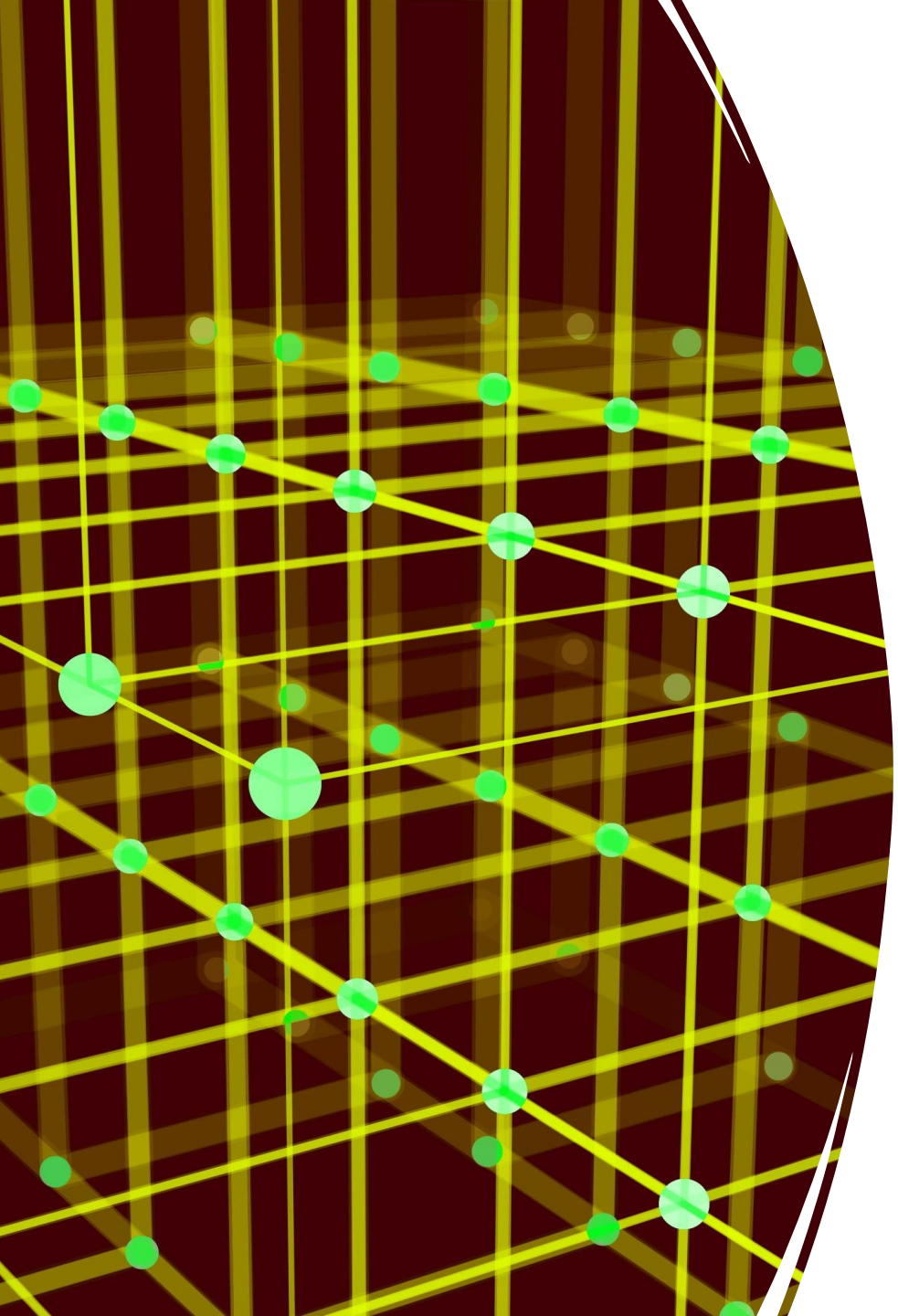


CEIS295
Data Structures and Algorithms

DeVry University

Session 1: January 20223

Glenn Delostrico

Introduction

- Corporate applications work with millions of records
- Data must be accessed quickly to provide good customer service
- Algorithms can cause application speeds to be fast, slow or ‘horrible’.
- This project measures real-world algorithm speeds
- Speeds are analyzed to determine best scenario for data structures

Python Code – TimeProcess.py

Python Code - Client.py

Python Code – ArrayListActualSpeed.py

Screenshot of Running Code

Table of Speeds

Module 1: Speed Test for ADT using Queue

TimeProcess.py Python Code

Paste your TimeProcess.py Python code.

