CEIS295 Data Structures and Algorithms

Developer: Roger Burns II

Introduction

Businesses work with Big Data and often have millions of records. Data must be available and able to be accessed quickly. How rapidly this data can be retrieved is a direct result on the algorithm used. The speed of the application can be fast, slow or “omg this is awful”. This project explores real-world scenarios that businesses face and tests the algorithm speeds that were used. The results are then analyzed to give the best choice.

Measure the speed of an ArrayList Data Structure

Scenario 1: Printer Queue or Service Queue
Scenario 2: Service Center
Scenario 3: Call Center

TimeProcess.py Python Code

Tested the following process speed:

- Display “Hello Everyone” one thousand times

Code:

```
#Name: Roger Burns
#Date:  9/4/2021
import time    # use time library to time the code executions
# get current time before the process
start_time = time.time()
# run the process
for i in range(1000):
print( "Hello Everyone!" )
# get current time after the process
end_time = time.time()
# subtract start time from end time to get time used by process
total_time = end_time - start_time
# Show the result.  Note: .6f means “show six decimal places”
print("\nSeconds run 1000 times: {:.6f}".format(total_time))
```

Client.py Python Code

- Attributes
  - __client_id
  - __first_name
  - __last_name
  - __phone
  - __email
- Behaviors
  - __eq__
  - __str__
- Getters and Setters

Code:

```
# Roger Burns
# 9/4/2021

class Client:
    def __init__(self, client_id=0, first_name="Unknown",
last_name="Unknown", phone="Unknown", email="Unknown"):
        self.__client_id = client_id
        self.__first_name = first_name
        self.__last_name = last_name
        self.__phone = phone
        self.__email = email

    #classes that compare objects must implement __eq__ and
__lt__ methods
    # __lt__ means "less than" and returns a boolean
    #__eq__ means "equals" and returns a boolean
    def __lt__(self, other):
        return self.__client_id < other.__client_id

    def __eq__(self, other):
        return self.__client_id == other.__client_id

    # __str__() method is automatically called when you print the
object
    def __str__(self):
        return(str(self.__client_id) + ", " + self.__last_name + ", " +
self.__first_name)

    #getters and setters
    def get_client_id(self):
        return self.__client_id
```

```
def set_client_id(self, client_id):
    self.__client_id = client_id

def get_first_name(self):
    return self.__first_name

def set_first_name(self, first_name):
    self.__first_name = first_name

def get_last_name(self):
    return self.__last_name

def set_last_name(self, last_name):
    self.__last_name = last_name

def get_phone(self):
    return self.__phone

def set_phone(self, phone):
    self.__phone = phone

def get_email(self):
    return self.__email

def set_email(self, email):
    self.__email = email
```

ArrayListActualSpeed.py Python Objective

Read records into application

Tested the following process speeds:

Added records to beginning of data structure

Added records to end of data structure

Added records to middle of data structure

Retrieved records from beginning of data structure

Retrieved records from end of data structure

Retrieved records from middle of data structure

ArrayListActualSpeed.py Python Code

```
# Roger Burns
# 9/4/2021

from ArrayList import ArrayList
from Client import Client
from Quicksort import Quicksort
from datetime import date
import time     #times code execution
import random   # generates random nums
import sys      #terminates the application early

# display date and name in output
print("Name: ", "Roger Burns")
print("Date: ", date.today())
print()

#create a list
clients = []

#read from ClientData.csv
#into Client objects and place Client objects into the list
input_file_name = 'ClientData.csv'
with open(input_file_name) as infile:
    for line in infile:
        #split the line based on the commas
        s = line.split(',')
        client_id =  int( s[0] ) #convert str to int
        f_name = s[1]
        l_name = s[2]
        phone = s[3]
        email = s[4]
        # create client object using data from the file
        clt = Client( client_id, f_name, l_name, phone, email)
        # add the client object to the list
        clients.append(clt)

    #sort the clients list
    Quicksort.sort(clients)

    #how many client objects do we have?
    num_records = len(clients)

    #create an array list object
    my_array_list = ArrayList()

    #Scenario 1: Printer Queue or Call Queue
    section_title = "Scenario: Printer Queue or Call Queue"
    print(section_title)
    print("-" * len(section_title))

    #how long does it take to add the client records to the ArrayList?
    start_time = time.time()

    for i in range(num_records):
        my_array_list.append(clients[i])

    end_time = time.time()
    total_time = end_time - start_time
    print("Seconds to add records: {:.6f}".format(total_time))

    #how long does it take to remove records from the front of the ArrayList?
    start_time = time.time()

    for i in range(num_records):
        my_array_list.remove_at(0)

    end_time=time.time()
    total_time = end_time - start_time
    print("Seconds to remove records from front: {:.6f}".format(total_time))

    #Scenario 2: Customer Service Center
    answer = input("Continue (y/n)?")
    if answer.lower() != "y":
        sys.exit()#exit app

    section_title = "Scenario: Customer Service Center"
    print(section_title)
    print("-" * len(section_title))

    #add clients to the ArrayList
    for i in range(num_records):
        my_array_list.append(clients[i])

    #how long does it take to randomly display 1000 records?
    start_time = time.time()

    for i in range(1000):
        smallest_id = 100001
        largest_id = smallest_id + num_records
        random_num = random.randint(smallest_id, largest_id)
        #print( my_array_list.search(Client(random_num)))
        print(?
my_array_list.search_sorted(Client(random_num)))

    #remove random records
    for i in range(1000):
        smallest_id = 100001
        largest_id = smallest_id + num_records
        random_num = random.randint(smallest_id, largest_id)
        print( my_array_list.search(Client(random_num)))


    end_time = time.time()
    total_time = end_time - start_time
    print("Seconds to add, display and remove records: {:.6f}".format(total_time))
```

