term
$\frac{1}{2}n^2 \longrightarrow O(n^2)$	the constant can be a fraction
$8 \longrightarrow O(1)$	no n at all

a. Try these two yourself:

$6n^2 + 300n \rightarrow$	$O(\boldsymbol{n^2})$	$2.5n \rightarrow$	$O(\boldsymbol{n})$
$9n^3 + 100n^2 + 7n \rightarrow$	$O(\boldsymbol{n^3})$	$12 \rightarrow$	$O(\boldsymbol{1})$

b. Now we will count the operations in a real program. The function below takes a list `lst` of n numbers.

```
def sum_doubles(lst):
    total = 0
    for i in range(len(lst)):
        total = total + lst[i] * 2
    return total
```

How many times does the body of the `for` loop run, in terms of n? n **times**

c. Count each kind of operation in the whole function. Your answers should be in terms of n. (Do not forget the assignment on the line `total = 0`.)

assignments (`=`)	:	$n + 1$	(one before the loop, one per iteration)
multiplies (`*`)	:	n	(one `* 2` per iteration)
additions (`+`)	:	n	(one `+` per iteration)

Now add them up:

total = $3 \cdot n + 1$

Note: if you also count the list indexing `lst[i]` as an operation, the total becomes $4n + 1$. The Big-O answer is the same either way.

d. Apply step (2) to your answer from part c: drop the multiplicative constant and the lower-order term. What is the Big-O runtime?

$$O(\boldsymbol{n})$$

2. Reading Loops

For each program below, first write how many times the `print` line runs, and then write the Big-O runtime in terms of n where $n =$ `len(lst)`.

a.

```
for i in range(n):
    for j in range(n):
        print(i, j)
```

`print` runs n^2 times. Runtime: $O(\boldsymbol{n^2})$

b.

```
for i in range(n):
    print(i)

for j in range(n):
    print(j)
```

`print` runs $2n$ times. Runtime: $O(\boldsymbol{n})$

The loops are sequential, not nested, so we add instead of multiplying.

c.

```
i = 1
while i < n:
    print(i)
    i = i * 2
```

Hint: write out the values that `i` takes when $n = 16$:

1,

2, 4, 8, stop

(16 is not less than 16, so the loop stops there.)

`print` runs $\log_2 n$ times. Runtime: $O(\boldsymbol{\log n})$