Class 7 - Data Structures and Performance

MISE Summer Programming Camp 2023

Intro

Isabelle Quaye

Theoretical Computer Science(TCS) MEng @ MIT

MISE Math & Computer Science Summer Camp Alumni

Review of Class 6

- Recursion
  - Break a problem into smaller versions of the same problem
  - Base case and Recursive case
- More on List References
- Backtracking
  - Incrementally build candidates to solutions

Performance and Code Analysis

Thinking of Performance

We usually care about writing correct programs free of bugs

We also want our programs to run quickly

But how do we measure performance?

Big O Notation

- Notation we use for expressing how long a program takes to run
- Big O notation is often expressed in terms of the size of the input to your program(the variable **n** is often used for this) rather than the time it takes to actually run
- We like Big O notation because programs run on different machines which may be faster or slower
- Expressing performance in terms of input size makes things uniform
- There are a few rules/techniques for using Big O notation

Basic Rules in terms of unit of work

Variable assignment is considered **one unit** of work

Simple arithmetic, print statements and **conditional checks** are **one unit** of work

For/while loops over **n items** is considered **c*n units** of work

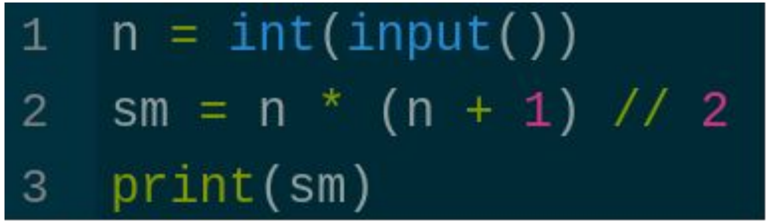
2 Nested Loops over n items takes **c*(n^2) units** of work

Units of work $\longrightarrow$ Big O notation

Unit of Work	Big O notation
1 unit of work	O(1)
n units of work	O(n)
5 units of work*	O(1)
2n units of work*	O(n)
n/2 units of work*	O(n)
n^2 units of work	O(n^2)
$\log_2 n$ units of work*	O(log n)

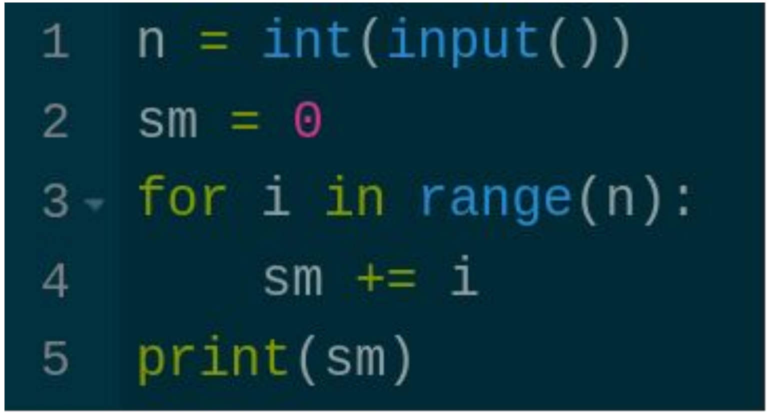
*** No multiplicative or additive constants in Big O!**

Big O notation in action



```python
n = int(input())
sm = n * (n + 1) // 2
print(sm)
```

Big O notation in action 2



```
n = int(input())
sm = 0
for i in range(n):
    sm += i
print(sm)
```

Time complexity

- Usually, we don't ask "how fast does your program run?"
- We usually ask "**what is the time complexity of your program?**"
- If we say this, we are asking **how fast your program is as express using Big O notation**.

Key Ideas from Performance

- Performance matters.
- Quantify performance using **Big O** notation.
- Big O notation is **in terms of the size of our program's input.**
- First figure out **units of work then convert to Big O notation**.
- **Time complexity** = how fast does your program run as written using Big O notation?