Screenshot of Running Code

```
In [49]: runfile('C:/Users/there/OneDrive/Desktop/CEIS 295/Week 1 Project/ArrayListActualSpeed.py', wdir='C:/
Users/there/OneDrive/Desktop/CEIS 295/Week 1 Project')
Reloaded modules: ArrayList, Client, Quicksort
Name:  Roger Burns
Date:  2021-09-04

Scenario: Printer Queue or Call Queue
-------------------------------------
Seconds to add records: 0.005983
Seconds to remove records from front: 4.750297

Continue (y/n)?
```

Table of Speeds

Table of Real World Speeds		
	Array List (unsorted)	**Array List (sorted)**
Scenario: Printer or Call or Service Queue		
Add many records to end of data structure	0.003989	0.003988
Pull all records off front of data structure	4.636598	5.023567
Scenario: Customer Service Center		
Display random records	1.180177	0.063829
Scenario: Call Center		
Add some records, display random records, remove random records	2.145808	1.259557

- Printer Queue or Call Queue or Service Queue
  - Added many records to end of data structure
  - Pulled all records off front of data structure
- Customer Service Center
  - Displayed random records
- Call Center
  - Added some records
  - Displayed random records
  - Removed some records

LinkedListActualSpeed.py Python Objective

Read records into application

Tested the following process speeds:

Added records to beginning of data structure

Added records to end of data structure

Added records to middle of data structure

Retrieved records from beginning of data structure

Retrieved records from end of data structure

Retrieved records from middle of data structure

LinkedListActualSpeed.py Python Code

```
# Roger Burns
# 9/4/2021

from LinkedList import LinkedList
from Client import Client
from datetime import date
import time     #times code execution
import random   # generates random nums
import sys      #terminates the application early

# display date and name in output
print("Name: ", "Roger Burns")
print("Date: ", date.today())
print()

#create a list
clients = []

#read from ClientData.csv
#into Client objects and place Client objects into the list
input_file_name = 'ClientData.csv'
with open(input_file_name) as infile:
    for line in infile:
        #split the line based on the commas
        s = line.split(',')
        client_id =  int( s[0] ) #convert str to int
        f_name = s[1]
        l_name = s[2]
        phone = s[3]
        email = s[4]
        # create client object using data from the file
        clt = Client( client_id, f_name, l_name, phone, email)
        # add the client object to the list
        clients.append(clt)

    #how many client objects do we have?
    num_records = len(clients)

    #create an  Linkedlist object
    my_linked_list = LinkedList()

    #Scenario 1: Printer Queue or Call Queue
    section_title = "Scenario: Printer Queue or Call Queue"
    print(section_title)
    print("-" * len(section_title))

    #how long does it take to add the client records to the LinkedList?
    start_time = time.time()

    for i in range(num_records):
        my_linked_list.add_last(clients[i])

    end_time = time.time()
    total_time = end_time - start_time
    print("Seconds to add records: {:.6f}".format(total_time))

    #how long does it take to remove records from the front of the LinkedList?
    start_time = time.time()

    for i in range(num_records):
        my_linked_list.remove_first()

    end_time=time.time()
    total_time = end_time - start_time
    print("Seconds to remove records from front: {:.6f}".format(total_time))

    #Scenario 2: Customer Service Center
    answer = input("Continue (y/n)?")
    if answer.lower() != "y":
        sys.exit()#exit app

    section_title = "Scenario: Customer Service Center"
    print(section_title)
    print("-" * len(section_title))

    #add clients to the LinkedList
    for i in range(num_records):
        my_linked_list.add_last(clients[i])

    #how long does it take to randomly display 1000 records?
    start_time = time.time()

    for i in range(1000):
        smallest_id = 100001
        largest_id = smallest_id + num_records
        random_num = random.randint(smallest_id, largest_id)
        print( my_linked_list.search(Client(random_num)) )

    end_time = time.time()
    total_time = end_time - start_time
    print("Seconds to display random records: {:.6f}".format(total_time))

    #Scenario 3: Call Center
    answer = input("Continue (y/n)?")
    if answer.lower() != "y":
        sys.exit()#exit app

    section_title = "Scenario: Call Center"
    print(section_title)
    print("-" * len(section_title))

    #how long does it take to add records, randomly display 1000 records
    #and randomly remove 1000 records?
    start_time = time.time()

    #add records
    current_id = 100001 + num_records + 1
    for i in range(1000):

        my_linked_list.add_last(Client(current_id))
        current_id += 1

    #display random records
    for i in range(1000):
        smallest_id = 100001
        largest_id = smallest_id + num_records
        random_num = random.randint(smallest_id, largest_id)
        print( my_linked_list.search(Client(random_num)) )

    #remove random records
    for i in range(1000):
        smallest_id = 100001
        largest_id = smallest_id + num_records
        random_num = random.randint(smallest_id, largest_id)
        print( my_linked_list.search(Client(random_num)) )


    end_time = time.time()
    total_time = end_time - start_time
    print("Seconds to add, display and remove records: {:.6f}".format(total_time))
```

