Exercise 1

Code

```
""" CMPSC 413 - Lab 2: Searching and Sorting


1. Write and implement the algorithm for Linear search, Binary search, Insertion sort, Selection sort
and Bubble sort. Calculate the time complexity for these searching and sorting algorithms.
Table-1: tabulate the time complexity for these algorithms with best and worst time
complexities.

"""
import random

def linear_search(val, data=[]):
    for v in data:
        if v == val:
            return val

def binary_search(val, data=[]):
    if not data:
        return None

    # sort data first (use insertion since we might get a sorted list...insertion is faster)
    data = insertion_sort(data)

    mid = len(data)//2
    l,r = 0, len(data)-1

    while l <= r:
        mid = l + ((r - l) // 2) # find mid again based on l,r

        if val > data[mid]:
            l = mid + 1

        elif val < data[mid]:
            r = mid - 1

        elif val==data[mid]: # got it
            return (data[mid], f'idx={ mid }')

    return None

def selection_sort(data=[]):
    if not data:
        return None

    for i in range(len(data)):
        min_idx = i # store index of current min element

        for j in range(i + 1, len(data)):

            # find the minimum of the unsorted portion
            if data[j] < data[min_idx]:
                min_idx = j

        # swap
        temp = data[i]
        data[i] = data[min_idx]
        data[min_idx] = temp
```

```
    return data

def insertion_sort(data=[]):
    if not data:
        return None


    for i in range(1, len(data)):

        key = data[i] # value were comparing
        comp_idx = i - 1

        # shift elements until correct spot is found
        while key < data[comp_idx] and comp_idx > 0:
            data[comp_idx + 1] = data[comp_idx]
            comp_idx -= 1

        # putting the data in the right spot
        data[comp_idx + 1] = key


    return data

def bubble_sort(data=[]):
    if not data:
        return None

    for i in range(len(data)):
        for j in range(0, len(data) - i - 1): # sorted part at end of list
            if data[j] < data[i]:
                # swap
                temp = data[i]
                data[i] = data[j]
                data[j] = temp

    return data


print(binary_search(2, [random.randint(0,100) for i in range(1000)]))
```

Time Complexities

Best Time Complexities

Algorithm	Best Time Complexity
Insertion	O(n)
Selection	$O(n^2)$
Bubble	O(n)
Merge	O(n log n)
Linear Search	O(1)
Binary Search	O(1)

Worst Time Complexities

Algorithm	Worst Time Complexity
Insertion	$O(n^2)$

Algorithm	Worst Time Complexity
Selection	$O(n^2)$
Bubble	$O(n^2)$
Merge	O(n log n)
Linear Search	O(n)
Binary Search	O(log n)