Example problem 1

Which of these is correct Big O notation?

A. O(5n)

B. O(log(n/2))

C. O(log n^2)

D. O(5)

Example problem 2

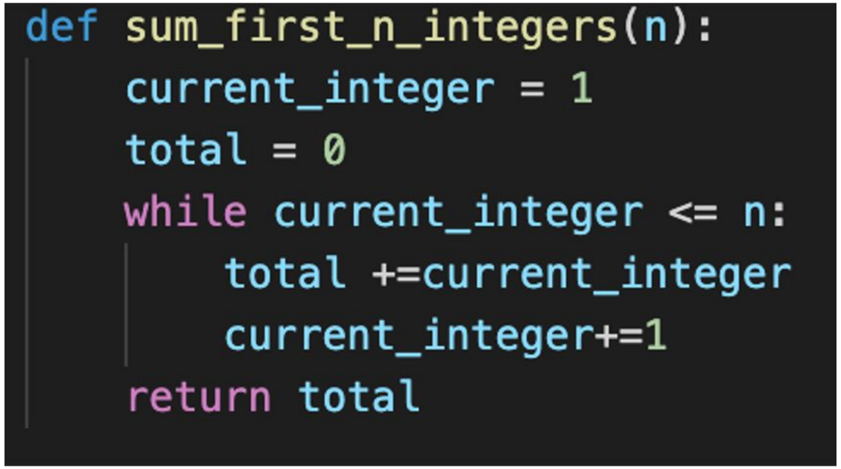
What is the time complexity of this function when executed?

A. O(1)

B. O(n/2)

C. O(log n)

D. O(n)



```python
def sum_first_n_integers(n):
    current_integer = 1
    total = 0
    while current_integer <= n:
        total +=current_integer
        current_integer+=1
    return total
```

Example problem 3

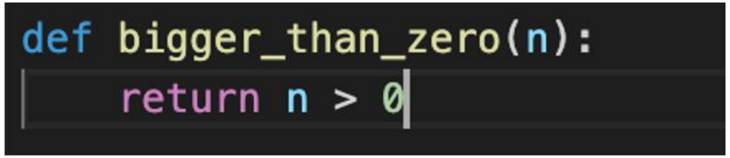
What is the time complexity of this function?

A. O(1)

B. O(n/2)

C. O(log n)

D. O(n)



```python
def bigger_than_zero(n):
    return n > 0
```

Data Structures

Data Structures

- A **data structure** is a way of organizing and storing data.
- One goal is to efficiently perform operations on that data.
- In Python, there are several built-in data structures, some of which you have seen
- Different data structures have different properties and are suited for different use cases
- We'll see two more examples: **sets** and **dictionaries**

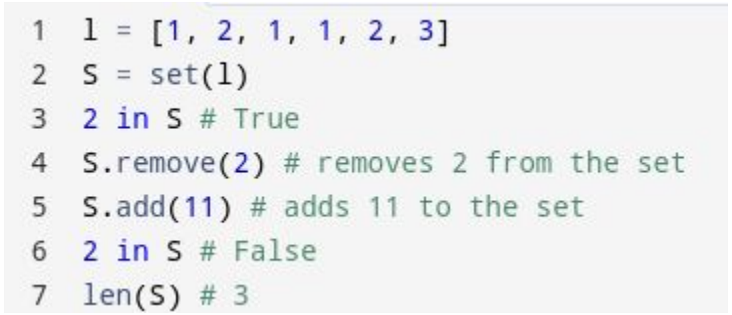
Lists as data structures

1. Inserting data:
   - *append()* method adds elements to the end of the list.
2. Removing data:
   - *remove()* method deletes elements by value
3. Accessing data:
   - Elements accessed using indexing, e.g. *a[3]* retrieves the 4th element
4. Additional Operations:
   - We can modify elements with assignment, e.g. *a[3]* = *1* sets the 4th element to 1
   - Find list length using *len()*
   - Slicing extracts subsets (see class 5)