Screenshot of Running Code

```
In [4]: runfile('C:/Users/there/OneDrive/Desktop/CEIS 295/Week 2 Project/
LinkedListActualSpeed.py', wdir='C:/Users/there/OneDrive/Desktop/CEIS 295/Week 2
Project')
Reloaded modules: LinkedList, Node, Client
Name:  Roger Burns
Date:  2021-09-11

Scenario: Printer Queue or Call Queue
-----------------------------------
Seconds to add records: 0.005552
Seconds to remove records from front: 0.005045

Continue (y/n)?
```

Table of Speeds

Table of Real World Speeds			
	Array List (unsorted)	**Array List (sorted)**	**LinkedList**
Scenario: Printer or Call or Service Queue			
Add many records to end of data structure	0.003989	0.003988	0.008023
Pull all records off front of data structure	4.636598	5.023567	0.003077
Scenario: Customer Service Center			
Display random records	1.180177	0.063829	1.365759
Scenario: Call Center			
Add some records, display random records, remove random records	2.145808	1.259557	2.879958

- Printer Queue or Call Queue or Service Queue
  - Added many records to end of data structure
  - Pull all records off front of data structure
- Customer Service Center
  - Displayed random records
- Call Center
  - Added some records
  - Displayed random records
  - Removed some records

Call.py

Attributes

- call_id
- customer_name
- customer_phone
- call_date
- call_time

Behaviors

- __str__

Call.py
Python Code

```
# Developer Roger Burns
# 9-19-21
from time import strftime #used for current date/time
class Call:
def __init__(self, client_id = 0, client_name = "Unknown", client_phone = "Uknown"):
self.client_id = client_id
self.client_name = client_name
self.client_phone = client_phone
self.call_date = strftime("%m/%d/%y")
self.call_time = strftime("%H:%M")
# __str__() method is automatically called when you print the object
def __sztr__(self):
return str(self.client_id) + ", " + self.client_name + \
"\n\tPhone: " + self.client_phone + \
"\tDate/Time: " + self.call_date + " @ " + self.call_time
```

AutomaticCallDistributor.py Python Code

Simulation of ADT

- Asks the user how many times to repeat
- On each loop:
  - Randomly generates a number from 1 to 3
  - If 1, adds call to queue
  - If 2, removes call from queue
  - If 3, does nothing

```
# Developer Roger Burns
 # 9-19-21

from Queue import Queue
from Call import Call
from datetime import date
import time    # pauses app
import random  # gen random nums

# display author's name and date in tChe output
print("Name: ", "Roger Burns")
print("Date: ", date.today())
print()

#create a list
calls = []

#read call records into the list
input_file_name = "CallsData.csv"
with open(input_file_name) as infile:
    for line in infile:
        # split the line based on the commas
        s = line.split(',')
        client_id = int( s[0])
        client_name = s[1]
        client_phone = s[2]
        #create a call object based on the data from the line
        a_call = Call(client_id, client_name, client_phone)
        # add the call object to the list
        calls.append(a_call)

#queue object for calls
calls_waiting = Queue()
call_number = 0
```

```
#
#how long is the simulation?
seconds = int(input("How many seconds do you want to simulate?"))

#run the simulation for the given number of seconds
for i in range(seconds):
    print("-" * 40)
    time.sleep(1)
    random_event = random.randint(1,4)
    #do event based on the random number generated
    if random_event == 1 or random_event == 4:
        print("Call received. Caller added to queue.")
        calls_waiting.enqueue(calls[call_number])
        call_number +- 1
        print("\tNumber of calls waiting in queue: ", calls_waiting.get_length())
    elif random_event == 2:
        print("Call sent to representative for service. ")
        if calls_waiting.get_length() > 0:
            print("Caller Information: ")
            print(calls_waiting.dequeue())
        else:
            print("The call queue is empty.")
        print("\tNumber of calls waiting in queue: ",calls_waiting.get_length())
    else:
            print("Nothing happened during this second in time.")
            print("\tNumber of calls waiting in queue:
",calls_waiting.get_length())
            15
print("\nThe 'Automatic Call Distributor' simulation has completed.")
```

