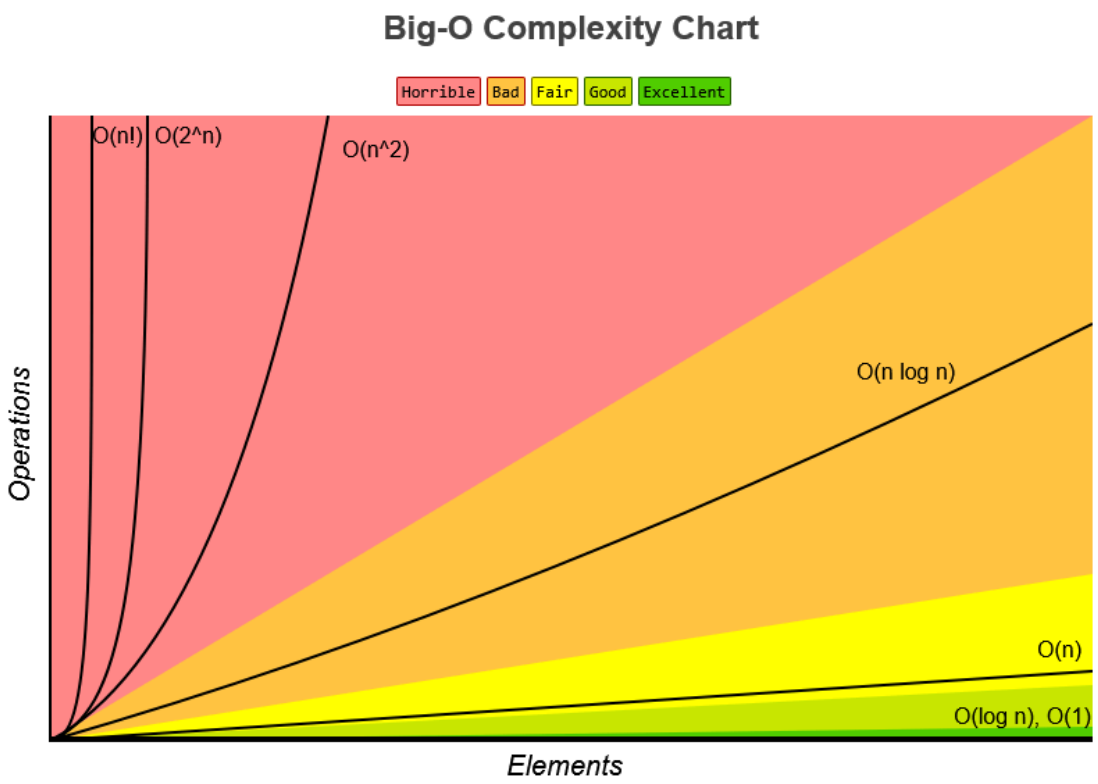
Python Built-in Operations: Time & Space Complexity Cheat Sheet

Python Built-in Operations: Time & Space Complexity Cheat Sheet **1**
- Basics 2
- Data Structures 3
  - List S:O(n) 4
  - Tuples S:O(n) 5
  - Sets S:O(n) 5
  - Dictionary S:O(n) 6
- Control Flow 7
- Callables 8
- Popular Modules and its methods 8
  - Math 8
  - Itertools 9
  - JSON 9
  - Date 9
  - RegEx 9
  - Random 9
  - Request 10
  - Statistics 10
  - OS 10
  - SYS 10
- File Handling 10
- MySQL 11



Ref- https://www.bigocheatsheet.com/

Goal: Keep your code operations below O(n log n) whenever possible — to optimize both time and memory.

This guide is designed for practical Python usage, not theoretical data structures or academic DSA problems.

Basics

print("Hello, World!") | Print |O(1)

Casting functions
- Mostly O(1)
- Some exceptions:-
  - str(n) | O(n)
  - str(int) | O(log n)
  - float(str) | O(n)
  - int(n) | O(n)
  - int(binary,base) | O(n)
- type(object) | O(1)

String functions
- Almost all built-in string functions are O(n)

    - Including popular ones: s.capitalize() | s.split(“,”) | s.find() | s.isalpha() | s.lower() | s.strip() | s.replace(“a”,”b”) | s.strip() |

- Exceptions:-
    - str.maketrans(s) | O(1)
    - str.startswith(k) | O(k)
    - str.endswith(k) | O(k)

- Concatenation
    - c = a + " " + b | Direct concatenation | O(a + b) or O(n)
    - "a += b" mutates a; time complexity is O(n) because a new copy may be created depending on reference count
    - s= a.join(b) | Joins: Best for Multiple concatenations | O(n)

- Format Strings. All complexity is O(n).
    - s.format() | Only use dynamically
    - txt = f"The price is {price:.2f} dollars" | F-String Fastest method.