Sets and Dictionaries

Other types of collections of data: Sets

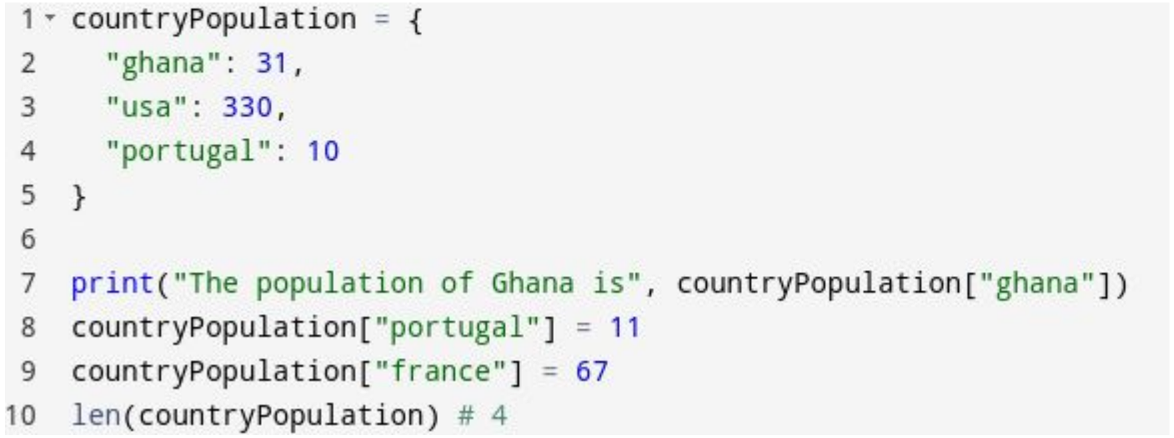
A *set* is a data structure that can hold multiple elements in no particular order. We can't index elements, but we can check for membership, add/remove elements and iterate over them. Sets work like math sets, meaning they only hold one of each unique element



```
l = [1, 2, 1, 1, 2, 3]
S = set(l)
2 in S # True
S.remove(2) # removes 2 from the set
S.add(11) # adds 11 to the set
2 in S # False
len(S) # 3
```

Other types of collections of data: Dictionaries

A *dictionary* is a data structure that maps keys to values in the same way that a list maps indexes to values. It’s named a dictionary because it works like an actual dictionary mapping keys (words) to values (definitions)



```python
countryPopulation = {
  "ghana": 31,
  "usa": 330,
  "portugal": 10
}

print("The population of Ghana is", countryPopulation["ghana"])
countryPopulation["portugal"] = 11
countryPopulation["france"] = 67
len(countryPopulation) # 4
```

Key Ideas from Data structures

- **Different data structures** are good for **different things**
- **Dictionaries** are great for **looking up items using a key** or identifier.
- **Sets** are useful for **checking membership**.

Example problem 4

Selom runs a summer remedial school. He wants to store the information for each student digitally.

Ideally, he would like to quickly look up students by their unique ID number. What kind of data structure(s) should Selom use?

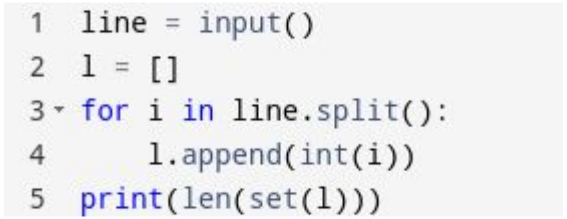
Example problem 5

Ama has a small business selling vegetables from her garden. She wants to store her daily sales digitally and asks you for help.
At the end of the week, she wants to know on which day she earned the most amount of money.

Example problem 6

You are given a list of integers. Count how many different integers there are.

Hint: Use a set!



```python
line = input()
l = []
for i in line.split():
    l.append(int(i))
print(len(set(l)))
```