Screenshot of Running Code

```
In [27]: runfile('C:/Users/there/OneDrive/Desktop/CEIS 295/Week 3 Project/Au
OneDrive/Desktop/CEIS 295/Week 3 Project')
Reloaded modules: Queue, LinkedList, Node, Call
Name:  Roger Burns
Date:  2021-09-19


How many seconds do you want to simulate?5
----------------------------------------
Nothing happened during this second in time.
        Number of calls waiting in queue:  0
----------------------------------------
Nothing happened during this second in time.
        Number of calls waiting in queue:  0
----------------------------------------
Call sent to representative for service.
The call queue is empty.
        Number of calls waiting in queue:  0
----------------------------------------
Call received. Caller added to queue.
        Number of calls waiting in queue:  1
----------------------------------------
Nothing happened during this second in time.
        Number of calls waiting in queue:  1

The 'Automatic Call Distributor' simulation has completed.

In [28]:
```

Client.py Python Code

- Attributes
  - __client_id
  - __first_name
  - __last_name
  - __phone
  - __email
- Behaviors
  - __lt__
  - __le__
  - __eq__
  - __str__
- Getters and Setters

```
# Roger Burns
# 9/24/2021

class Client:
    def __init__(self, client_id=0, first_name="Unknown",
last_name="Unknown", phone="Unknown", email="Unknown"):
        self.__client_id = client_id
        self.__first_name = first_name
        self.__last_name = last_name
        self.__phone = phone
        self.__email = email

    #classes that compare objects must implement __eq__ and
__lt__ methods
    # __lt__ means "less than" and returns a boolean
    # __le__ means "less than or equal to" and returns a boolean
    #__eq__ means "equals" and returns a boolean
    def __lt__(self, other):
        this_full_name = self.__last_name + " " + self.__first_name
        other_full_name = other.__last_name + " " +
other.__first_name
        return this_full_name < other_full_name

    def __le__(self, other):
        this_full_name = self.__last_name + " " + self.__first_name
        other_full_name = other.__last_name + " " +
other.__first_name
        return this_full_name <= other_full_name

    def __eq__(self, other):
        return self.__client_id == other.__client_id

    # __str__() method is automatically called when you print the
object
    def __str__(self):
        return self.__last_name + ", " + self.__first_name

    #getters and setters
```

```
def get_client_id(self):
        return self.__client_id

    def set_client_id(self, client_id):
        self.__client_id = client_id

    def get_first_name(self):
        return self.__first_name

    def set_first_name(self, first_name):
        self.__first_name = first_name

    def get_last_name(self):
        return self.__last_name

    def set_last_name(self, last_name):
        self.__last_name = last_name

    def get_phone(self):
        return self.__phone

    def set_phone(self, phone):
        self.__phone = phone

    def get_email(self):
        return self.__email

    def set_email(self, email):
        self.__email = email
```

SortingActualSpeed.py Python Code

- Read records into application
- Tested the following sorting algorithm speeds:
  - Bubble Sort
  - Selection Sort
  - Insertion Sort
  - Shell Sort
  - Quicksort
  - Merge Sort

```
# Roger Burns
# 9/24/2021

from MergeSort import MergeSort
from Client import Client
from datetime import date
import time #times execution

# display date and name in output
print("Name: ", "Roger Burns")
print("Date: ", date.today())
print()

input_file_name = 'ClientData100.csv'
#input_file_name = 'ClientData1000.csv'
#input_file_name = 'ClientData10000.csv'
#input_file_name = 'ClientData100000.csv'

#create a list
clients = []

#read from ClientData.csv
#into Client objects and place Client objects into the list
with open(input_file_name) as infile:
    for line in infile:
        #split the line based on the commas
        s = line.split(',')
        client_id =  int( s[0] ) #convert str to int
        f_name = s[1]
        l_name = s[2]
        phone = s[3]
        email = s[4]
        # create client object using data from the file
        clt = Client( client_id, f_name, l_name, phone, email)
        # add the client object to the list
        clients.append(clt)

    #how many client objects do we have?
    num_records = len(clients)

    #Scenario 1: Sorting records from a data file
    section_title = "Scenario: Sorting " +  str(num_records) + "
Records"
    print(section_title)
    print("-" * len(section_title))

    #how long does it take to sort the client records?
    start_time = time.time()

    #call the static sort method in the class
    MergeSort.sort(clients)

    end_time = time.time()
    total_time = end_time - start_time
    print("Seconds to sort {0} records:
{1:.6f}".format(num_records,total_time))

    #display sorted list
    #for clt in clients:
     #  print(clt)
```