Exercise 2

StudentDb

this is the database which contains all the functionality of creating students, adding them to a databse (text or json file) and sorting copies of the data.

```
class studentDb:
    def __init__(self):
        self.db_file_name = "student_db.json"

        self.students = []
        self.test_fnames = ['Aarav', 'Luna', 'Kai', 'Nia', 'Yusuf', 'Sakura', 'Ivan', 'Fatima', 'Jin', 'Alejandra',
'Emeka', 'Anya', 'Hiroshi', 'Ingrid', 'Kwame', 'Marta', 'Raj', 'Chen', 'Yelena', 'Diego']
        self.test_lnames = ['Kim', 'Müller', 'Singh', 'Lebedev', 'Okeke', 'Fernandez', 'Wong', 'Rossi', 'Silva', 'Kováč',
'Nguyen', 'Khan', 'Petrov', 'Díaz', 'Sato', 'Ibrahim', 'Johansson', 'Chen', 'Novak', 'Moreau']
        self.test_majors = [
                                        "Computer Science",
                                        "Mechanical Engineering",
                                        "Psychology",
                                        "Business Administration",
                                        "Biology",
                                        "Political Science",
                                        "Economics",
                                        "English Literature",
                                        "Graphic Design",
                                        "History",
                                        "Environmental Science",
                                        "Nursing",
                                        "Mathematics",
                                        "Chemical Engineering",
                                        "Sociology",
                                        "Music",
                                        "Art History",
                                        "Physics",
                                        "Philosophy",
                                        "Anthropology"
                                        ]
        self.format_student_data()

    def format_student_data(self):
        self.test_f_names = [i.lower() for i in self.test_fnames]
        self.test_l_names = [i.lower() for i in self.test_lnames]
        self.test_majors = [i.lower() for i in self.test_majors]

    def make_students(self, count = 0):
        for i in range(count-1):
            # make a student
            s = {}

            s["id"]     = int(''.join([str(random.randint(1,9)) for i in range(10)]).rstrip().lstrip()) # 0 could be first
digit so we cant start it with 0
```

```
            s["f_name"] = random.choice(self.test_f_names).lower()
            s["l_name"] = random.choice(self.test_l_names).lower()
            s["email"]  = ''.join([''.join(s["f_name"][0:2]) + ''.join(s["l_name"][0]) + ''.join([str(random.randint(0,9))
for i in range(4)]) + "@psu.edu"]) # first 2 letters of f_name, last of l_name
            s["major"]  = ''.join(random.choice(self.test_majors)).lower()


            self.add_student(s)

    def add_student(self, student=None):
        self.students.append(student)

    def delete_all_students(self):
        self.students = []

    def write_to_db(self, file_name=None):
        if not file_name:
            file_name = self.db_file_name

        with open(file_name, 'w') as db:
            json.dump(self.students, db, indent=4) # indent makes it more readable

    def read_from_db(self):
        with open(self.db_file_name, 'r') as db:
            return json.load(db)

    # ------------------------ Linear Search ------------------------
    def linear_search(self, val=None, param=None):
        if not val or not param:
            return None

        for student in self.read_from_db():
            if student[param] == val:
                return student

        return None

    def binary_search(self, val, param):
        if val is None or param is None:
            return None

        # sort data  based on param
        data = self.merge_sort(param=param)

        l, r = 0, len(data) - 1

        while l <= r:
            mid = l + (r - l) // 2

            # compare param field at mid index with target
            if data[mid][param] < val:
                l = mid + 1
            elif data[mid][param] > val:
                r = mid - 1
            else:
                return data[mid]  # found the target

        return None  # didnt find the target

    # ------------------------ sorting algorithms ------------------------
    def insertion_sort(self, param=None):
        if not param:
            return None

        data = self.read_from_db()

        for i in range(1, len(data)):
```

```
            key = data[i]  # value were comparing
            comp_idx = i - 1

            # shift elements until  correct spot is found
            while comp_idx >= 0 and key[param] < data[comp_idx][param]:
                data[comp_idx + 1] = data[comp_idx]
                comp_idx -= 1

            # putting the data in the right spot
            data[comp_idx + 1] = key

        return data

    def selection_sort(self, param=None):
        if not param:
            return None

        data = self.read_from_db()

        for i in range(len(data)):
            min_idx = i  # store index of current min element

            for j in range(i + 1, len(data)): # sorted portion forms at start
                # find the minimum of the unsorted portion
                if data[j][param] < data[min_idx][param]:
                    min_idx = j

            # Swap
            data[i], data[min_idx] = data[min_idx], data[i]

        return data

    def bubble_sort(self, param=None):
        if not param:
            return None

        data = self.read_from_db()

        for i in range(len(data)-1):
            for j in range(0, len(data) - i - 1):  # sorted part at end of list
                if data[j][param] < data[j+1][param]:
                    # swap
                    data[i], data[j] = data[j], data[i]

        return data

    # merge sort using nested functions to keep it clean
    def merge_sort(self, param=None):
        if not param:
            return None

        data = self.read_from_db()

        # merging function
        def merge(left, right):
            result = []
            i = j = 0

            # recombining data
            while i < len(left) and j < len(right):
                if left[i][param] < right[j][param]:
                    result.append(left[i])
                    i += 1
                else:
                    result.append(right[j])
                    j += 1
```

```
            result.extend(left[i:])
            result.extend(right[j:])
            return result

        # dividing/entry function
        def merge_sort_recursive(arr):
            if len(arr) <= 1:
                return arr

            # splitting process
            mid = len(arr) // 2
            left = arr[:mid]
            right = arr[mid:]

            # recurse till len(1)
            left = merge_sort_recursive(left)
            right = merge_sort_recursive(right)

            return merge(left, right)

        sorted_data = merge_sort_recursive(data)
        return sorted_data
```