Slicing in String. Applicable to Lists and Tuples.
- seq[start:stop] | O(k) - k elements | Applies to Negative Indexing
- seq[start:stop:step] | O(k)
- seq[:] | Fully Copy | O(n)
- seq[::-1] | Reverse sequence | O(n)

Boolean operations
- Logical : AND, OR, NOT | O(1)
- Comparison: (==, !=, >, <, >=, <=) | O(1)
- Identity: (is, is not) | O(1)
- Bitwise: (&, `, ^, ~, <<, >>) | O(1)
- 'true' if True else 'false' | Ternary operation | O(1)
- Exceptions:
    - Membership (in, not in) | O(n) - Scanning sequence required in List, Tuple and Strings. However O(1) for Sets and Dicts.
    - Quantifier: all(iter), any(iter) | O(n) worst-case | For any iterables List, Set, Tuple and Generators.

Main Operators
- Arithmetic | O(1)
- Assignment | O(1)
- Logical | Identity | Membership | Bitwise | O(1)
- Full reference https://www.w3schools.com/python/python_operators.asp

Built-in Math
- min(5, 10, 25) | min(iterable) | O(n)
- max(5, 10, 25) | max(iterable) | O(n)
- abs(-7.25) | O(1)
- pow(4, 3) | O(1)

Data Structures

Iteratables

List S:O(n)

Ordered, changeable, and allow duplicate

Basics

- print(list) | O(n)
- len(list) | O(1)
- type(list) | O(1)
- list[1] | list[-1] | O(1)
- list[2:5] | list[:4] | list[2:] | thislist[-4:-1] | O(k), k range of items
- if "apple" in list: | O(n)
- list.index("item") | Find the index with item | O(n)
- list.count("item") | Number of times specified item in the list | O(n)
- Others | max(list) | min(list) | sum(list) | all(list) | any(list) | O(n)

Manipulation

- list[1] = "new_item" | O(1)
- list[1:3] = ["new_item_1", "new_item_2"] | O(k)
- list.insert(2, "new_item") | O(n)
- list.append("new_item") | O(1)
- list1.extend(list2) | list1.extend(tuple) | O(m), m = length of new iterable
- list.remove("item") | O(n)
- list.pop() | O(1)
- list.pop(2) | O(n)
- del list[0] | O(n)
- del list | O(1)
- list.clear() | O(1)

Loop - If n is the length of list then O(n).

- for item in list:
- for index in range(len(list)):
- while i < len(list):
- [print(x) for x in thislist] | List comprehension
    - List comprehension rule | *newlist = [expression for item in iterable if condition == True]* | newlist = [x for x in fruits if "a" in x]

Sorting

- list.sort() | Timsort | O(n log n)
- list.sort(reverse = True) | Descending order | O(n log n)
- list.sort(key = customFunction) | Custom Function applied to each element before sorting | O(n log n)
- list.sort(key = str.lower) | case-sensitive | Using built-in function to each element before sorting | O(n log n)
- list.reverse() | O(n)
- random.shuffle(list) | Shuffling | O(n)

External Manipulation

- new_list = original_list.copy() | Copying list | O(n)
- new_list = list(original_list) | O(n)
- new_list = original_list[:] | Copying list using slice operator | O(n)
- list3 = list1 + list2 | Joining List | O(n)
- list1.extend(list2) | list1.extend(tuple) | Extend method to join list | O(m), m = length of new iterable

- list(set(original_list)) | List conversion - E.g. remove duplicates | O(n)

Tuples S:O(n)

Ordered, Unchangeable and allow duplicates

Basics

- Same as List:
    - print(tuple), len(tuple), type(tuple) | O(1)
- tuple[1] | tuple[-1] | O(1)
- tuple[2:5] | tuple[:4] | tuple[2:] | tuple[-4:-1] | O(k), k range of items
- if "item" in tuple: | O(n)
- tuple.index("item") | O(n)
- tuple.count("item") | O(n)
- Others | max(tuple) | min(tuple) | sum(tuple) | all(tuple) | any(tuple) | O(n)