Screenshot of Running Code

```
In [27]: runfile('C:/Users/there/OneDrive/Desktop/CEIS 295/Week 4 Project/SortingActualSpeeds.py', wdir='C:/Users/there/OneDrive/
Desktop/CEIS 295/Week 4 Project')

Reloaded modules: MergeSort, Client
Name:  Roger Burns
Date:  2021-09-24

Scenario: Sorting 100000 Records
--------------------------------
Seconds to sort 100000 records: 1.559704
Reloaded modules: MergeSort, Client
Name:  Roger Burns
Date:  2021-09-24

Scenario: Sorting 100000 Records
--------------------------------
Seconds to sort 100000 records: 1.426004

In [28]: runfile('C:/Users/there/OneDrive/Desktop/CEIS 295/Week 4 Project/SortingActualSpeeds.py', wdir='C:/Users/there/OneDrive/
Desktop/CEIS 295/Week 4 Project')
```

Table of Sorting Speeds

Table of Real World Sorting Speeds				
	100 records	1,000 records	10,000 records	100,000 records
Bubble Sort	0.004004	0.368845	37.762831	over 1 hour
Selection Sort	0.002557	0.249901	27.010021	45 minutes
Insertion Sort	0.002284	0.219663	21.050717	~35 minutes
Shell Sort	0.008847	0.008795	0.184634	3.679336
Quicksort	0.006643	0.009204	0.107905	1.660007
Merge Sort	0.000865	0.021394	0.099046	1.426004

- Bubble Sort
- Selection Sort
- Insertion Sort
- Shell Sort
- Quicksort
- Merge Sort

Client.py Python Code

- Attributes
  - __client_id
  - __first_name
  - __last_name
  - __phone
  - __email
- Behaviors
  - __lt__
  - __eq__
  - __str__
- Getters and Setters

```
# Roger Burns
# 10/1/2021

class Client:
    def __init__(self, client_id=0, first_name="Unknown", last_name="Unknown",
                 phone="Unknown", email="Unknown"):
        self.__client_id = client_id
        self.__first_name = first_name
        self.__last_name = last_name
        self.__phone = phone
        self.__email = email

    #classes that compare objects must implement __eq__ and __lt__ methods
    # __lt__ means "less than" and returns a boolean
    # __le__ means "less than or equal to" and returns a boolean
    #__eq__ means "equals" and returns a boolean
    def __lt__(self, other):
        return self.__client_id < other.__client_id

    def __le__(self, other):
        return self.__client_id <= other.__client_id

    def __eq__(self, other):
        return self.__client_id == other.__client_id

    # __str__() method is automatically called when you print the object
    def __str__(self):
        return str(self.__client_id) + ", " + self.__last_name + "," + self.__first_name

    #getters and setters
    def get_client_id(self):
        return self.__client_id

    def set_client_id(self, client_id):
        self.__client_id = client_id
```

```
def get_first_name(self):
    return self.__first_name

def set_first_name(self, first_name):
    self.__first_name = first_name

def get_last_name(self):
    return self.__last_name

def set_last_name(self, last_name):
    self.__last_name = last_name

def get_phone(self):
    return self.__phone

def set_phone(self, phone):
    self.__phone = phone

def get_email(self):
    return self.__email

def set_email(self, email):
    self.__email = email
```

