SI SESSION: GCIS-123

Units 7-9 Exam Material Review

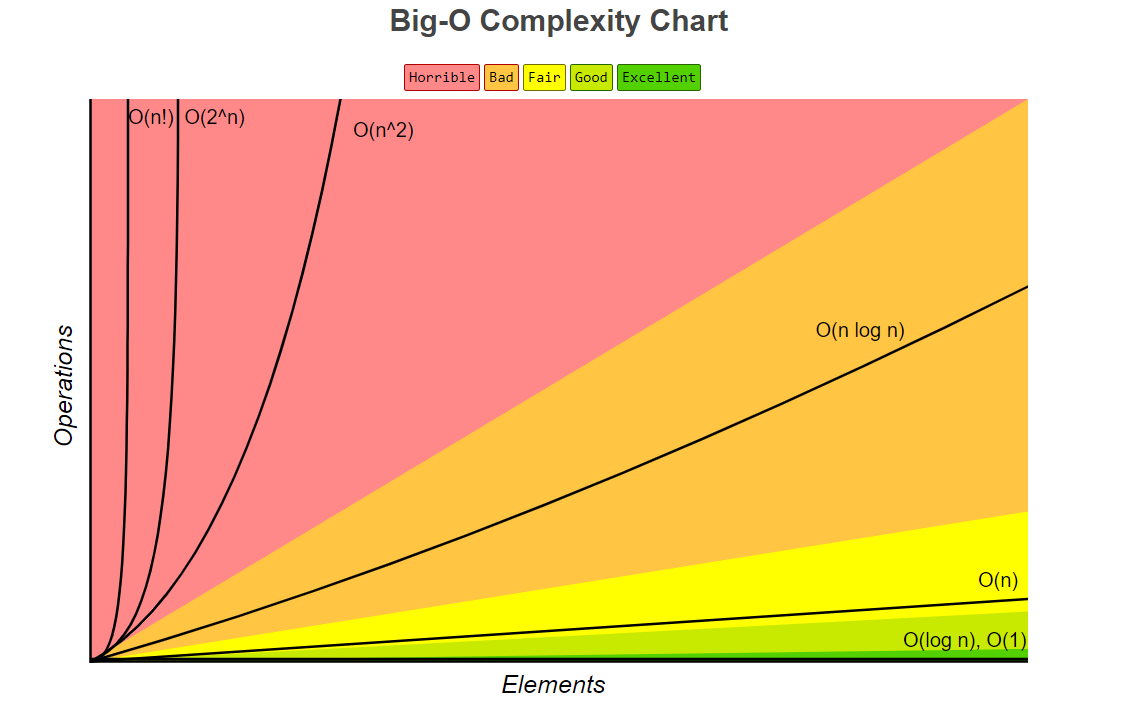
# 

# Unit 7: Sorting

* In-place: original destroyed
* Not in-place: makes a copy

## Time Complexity

* import time
* (begin = time.perf\_counter()) - (end = time.perf\_counter()) = elapsed time
* Can be used to predict time complexity (number of operations)
  + Time complexity is the *average* time it takes to complete an operation (scalars not important)



## 

## Insertion Sort (Nieve Sort)

* Method:
  + Array split into sorted and unsorted half
  + For each element in unsorted half, move into sorted half
  + *Swap* or *shift* left until in correct (sorted) position
  + Repeat for entire list
* Time Complexity:
  + Best Case O(n)
  + Average O(n2)
  + Worst Case O(n2)

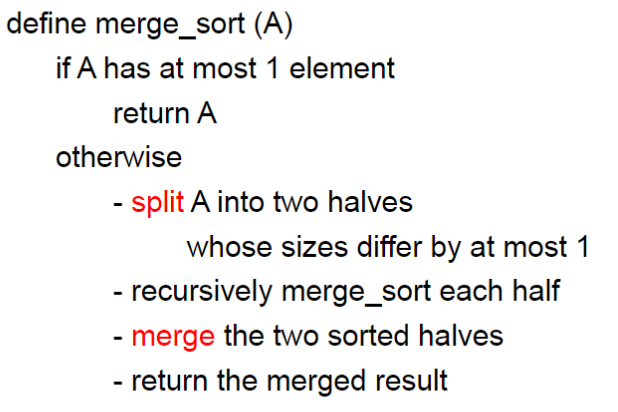
## 

## Function Parameters

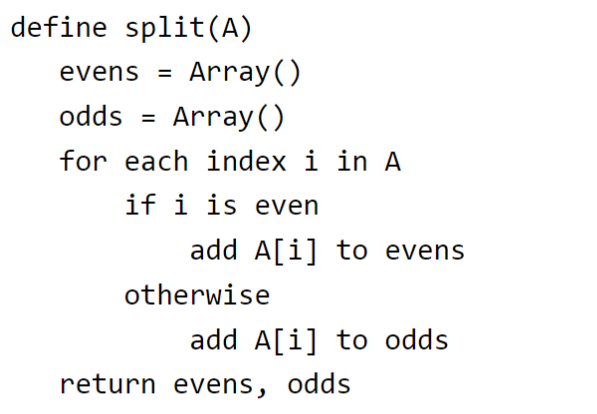
* Pass a function as an argument into another function
* function.\_\_name\_\_
* passed without parenthesis