Include the following processes:

- Test the following process speed:
  - Display “Hello Everyone” one thousand times

```
Hello Everyone!
Hello Everyone!
Hello Everyone!
Hello Everyone!
Hello Everyone!
Hello Everyone!
Hello Everyone!
Hello Everyone!
Hello Everyone!
Hello Everyone!
Hello Everyone!
Hello Everyone!
Hello Everyone!
Hello Everyone!
Hello Everyone!
Hello Everyone!
Hello Everyone!

Seconds run 1000 times: 0.076061
PS D:\Google Drive\glenndelostrico\Personal Files\Education\DeVry\2023\Session 1\CEIS295\Week 1>
```

```python
#Name: Glenn Delostrico
#Date:  1/6/2023
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

Paste your Client.py Python code in the box on the right side.

Include the following attributes and behaviors:

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

```
#Name: Glenn Delostrico
#Date:  1/6/2023

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 it must return a boolean
    # __eq__ means "equals" and it must return 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
```

ArrayListActualSpeed.py Python Code

Paste your Python code. Include the following processes:

- Read records into application
- Test the following process speeds:
  - Add records to beginning of data structure
  - Add records to end of data structure
  - Add records to middle of data structure
  - Retrieve records from beginning of data structure
  - Retrieve records from end of data structure
  - Retrieve records from middle of data structure

```
#Name: Glenn Delostrico
#Date:  1/6/2023

from ArrayList import ArrayList
from Client import Client
from Quicksort import Quicksort
from datetime import date
import time
import random
import sys

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

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

Screenshot of Running Code

Run your ArrayListActualSpeed.py code and take a screenshot of the output. The output screenshot must show your name and the date to be accepted.

Please note. We are looking to avoid plagiarism. As a result, if the running code does not show your name and the current date, then this Course Project will be given a zero until you submit a PowerPoint that shows your name and the current date in the output screenshot.

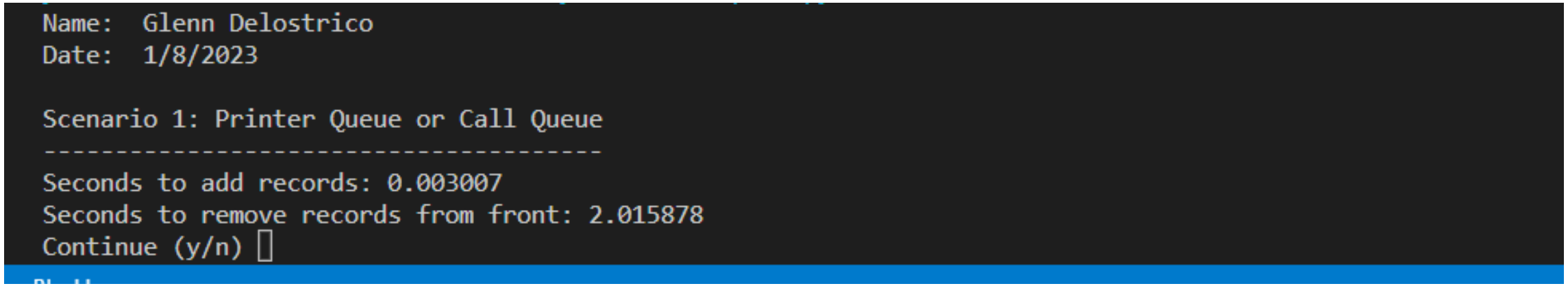


Table of Speeds

Create a table showing the time that it takes for the following scenarios:

- Printer Queue or Call Queue or Service Queue
  - Add many records to end of data structure
  - Pull all records off front of data structure
- Customer Service Center
  - Display random records
- Call Center
  - Add some records
  - Display random records
  - Remove some records

Table of Real World Speeds		
	ArrayList (Unsorted)	ArrayList (sorted data)
Scenario 1: Printer Queuee or Call Queue or Service Queue		
Add many records to end of data structure:	0.001498	0.002
Pull all records off front of data structure	1.93102	2.048328
Scenario 2: Customer Service Center		
Display random records	0.460364	0.090753
Scenario 3: Call Center		
Add some records, display random records, remove random records	0.92214	0.837739

Python Code – Call.py

Python Code - AutomaticCallDistributor.py

Screenshot of Running Code

Module 3: Speed Test for ADT using Queue

Call.py
Python Code

Paste your Call.py Python code in the box on the right side.