SearchingActualSpeed.py Python Code

- Read records into application
- Tested the following searching algorithm speeds:
  - Linear Search
  - Binary Search
- Tested these two algorithms against different size datasets
  - 100 records
  - 1,000 records
  - 10,000 records
  - 100,000 records

```
# Roger Burns
# 10/1/2021

from LinearSearch import LinearSearch
from BinarySearch import BinarySearch
from Quicksort import Quicksort
from Client import Client
from datetime import date
import random #used to generate random numbers
import time #times execution
import sys #exits app early

# display date and name in output
print("Name: ", "Roger Burns")
print("Date: ", date.today())
print()

#input_file_name = 'ClientData100.csv'
#input_file_name = 'ClientData1000.csv'
#input_file_name = 'ClientData10000.csv'
input_file_name = 'ClientData100000.csv'

#create a list
clients = []
#read from ClientData.csv
#into Client objects and place Client objects into the list
with open(input_file_name) as infile:
    for line in infile:
        #split the line based on the commas
        s = line.split(',')
        client_id =  int( s[0] ) #convert str to int
        f_name = s[1]
        l_name = s[2]
        phone = s[3]
        email = s[4]
        # create client object using data from the file
        clt = Client( client_id, f_name, l_name, phone, email)
```

```
# add the client object to the list
        clients.append(clt)

    #how many client objects do we have?
    num_records = len(clients)

    #Scenario 1: Searcg 1000 records from a data file
    section_title = "Scenario: Searching Through " +
str(num_records) + " Records"
    print(section_title)
    print("-" * len(section_title))

    #must sort data to do a binary search
    Quicksort.sort(clients)

    #start and end record numbers
    start_record = 100001
    end_record = start_record + num_records

    #how long does it take to sort the client records?
    start_time = time.time()

    #how long does it take to do a random binary search through the
records
    for i in range(1000):
        client_id = random.randint(start_record, end_record)
        clt = Client(client_id)
        result = BinarySearch.search(clients,clt)
        if result is None:
            print(clt, "was not found.")
        else:
            print(result)

    end_time = time.time()
    total_time = end_time - start_time
    print("Seconds to Kinear Search for 1000 records:
{:.6f}".format(total_time))
```

Screenshot of Running Code

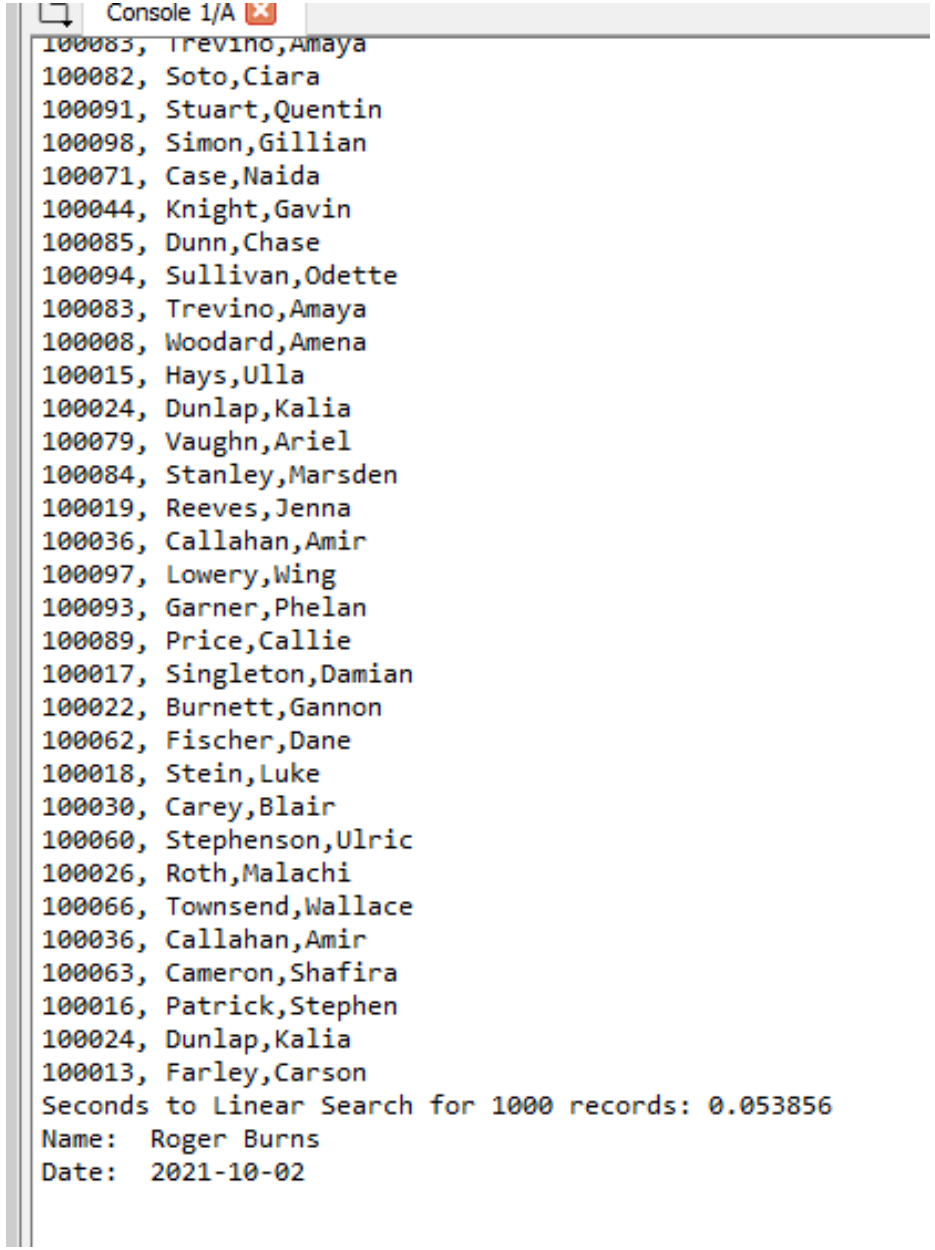

Table of Searching Speeds

Table of Real World Searching Speeds				
Data Size ------->	**100 records**	**1,000 records**	**10,000 records**	**100,000 records**
Linear Search for 1000 random records	0.092043	0.166991	1.213584	10.708908
Binary Search for 1000 random records	0.045034	0.042885	0.062101	0.049992