Read and Write To Database (File)

```
    def write_to_db(self, file_name=None):
        if not file_name:
            file_name = self.db_file_name

        with open(file_name, 'w') as db:
            json.dump(self.students, db, indent=4) # indent makes it more readable

    def read_from_db(self):
        with open(self.db_file_name, 'r') as db:
            return json.load(db)
```

Sorting

Setup Code

```
# ---------------------------------- Generating Stats ----------------------------------

# function to get running times
def get_run_times(f, param, count=100):
    start = perf_counter() # perf_counter is higher res than time.process_time() (was getting 0.0 times)
    for i in range(count):
        f(param)  # calls func (ill use class function studentDb.<f>)
    stop = perf_counter()

    return (stop - start) / count


sdb = studentDb()
sdb.make_students(20)
print(sdb.students)  # initial db that the unsorted portion of the code
sdb.write_to_db()

sort_stats = { k:None for k in sdb.students[0].keys() }

params = list(sdb.students[0].keys()) # getting all params except email
```

```
params.remove("email")
params.remove("major")

print(params)

sort_funcs = {"insertion" : studentDb.insertion_sort, "selection" : studentDb.selection_sort, "bubble" :
studentDb.bubble_sort, "merge" : studentDb.merge_sort}

# holds dbs sorted by a given param
sorted_student_dbs = {}
# holds sorting times for each sort over a db sorted by some param
sorted_student_dbs_by_param_times = {}
# holds lists of students sorted by params (id, email,..,)
sorted_students_by_param = {}
```

Sorting Unsorted Data

```
# populates and prints sort_times with time for sorting each param
# will run program twice for table-4
def sort_all():
    sort_times = {
        "insertion": {},
        "selection": {},
        "bubble": {},
        "merge": {}
    }

    for p in params:
        print(f"\n\tSort Parameter: { p }\n")
        for sort_algo in sort_times:
            sort_times[sort_algo][p] = get_run_times(sort_funcs[sort_algo], param=p)
            print(f"Sort Algorithm: { sort_algo } took { sort_times[sort_algo][p] }")

    return sort_times

# printing sort times for unsorted students for each algo based on each parameter
print(sort_all())
print("\n")
```

Tables

Algorithm	ID	First Name	Last Name
Insertion	8.30e-08	5.20e-08	5.30e-08
Selection	5.40e-08	5.00e-08	5.00e-08
Bubble	5.30e-08	5.10e-08	5.10e-08
Merge	8.50e-08	8.20e-08	1.46e-07

Tables of best results

Parameter	Best Algorithm	Best Time (s)
ID	Selection	5.40e-08
First Name	Selection	5.00e-08
Last Name	Selection	5.00e-08

Sorting Sorted Data

within the StudentDb() code are all the sorting and searching algorithms used. Below is the application of these algorithms. Each Algorithm only makes a copy of the data before returning a sorted list of dictionary values representing the student. This is not a problem for sorting an unsorted dictionary of students, but was a problem for sorting a sorted dictionary. This was solved below with a forloop that saved instances of StudentDb() in the following way:

```
# sorting the data by each param with merge, then running algos over it again
for p in params:
    sorted_student_dbs[p] = studentDb() # create new studentDB
    sorted_student_dbs[p].make_students(20) # randomly populate
    sorted_student_dbs[p].merge_sort(param=p) # sort
    sorted_student_dbs[p].write_to_db() # overwrite file with current sorting by param

    # run and time each algo over the file    sorted_student_dbs_by_param_times[p] = {f : get_run_times(sort_funcs[f],
param = p) for f in sort_funcs.keys()}

print(f"Sorting Already Sorted Data: { sorted_student_dbs_by_param_times }")

print(f"\nAlgorithm performance over sorts by parameter times { sorted_student_dbs_by_param_times }")
#print(sort_times)
```

Tables

Parameter	Algorithm	Time (s)
ID	Insertion	7.40e-08
ID	Selection	5.20e-08
ID	Bubble	4.90e-08
ID	Merge	8.10e-08
First Name	Insertion	7.60e-08
First Name	Selection	5.50e-08
First Name	Bubble	4.90e-08
First Name	Merge	8.70e-08
Last Name	Insertion	7.00e-08
Last Name	Selection	5.10e-08
Last Name	Bubble	4.80e-08
Last Name	Merge	8.90e-08