Manipulation

- Tuple is **immutable.** Need some workarounds.
- Add item Option 1 - Requires converting to List, append item and then convert back to Tuple - O(n).
- orig_tuple += (“item1”, “item2”) | Add item 2 | O(n + k) or O(n)
- Delete item - Requires converting to List, remove item and convert back to Tuple - O(n)
- del tuple | O(1)
- tuple3 = tuple1 + tuple2 | Join two tuples | O(n+m) or O(n)
- new_tuple = orig_tuple * 2 | Multiply tuples | O(n)

Unpacking

- (var1, var2, var3) = tuple | O(k), k no of variables
- If the number of variables is less than the number of values, can add an “*****” to the variable name and the values will be assigned to the variable as a list - O(n)
    - fruits = ("apple", "banana", "cherry", "strawberry", "raspberry"); (green, yellow, *red) = fruits
    - fruits = ("apple", "mango", "papaya", "pineapple", "cherry"); (green, *tropic, red) = fruits

Loop

- Same as List | for | while | O(n)

Sets S:O(n)

Unordered, Mutable, No Duplicates

Basics

- my_set = set(("apple", "banana", "cherry")) | Set constructor | O(n)
- Same as List and Tuple:
    - print(tuple), len(tuple), type(tuple) | O(1)
- for x in thisset: | No indexing allowed for set. Have to iterate | O(n)
- if "item" in set: | O(1)
- Others | max(set) | min(set) | sum(set) | all(set) | any(set) | O(n)

Manipulation

- set.add("orange") | O(1)
- orig_set.update(new_set) | O(n)
- orig_set.update(new_list) | Can update with any iterable | O(n)
- orig_set.remove("item") | O(1) when amortized or worst-case O(n)
- orig_set.discard("item") | Like remove(), but no KeyError if an item is missing. | O(1)
- orig_set.clear() | O(1)
- del orig_set | O(1)

Relational Algebra methods

- Union. Excludes duplicate items. True - 1 are the same and False - 0 are the same.
  - set3 = set1.union(set2) | set3 = set1 | set2 | O(n + m) or O(n)
  - my_set = set1.union(set2, set3, set4) | O(n + m + p + r) or O(n)
  - new_set = old_set.union(new_tuple) | Joining set with List or Tuple | O(n)
- Intersection
  - set3 = set1.intersection(set2) | set3 = set1 & set2 | O(k), k = no of elements in set2
  - set1.intersection_update(set2) | Only update set1 without returning; keeps duplicates | O(n)
- Difference
  - set3 = set1.difference(set2) | set3 = set1 - set2 | C = A - B | O(n + m) or O(n)
  - set1.difference_update(set2) | O(n)
- Symmetric Difference - Exclusive Union - $A \oplus B = (A-B) \cup (B-A)$
  - set3 = set1.symmetric_difference(set2) | set3 = set1 ^ set2 | O(n+m) or O(n)
  - set1.symmetric_difference_update(set2) | O(n+m) or O(n)

Dictionary S:O(n)

Ordered, Changeable, Unique

Basics

- New Dictionary
  - new_dict = {} | O(1)
  - new_dict = {"key1": "value1", "key2": 1} | O(n)
- len(dict) | type(dict) | O(1)
- dict["key1"] | dict.get("key1") | dict.get("key1", default_value) | O(1)
- value = dict.setdefault("key1", "value_if_key1_does_not_exist") | O(1)
- Get keys list | dict.keys() | O(n)
- Get values list | dict.values() | O(n)
- Get key-value pair of tuples | dict.items() | O(n)
- if "key" in dict: | O(1)
- Others | max(dict) | min(dict) | all(dict) | any(dict) | O(n)

Manipulation

- Update or Add new item if key does not exist
  - dict["key1"] = “New value” | O(1)
  - dict.update({"key": “value”}) | O(m), m = number of new items
- Merge dictionary
  - dict1.update(dict2) | O(n + m) or O(n)
  - dict3 = dict1 | dict2 | O(n + m) or O(n)
- Remove item
  - dict.pop("key1") | O(1)
  - del dict["key1"] | O(1)

- dict.popitem() | Remove last added item | O(1)
- del dict | Delete dictionary | O(1)
- dict.clear() | Empty dictionary | O(1)
- newDict = OrigDict.copy() | newDict = dict(OrigDict) | Copy | O(n)
- newDict = *dict*.fromkeys(key_tuple, default_value_for_all_items) | Create new Dictionary from list/tuple of keys | O(k), k = length of keys

Loop

- for key in dict: | O(n)
- for value in dict.values(): | O(n)
- for key in dict.keys(): | O(n)
- for key, value in dict.items(): | O(n)
- for key in sorted(dict.keys()): | O(n log n)

Nested Dictionary | Time to access a deeply nested key is O(depth), assuming direct key access. Space grows exponentially only if nesting is deeply recursive with large branches — typically O(n) where n = total keys.