- Table showing the time that it takes for the following searching algorithms using different data sizes:
  - Linear Search
  - Binary Search

BinarySearchTreeActualSpeed.py Objective

Read records into application

Tested the following process speeds:

Added records to beginning of data structure

Added records to end of data structure

Added records to middle of data structure

Retrieved records from beginning of data structure

Retrieved records from end of data structure

Retrieved records from middle of data structure

BinarySearchTreeActualSpeed.py Python Code

Code:

```
# Roger Burns
# 10/6/2021

from BinarySearchTree import BinarySearchTree
from Client import Client
from datetime import date
import time
import random
import sys

#display developer's name n date
print("Name: ", "Roger Burns")
print("Date: ", date.today())
print()

#create a list
clients = []

# read records into the list
input_file_name = "ClientData.csv"
with open(input_file_name) as infile:
    for line in infile:
        #split the line based on the commas
        s = line.split(',')
        client_id = int(s[0])
        f_name = s[1]
        l_name = s[2]
        phone = s[3]
        email = s[4]
        # create client object using the data from the line
        clt = Client(client_id, f_name, l_name, phone, email)
        # add the client object to the list
        clients.append(clt)

# how many client objects do we have?
        num_records = len(clients)

#create the Binary Search Tree to test real-world speeds
        my_bst = BinarySearchTree()

# Scenario 1: Printer queue or call queue or service queue
section_title = "Scenario: Printer Queue or Call Queue or Service Queue"
print(section_title)
print("-" * len(section_title))

#how long does it take to add the client records to the BST?
start_time = time.time()

for i in range(num_records):
    my_bst.insert(clients[i])

end_time = time.time()
total_time = end_time - start_time
print("Seconds to add records: {0:.6f}".format(total_time))

# how long does it take to remove records from the front (smallest) of the binary search tree
start_time = time.time()

for i in range(num_records):
    my_bst.remove_minimum()

end_time = time.time()
total_time = end_time - start_time
print("Seconds to remove records: {0:.6f}".format(total_time))

# Scenario 2:  Service Queue
answer = input("Continue (y/n) ? ")
if answer.lower() != "y":
    sys.exit()

section_title = "Scenario: Service Queue"
print(section_title)
print("-" * len(section_title))

# add clients to the BST
for i in range(num_records):
    my_bst.insert(clients[i])

#how long does it take to randomly display 1000 client records to the BST?
start_time = time.time()

for i in range(1000):
    smallest_id = 100001
    largest_id = smallest_id + num_records
    random_num = random.randint(smallest_id, largest_id)
    print(my_bst.search(Client(random_num)))

end_time = time.time()
total_time = end_time - start_time
print("Seconds to display 1000 records: {0:.6f}".format(total_time))

# Scenario 3:  Call Center Queue
answer = input("Continue (y/n) ? ")
if answer.lower() != "y":
    sys.exit()

section_title = "Scenario: Call Center Queue"
print(section_title)
print("-" * len(section_title))

# add clients to the BST
for i in range(num_records):
    my_bst.insert(clients[i])

#how long does it take to add more client records to the BST?
# randomly display 1000 records, and randomly remove 1000 records from the bst?
start_time = time.time()

#add records
current_id = 100001 + num_records + 1
for i in range(1000):
    my_bst.insert(Client(current_id))
    current_id += 1

# display records
for i in range(1000):
    smallest_id = 100001
    largest_id = smallest_id + num_records
    random_num = random.randint(smallest_id, largest_id)
    print(my_bst.search(Client(random_num)))

# remove 1000 records
for i in range(1000):
    smallest_id = 100001
    largest_id = smallest_id + num_records
    random_num = random.randint(smallest_id, largest_id)
    my_bst.remove(Client(random_num))
    end_time = time.time()
total_time = end_time - start_time
print("Seconds to add records,")
print("Display 1000 random records,")
print("Remove 1000 random records: {0:.6f}".format(total_time))
```