Table of best results

Parameter	Best Algorithm	Best Time (s)
ID	Bubble	4.90e-08
First Name	Bubble	4.90e-08
Last Name	Bubble	4.80e-08

Output of first run

```
C:\Users\aaron\AppData\Local\Programs\Python\Python39\python.exe
C:\Users\aaron\Projects\classWork\CMPSC413\lab2\student_db_manager.py
[{'id': 6643272251, 'f_name': 'diego', 'l_name': 'nguyen', 'email': 'din2513@psu.edu', 'major': 'anthropology'}, {'id':
3848328912, 'f_name': 'kwame', 'l_name': 'silva', 'email': 'kws6360@psu.edu', 'major': 'environmental science'}, {'id':
4512682274, 'f_name': 'ingrid', 'l_name': 'johansson', 'email': 'inj5227@psu.edu', 'major': 'anthropology'}, {'id':
```

4137236651, 'f_name': 'chen', 'l_name': 'moreau', 'email': 'chm7641@psu.edu', 'major': 'nursing'}, {'id': 1388733766, 'f_name': 'yusuf', 'l_name': 'sato', 'email': 'yus9655@psu.edu', 'major': 'business administration'}, {'id': 6662567767, 'f_name': 'kai', 'l_name': 'kováč', 'email': 'kak6191@psu.edu', 'major': 'nursing'}, {'id': 9358962951, 'f_name': 'chen', 'l_name': 'singh', 'email': 'chs1654@psu.edu', 'major': 'anthropology'}, {'id': 9727774923, 'f_name': 'nia', 'l_name': 'khan', 'email': 'nik5838@psu.edu', 'major': 'political science'}, {'id': 5872897242, 'f_name': 'fatima', 'l_name': 'johansson', 'email': 'faj4310@psu.edu', 'major': 'chemical engineering'}, {'id': 7917176368, 'f_name': 'yusuf', 'l_name': 'moreau', 'email': 'yum4670@psu.edu', 'major': 'mathematics'}, {'id': 6914235662, 'f_name': 'ivan', 'l_name': 'okeke', 'email': 'ivo9704@psu.edu', 'major': 'political science'}, {'id': 6535397353, 'f_name': 'raj', 'l_name': 'chen', 'email': 'rac4338@psu.edu', 'major': 'philosophy'}, {'id': 4913118877, 'f_name': 'kwame', 'l_name': 'petrov', 'email': 'kwp6302@psu.edu', 'major': 'political science'}, {'id': 8438684975, 'f_name': 'fatima', 'l_name': 'müller', 'email': 'fam5297@psu.edu', 'major': 'mechanical engineering'}, {'id': 5383563727, 'f_name': 'diego', 'l_name': 'díaz', 'email': 'did0376@psu.edu', 'major': 'economics'}, {'id': 3578815152, 'f_name': 'sakura', 'l_name': 'singh', 'email': 'sas5227@psu.edu', 'major': 'sociology'}, {'id': 5417317158, 'f_name': 'diego', 'l_name': 'chen', 'email': 'dic3158@psu.edu', 'major': 'biology'}, {'id': 2473254742, 'f_name': 'alejandra', 'l_name': 'lebedev', 'email': 'all1261@psu.edu', 'major': 'computer science'}, {'id': 4732756891, 'f_name': 'fatima', 'l_name': 'fernandez', 'email': 'faf7604@psu.edu', 'major': 'psychology'}]
['id', 'f_name', 'l_name']

        Sort Parameter: id

Sort Algorithm: insertion took 8.300000000016627e-08
Sort Algorithm: selection took 5.4000000000165026e-08
Sort Algorithm: bubble took 5.2999999999858716e-08
Sort Algorithm: merge took 8.499999999994623e-08

        Sort Parameter: f_name

Sort Algorithm: insertion took 5.2000000000107516e-08
Sort Algorithm: selection took 5.0000000000050006e-08
Sort Algorithm: bubble took 5.100000000007876e-08
Sort Algorithm: merge took 8.199999999985997e-08