## 

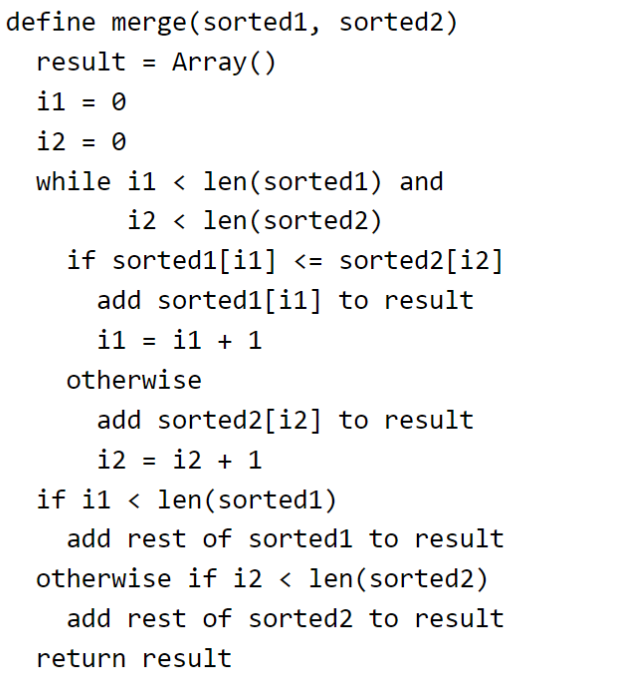
## Merge Sort



* + Functions

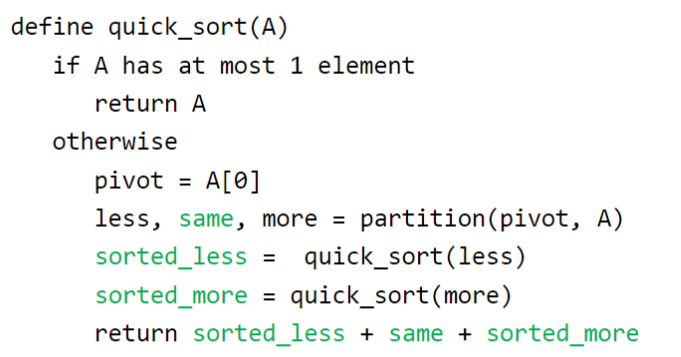


* + Can Return multiple variables by comma-separating



* + Divide and conquer algorithm
  + Time Complexity: O(n log n)

## Quicksort



* Time Complexity:
  + Average O(n log n)
  + Worst O(n2)

# Unit 8: Lists & Tuples

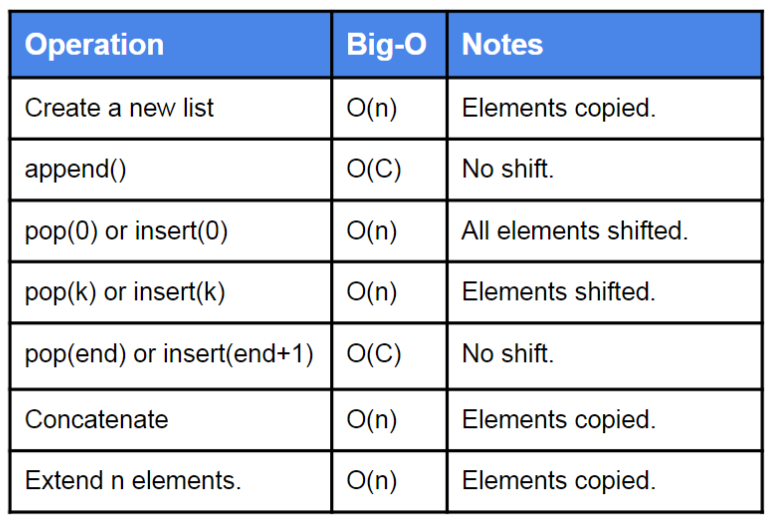
* Sequences
  + data structures that group related elements
  + len() to find length

## Tuples

* Fixed length of any type
* Immutable, comma separated, iterable
* use tuple(sequence) to create list

## Lists

* Dynamically sized array of any type
* Mutable, comma separated, iterable
* use list(sequence), = [] or [sequence] to create list
* list.append(value) to add value to the end and increase length
* Value Types
  + “passing by value”: a copy of a parameter is passed into the function
  + ex. int, float, and bool
* Reference Types
  + “passing by reference”: parameter where changes in a function persist
  + ex. list
* Concatenation
  + + used to create a new list that contains all of the elements in two other lists (unchanged)
  + Concatenating a value that is not a a list will cause a TypeError
* Extension
  + += in-place addition
  + Original list is modified to include the elements from the other sequence
* Popping & Inserting
  + .pop(index) permanently remove and return the element at the specified index (or last element if not specified), length -= 1
  + .insert(index, value) insert a value into the specified index, length += 1
  + destructive, permanent modifications