Screenshot of Running Code

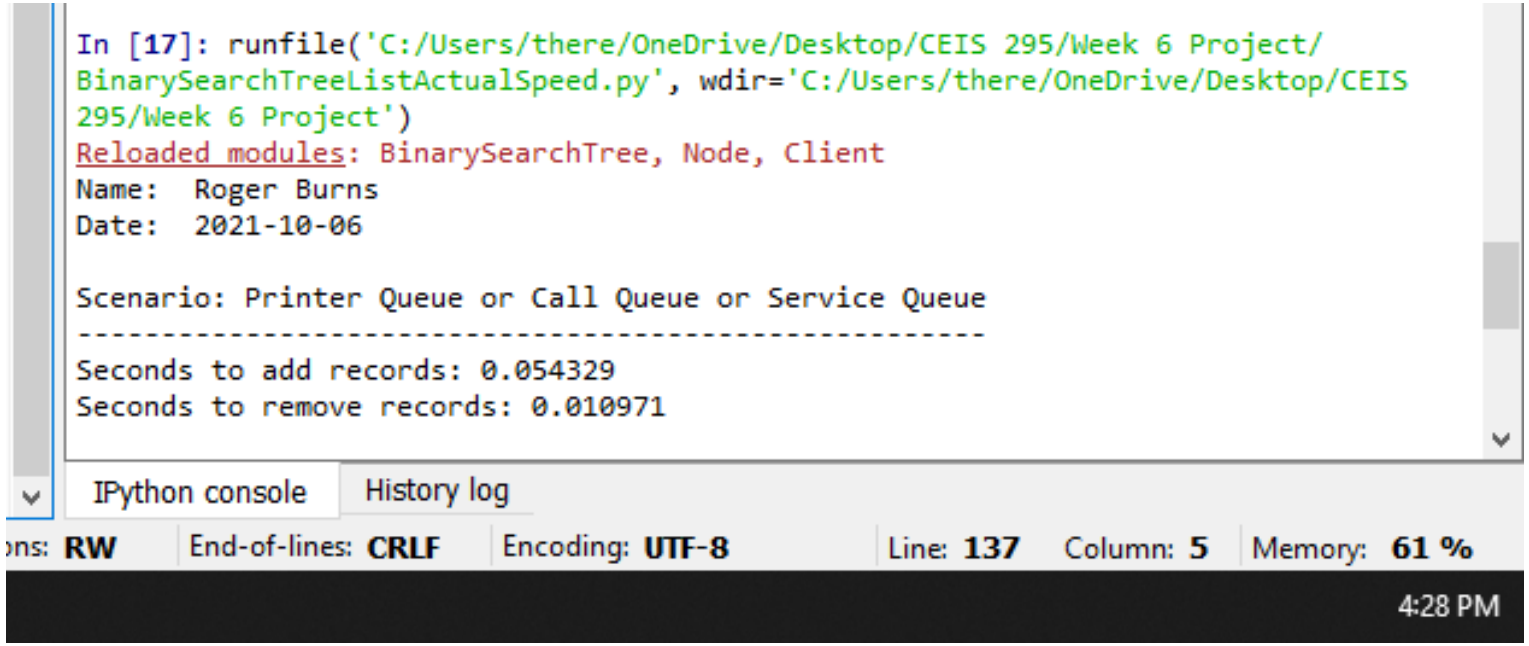

Table of Speeds

Table of Real-World Speeds

	Array List (unsorted)	Array List (sorted)	LinkedList	Binary Search Tree
Scenario: Printer or Call or Service Queue				
Add many records to end of data structure	0.003989	0.003988	0.008023	0.048870
Pull all records off front of data structure	4.636598	5.023567	0.003077	0.009978
Scenario: Customer Service Center				
Display random records	1.180177	0.063829	1.365759	0.024900
Scenario: Call Center				
Add some records, display random records, remove random records	2.145808	1.259557	2.879958	0.253335

- Printer Queue or Call Queue or Service Queue
  - Added many records to end of data structure
  - Pulled all records off front of data structure
- Customer Service Center
  - Displayed random records
- Call Center
  - Added some records
  - Displayed random records
  - Removed some records

Analysis

Data Structures have advantages and disadvantages.

ArrayList

- Fast for accessing data
- Slow at removing data
- Scenario to use it: Service Center

Linked List

- Fast for adding and removing data on the ends
- Slow for accessing data from the middle
- Scenario to use it: Printer or Service Queue

Binary Search Tree

- Reasonably fast at adding and removing data that is balanced
- Good speed for accessing data if the tree is balanced
- Scenario to use it: Call Center

Career Skills

Communication

- PowerPoint Presentations
- Excel tables used for analysis

Programming with Python

Troubleshooting errors within the code

Analysis

- Reviewing tables
- Determining best data structure for a given scenario

Conclusion

Project covered fundamental topics in Data Structures.

Project covered Algorithm speeds and analysis

Creating the project provided hands on learning

Project provided practice:

- Python programming
- Object-oriented programming
    - Importing classes
    - Client class