        Sort Parameter: l_name

Sort Algorithm: insertion took 5.2999999999858716e-08
Sort Algorithm: selection took 5.0000000000050006e-08
Sort Algorithm: bubble took 5.100000000007876e-08
Sort Algorithm: merge took 1.46000000000035e-07

Times for Unsorted Data: {'insertion': {'id': 8.300000000016627e-08, 'f_name': 5.2000000000107516e-08, 'l_name': 5.2999999999858716e-08}, 'selection': {'id': 5.4000000000165026e-08, 'f_name': 5.0000000000050006e-08, 'l_name': 5.0000000000050006e-08}, 'bubble': {'id': 5.2999999999858716e-08, 'f_name': 5.100000000007876e-08, 'l_name': 5.100000000007876e-08}, 'merge': {'id': 8.499999999994623e-08, 'f_name': 8.199999999985997e-08, 'l_name': 1.46000000000035e-07}}

dict_items([('id', <__main__.studentDb object at 0x000001B27DBDAD90>), ('f_name', <__main__.studentDb object at 0x000001B27DBDACD0>), ('l_name', <__main__.studentDb object at 0x000001B27DBD4FD0>)])

Algorithm performance over sorts by parameter times {'id': {'insertion': 6.600000000023255e-08, 'selection': 4.699999999996374e-08, 'bubble': 4.7999999999992495e-08, 'merge': 2.540000000000875e-07}, 'f_name': {'insertion': 6.399999999989747e-08, 'selection': 5.100000000007876e-08, 'bubble': 2.089999999999037e-07, 'merge': 8.299999999988872e-08}, 'l_name': {'insertion': 6.299999999986872e-08, 'selection': 5.0000000000050006e-08, 'bubble': 1.3499999999999625e-07, 'merge': 8.599999999997499e-08}}