* Slicing
  + [start : stop : step (optional)], to create a new list by selecting specific values from an existing list
  + Default values of [0 : length : 1]
* Sorting
  + sorted(list): create a sorted copy of the list (non-destructive)
  + .sort(): in-place sort (destructive)
  + optional second parameter reverse = True/False
  + optional sort key, key = len/str.lower to change behavior
  + can pass function with one parameter as the key if it returns some kind of comparable value
* List Comprehension
  + [*expression* for *item* in *sequence (*optional if *conditional)*]
  + Creating lists out of sequences
* 2D Lists
  + a list of lists
  + like a table storing data in rows and columns
  + value = list[row][column]

# Unit 9: Sets & Dictionaries

* Data Structure: grouping of related values

## Sets

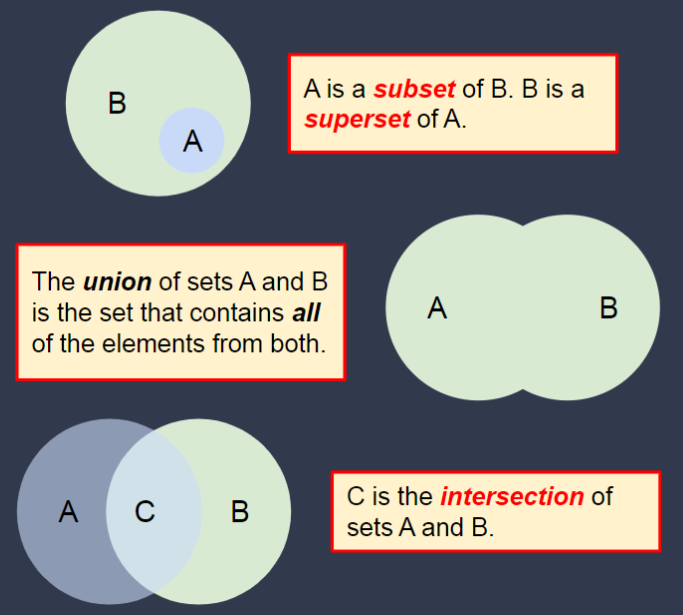
* All elements are unique
* Unordered (cannot access by index)
* if *element* in *set*
* set = {}
* .add(*element*)
* sorted(*set*)
* Much lower time complexity
* Works as a hashing data structure built on top of an array

**Hashing**

* Hash Function: converts arbitrary values into integers (hash codes), which can then be used to find an index in the array
* The same input will always produce the same output
* Collisions: two different values are hashed into the same index in an array
* Can resolve collisions using open addressing and chaining
* Open addressing: search for the next open address in which to store the value
* Chaining: elements are stored in a list at each index
* Hashing function: fast, consistent, minimizes collisions.
* Index = hashcode % length

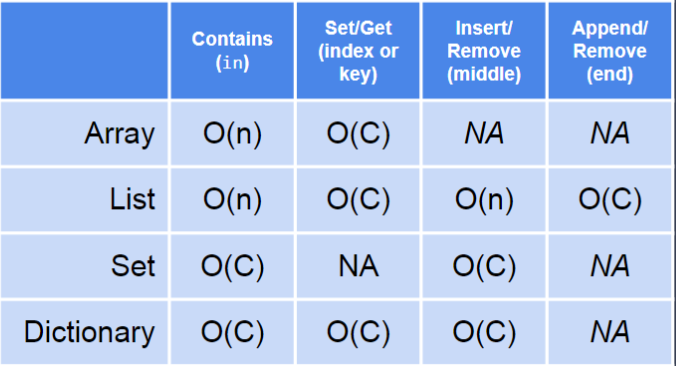
## Set Theory

* ∅ is the null or empty set, containing no elements, i.e. ∅ = {}
* A ∈ B: A is an **element** or member of B, i.e. set B contains A.
* A ⊂ B: A is a **subset** of B, i.e. set B contains all of the elements of set A.
* A ⊃ B: A is a **superset** of B, i.e. set A contains all of the elements of set B.
* A ∪ B: **Union** of sets A and B, contains all of the elements of both A and B.
* A ⋂ B: **Intersection** of sets A and B, and is a set that contains only the elements that are in both A and B.



## Dictionaries

* Hashing data structure to store key : value pairs
* dict() function or a pair of empty curly braces {}
* key:value pairs, use keys like indexes
* if key in a\_dict: used to check if a key is in a dictionary
* Retrieving value using a key that is not in the dictionary raises a KeyError
* New values replace old values for the same key
* dictionaries maintain their keys in insertion order
* keys() and sorted() functions to get sorted list of keys



## ASCII

* Encoding characters as integers
* ord(char): translates a character to an integer.
* chr(code): translates an integer to a character.