Control Flow

If-else elif | Conditional Statements | Almost all O(1)

- if b > a:
- elif a == b: | O(k) worst-case, all conditions are checked
- if a > b: print("a is greater than b") | Shorthand
- print("A") if a > b else print("B") | Shorthand If Else
- if a > b and c > a: | if a > b or a > c: | if not a > b: | With boolean operators
- if "item" in [list/tuple] : | O(n)
- if "item" in [set] : | O(1)

For Loop

- for item in iterable: | O(n)
- for index in range(r): | for index in range(start, end, step): | O(r), r = No of items or no. of items in range
- Nested loops | two or more
  - O(n*m) | for x in tuple1:
    for y in tuple2:
  - **O(n^2)** | for row in list:
    for col in list[row]:
  - **O(n^3)** | for x in list:
    for y in list[x]:
    for z in list[x][y]
- break | continue

While loop

- while condition is True: | O(c), Loop runs until condition is False.
- break | continue

Python Iterables - List, Tuple and Set

- my_iterator = iter(iterable) | Time:O(1), Space:O(n)
- next(my_iterator) | O(1)

Try - Except

- O(1)

```
try:
        Statements here. Complexity can increase here.
except:
        raise Exception(“Raise exception or built-in exceptions (33 available)”)
finally:
        Statements here
```

Callables

Function call

- my_function("1", 2) | O(1)

Lambda function

- lambda_func = *lambda arguments : expression* | lambda_func ("1", 2) | O(1)

Object and Classes

- object1 = Class1() | Instantiation | O(1)
- variable = object1.attribute1 | Attribute/variable call | O(1)
- object1.method1("1", 2) | Invoke Class method/function | O(1)
- super().__init__("1", 2) | Inheritance | O(1)

Modules

- import mymodule | import mymodule as mx | Python searches, loads, and compiles the module | O(n)
- mymodule.function() | Module function call | O(1)
- dir(mymodule) | List Module attributes | O(n)

All callable operations are O(1), but complexity depends on what the function does.

Popular Modules and its methods

Math

- math.sqrt(64) | O(1)
- math.ceil(1.4) | O(1)
- math.floor(1.4) | O(1)
- math.pi | O(1)
- math.exp(-6.89) | O(1)
- math.log2(8) | O(1)
- math.gcd(12, 36) | O(log n)
- math.prod(iterable) | O(n)
- math.factorial(12) | O(n)
- math.dist(list1, list2) | O(n)
- Full reference https://www.w3schools.com/python/module_math.asp

cMath - Mathematical tasks for Complex Number

- Full reference https://www.w3schools.com/python/module_cmath.asp

Itertools

- Full reference https://docs.python.org/3.10/library/itertools.html?highlight=pairwise

JSON

- Structure and storage | Space:O(1)
- json.loads({ "key1":"val1", "key2":30, "key3":"val3"}) | JSON to Python Dict | O(n)
- Python Object to JSON
    - json.dumps(python_object) | O(n)
    - json.dumps(dict, indent=4) | O(n)
    - json.dumps(dict, indent=4, separators=(". ", " = ")) | O(n)
    - json.dumps(dict, indent=4, sort_keys=True) | O(n log n)

Date

-

RegEx

- found_list = re.findall("ai", txt) | Find all matches using combination of Metacharacters, Special Sequence or Sets | Returns a List | O(n)
- matchObject = re.search("^The.*Spain$", my_text) | Search using combination of Metacharacters, Special Sequence or Sets | Returns a *Match* Object | O(n)
- matchObject = re.match("^The.*Spain$", my_text) | Same as Search but search matches from the beginning of the string | O(n)
- Match Object
    - range_tuple = matchObject.span() | Tuple containing start and end position of the first match occurrence | O(1)
    - str = matchObject.string | Return the string passed into the function | O(1)
    - str = matchObject.**group()** | Return the part of the string where there was a match | O(1)
- **O(n^2)** or worse cases
    - re.match(".*?", my_text) | Backtracking patterns (.*, .*?)
    - re.match("(a+)+", my_text) | Nested quantifiers ((a+)+)
- match_list = re.split("\s", txt, 4) | Return a list where string has been split at each match | O(n)
- replaced_text = re.sub("\s", "replacing text", txt) | Replaces the matches with the text | O(n)
- Reference to Metacharacters, Special Sequence and Sets https://www.w3schools.com/python/python_regex.asp