Process finished with exit code 0

Second Run Output

this corresponds to table-4

```
[{'id': 9886178637, 'f_name': 'ivan', 'l_name': 'kováč', 'email': 'ivk9070@psu.edu', 'major': 'biology'}, {'id':
9424772584, 'f_name': 'luna', 'l_name': 'novak', 'email': 'lun1206@psu.edu', 'major': 'political science'}, {'id':
8134457451, 'f_name': 'kai', 'l_name': 'novak', 'email': 'kan1399@psu.edu', 'major': 'art history'}, {'id': 9221595235,
'f_name': 'kwame', 'l_name': 'díaz', 'email': 'kwd3771@psu.edu', 'major': 'sociology'}, {'id': 9359497293, 'f_name':
'kai', 'l_name': 'díaz', 'email': 'kad5154@psu.edu', 'major': 'philosophy'}, {'id': 6451643655, 'f_name': 'yelena',
'l_name': 'ibrahim', 'email': 'yei6361@psu.edu', 'major': 'anthropology'}, {'id': 8837942676, 'f_name': 'jin', 'l_name':
'okeke', 'email': 'jio3028@psu.edu', 'major': 'biology'}, {'id': 5654635759, 'f_name': 'aarav', 'l_name': 'nguyen',
'email': 'aan1979@psu.edu', 'major': 'mechanical engineering'}, {'id': 3579315332, 'f_name': 'nia', 'l_name': 'nguyen',
'email': 'nin7317@psu.edu', 'major': 'nursing'}, {'id': 7653943821, 'f_name': 'nia', 'l_name': 'ibrahim', 'email':
'nii2546@psu.edu', 'major': 'art history'}, {'id': 7272927795, 'f_name': 'kai', 'l_name': 'khan', 'email':
'kak4924@psu.edu', 'major': 'business administration'}, {'id': 2574473299, 'f_name': 'jin', 'l_name': 'johansson',
'email': 'jij2168@psu.edu', 'major': 'mathematics'}, {'id': 5664624437, 'f_name': 'aarav', 'l_name': 'moreau', 'email':
'aam5079@psu.edu', 'major': 'computer science'}, {'id': 2867291114, 'f_name': 'fatima', 'l_name': 'müller', 'email':
'fam8686@psu.edu', 'major': 'anthropology'}, {'id': 9585288517, 'f_name': 'marta', 'l_name': 'nguyen', 'email':
'man9283@psu.edu', 'major': 'history'}, {'id': 9658259626, 'f_name': 'diego', 'l_name': 'fernandez', 'email':
'dif5195@psu.edu', 'major': 'biology'}, {'id': 8232811678, 'f_name': 'jin', 'l_name': 'chen', 'email': 'jic1595@psu.edu',
'major': 'graphic design'}, {'id': 9311432149, 'f_name': 'anya', 'l_name': 'kim', 'email': 'ank4917@psu.edu', 'major':
'mathematics'}, {'id': 9929873755, 'f_name': 'yelena', 'l_name': 'müller', 'email': 'yem8680@psu.edu', 'major': 'english
literature'}]
sort params: ['id', 'f_name', 'l_name']

        Sort Parameter: id

Sort Algorithm: insertion took 6.399999999989747e-08
Sort Algorithm: selection took 5.100000000007876e-08
Sort Algorithm: bubble took 5.0000000000050006e-08
Sort Algorithm: merge took 8.599999999997499e-08

        Sort Parameter: f_name

Sort Algorithm: insertion took 5.0000000000050006e-08
Sort Algorithm: selection took 5.100000000007876e-08
Sort Algorithm: bubble took 5.0000000000050006e-08
Sort Algorithm: merge took 8.099999999983121e-08

        Sort Parameter: l_name

Sort Algorithm: insertion took 4.900000000002125e-08
Sort Algorithm: selection took 5.0000000000050006e-08
Sort Algorithm: bubble took 5.0000000000050006e-08
Sort Algorithm: merge took 8.199999999985997e-08

Times for Unsorted Data: {'insertion': {'id': 6.399999999989747e-08, 'f_name': 5.0000000000050006e-08, 'l_name':
4.900000000002125e-08}, 'selection': {'id': 5.100000000007876e-08, 'f_name': 5.100000000007876e-08, 'l_name':
5.0000000000050006e-08}, 'bubble': {'id': 5.0000000000050006e-08, 'f_name': 5.0000000000050006e-08, 'l_name':
5.0000000000050006e-08}, 'merge': {'id': 8.599999999997499e-08, 'f_name': 8.099999999983121e-08, 'l_name':
8.199999999985997e-08}}


dict_items([('id', <__main__.studentDb object at 0x000001F31D0CAD90>), ('f_name', <__main__.studentDb object at
0x000001F31D0CACD0>), ('l_name', <__main__.studentDb object at 0x000001F31D0C4FD0>)])

Algorithm performance over sorts by parameter times {'id': {'insertion': 7.000000000034757e-08, 'selection':
```

```
5.1999999999982996e-08, 'bubble': 4.7000000000241296e-08, 'merge': 8.299999999961116e-08}, 'f_name': {'insertion':
1.0200000000015752e-07, 'selection': 5.4999999999916226e-08, 'bubble': 4.999999999977245e-08, 'merge': 8.400000000019503e-
08}, 'l_name': {'insertion': 9.599999999998499e-08, 'selection': 5.599999999994498e-08, 'bubble': 4.90000000000298806e-08,
'merge': 8.799999999975494e-08}}

Process finished with exit code 0
```

Analysis

This lab exercise was a practical review of sorting and searching algorithms. It highlighted that for small datasets, simpler algorithms like selection sort can outperform more complex ones like merge sort, which can be slower due to the overhead of recursive calls. This observation is crucial when considering the size of the dataset for algorithm selection. The exercise also emphasized the importance of real-world testing over theoretical analysis. By measuring actual performance, we can better understand how these algorithms work in practice. Additionally, the comparison between binary and linear search showed the efficiency trade-offs depending on the data structure. Overall, the exercise reinforced the idea that the best algorithm choice depends on the specific scenario, including factors like data size and structure.

Tabulating the data made some of these important facts quite clear. Firstly, under best-case conditions (sorted data), bubble sort usualyl came out on top, this was due to the face that the internal while-loop was essentially funcitoning as a conditional requiring o(1) time, leading to o(n) performance. It was also interesting to see that merge-sort is not a great option for small sets of data. This is due to the practical cost and computation time of creating new data frames each time a recursive frame is added to the stack. My particular implementation also suffers from the issue of splicing python lists, which in reality copies data into a whole new list, another impractical component.