Random

- random.random() | O(1)
- random.randint(3, 9) | O(1)
- random.randrange(3, 9, 2) | O(1)
- random.shuffle(sequence) | O(n)
- random.sample(sequence, 3) | O(k), k = No. of elements specified. In this case 3.

- random.seed(9) | O(1)
- Full reference https://www.w3schools.com/python/module_random.asp

Request

- requests.get('https://some-website.com', params_dict, args*) | O(n)
- requests.post('https://some-website.com', data_dict, args*) | O(n)
- requests.delete('https://some-website.com', args*) | O(n)
- Other methods | head(), patch(), put() and request() | O(n)
- Full reference https://www.w3schools.com/python/module_requests.asp

Statistics

- statistics.mean(iterable) | O(n)
- statistics.median(iterable) | O(n)
- statistics.mode(iterable) | O(n)
- statistics.variance(iterable) | O(n)
- statistics.stdev(iterable) | O(n)
- Full reference https://www.w3schools.com/python/module_statistics.asp

OS

- os.mkdir("mydir") | Create new directory | O(1)
- os.getcwd() | Current working directory | O(1)
- os.listdir(".") | List specified directory | O(n)
- os.remove("file.txt") | Delete file | O(1)
- os.path.dirname("path") | Directory name of pathname | O(1)
- os.path.exists("path") | True if path exists of not | O(1)
- os.path.isdir("directory/path") | os.path.isfile("file/path") | Verify if name is a dir or path | O(1)
- os.path.join("folder", "subfolder", "file.txt") | Create file path dynamically | O(k), k= no of path components.
- Full reference https://www.w3schools.com/python/module_os.asp?ref=escape.tech
- Full reference **os.path** https://docs.python.org/3/library/os.path.html

SYS

- sys.path("path") | List of directories on a specified path | O(n)
- sys.exit("message") | Stop program execution | O(1)
- sys.argv | List of arguments passed to the Python script | O(n)
- Full reference https://docs.python.org/3/library/sys.html

File Handling

Read

- f = open("demofile.txt", "rt") | Create File object | O(1)
- f.read() | Read everything | O(n)
- f.readline() | Read line by line | O(1)
- for x in f: | Loop through the file line by line | O(n)
- f.close() | Close | O(1)
- Modes reference https://www.w3schools.com/python/python_file_handling.asp

Editing

- ‘a’ append | ‘w’ write | ‘x’ creates a new file
- f = open("demofile2.txt", "a") | O(1)
- f.write("Now the file has more content!") | O(1) but O(n) in most cases.
- os.remove("demofile.txt") | O(1)

MySQL

Connect

- import mysql.connector
- myDb = mysql.connector.connect("host","*user*","*password*",”db”) | Connect to database | O(1)
- myDb.cursor() | Create Cursor object | O(1)
- myCursor.execute("CREATE DATABASE mydatabase") | Execute SQL query| O(1)

CRUD operations - Mostly handled by SQL queries

- myCursor.execute("SELECT * FROM customers") | O(r), r = Selected row in the Database
- myCursor.rowcount | O(1)
- my_results = myCursor.fetchall() | Fetches all rows from the last executed statements | O(r)
- for x in my_results : | Loop through result | O(n)
- my_results = myCursor.fetchone() | Fetch the first row of the result | O(1)
- myCursor.execute("INSERT INTO table1(col1, col2) VALUES (%s, %s)", tuple_value) | Execute SQL query with dynamic value (Good practice against SQL injection) | O(t), t = Size of tuple
- myCursor.executemany("UPDATE table1 SET col2 = %s WHERE col1 = %s", list_of_tuples) | Another example for dynamic value for an UPDATE operation | O(t)
- mydb.commit() | Confirm any changes to the database | O(1)
- Multi-line SQL comments

```
sql = "SELECT \
  users.name AS user, \
  products.name AS favorite \
  FROM users \
  INNER JOIN products ON users.fav = products.id"
```