Include the following attributes and behaviors:

- Attributes
  - call_id
  - customer_name
  - customer_phone
  - call_date
  - call_time
- Behaviors
  - __str__

```
# Name: Glenn Delostrico
# Date: 1/25/2023

from time import strftime # used for current date and time

class Call:
    def __init__(self, client_id=0, client_name="Unknown", client_phone="Unknown"):
        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")

    def __str__(self):
        return str(self.client_id) + ", " + self.client_name + \
            "\n\tPhone: " + self.client_phone + \
```

AutomaticCallDistributor.py Python Code

Paste your Python code. Include the following processes:

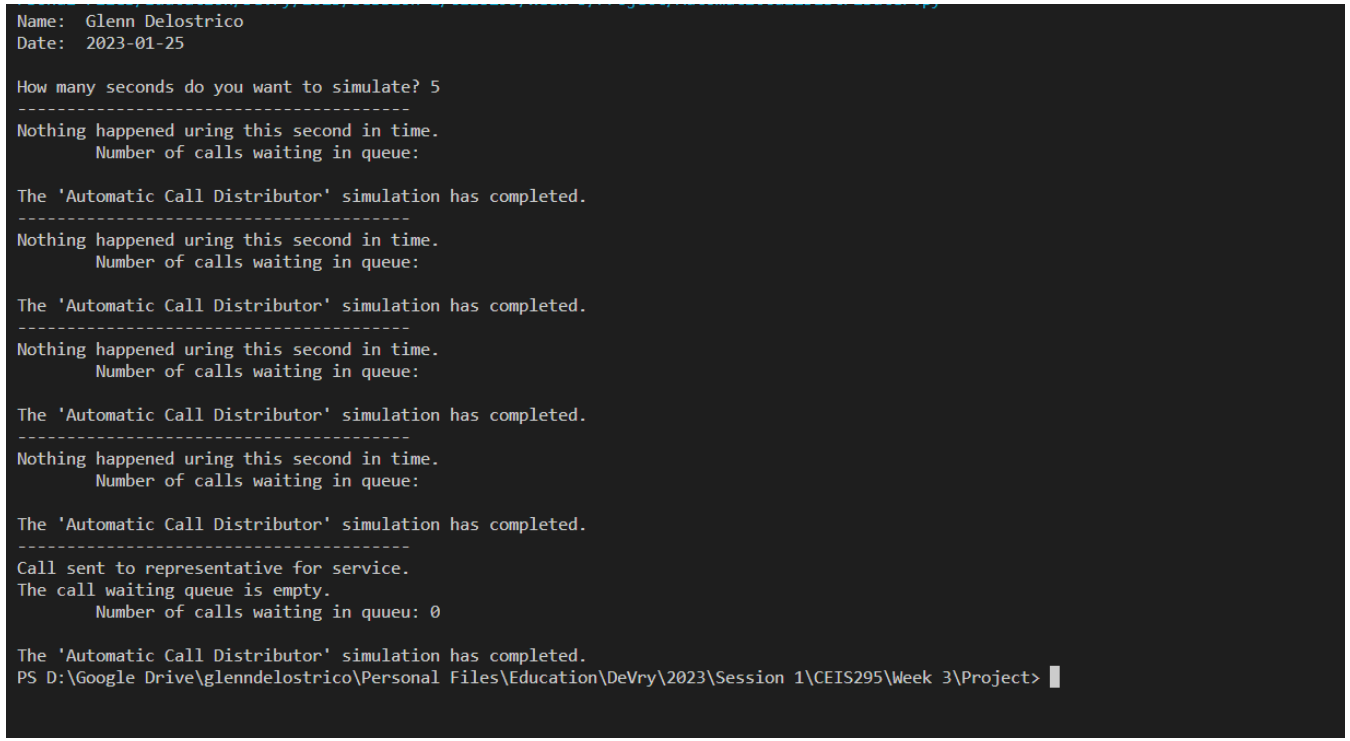
- Simulation of ADT
  - Ask user how many times to repeat
  - On each loop:
    - Randomly generate a number from 1 to 3
    - If 1, add call to queue
    - If 2, remove call from queue
    - If 3, do nothing

```
# Name: Glenn Delostrico
# Date: 1/25/2023

from Queue import Queue
from Call import Call
from datetime import date
import time
import random

print("Name: ", "Glenn Delostrico")
print("Date: ", date.today())
print()

calls = []

# read call record
input_file_name = 'ClientData.csv'
with open(input_file_name) as infile:
    for line in infile:
        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)
        # append call object to the list
        calls.append(a_call)

# Que object for calls
calls_waiting = Queue()
call_number = 0

seconds = int( input ("How many seconds do you want to simulate? "))

for i in range(seconds):
    print("-" * 40)
    time.sleep(2)
    random_event = random.randint(1,3)
    # do event from random number
```

Screenshot of Running Code

Run your AutomaticCallDistributor.py code and take a screenshot of the output. The output screenshot must show your name and the date to be accepted.

Please note. We are looking to avoid plagiarism. As a result, if the running code does not show your name and the current date, then this Course Project will be given a zero until you submit a PowerPoint that shows your name and the current date in the output screenshot.



Python Code – Client.py

Python Code - SortingActualSpeed.py

Screenshot of Running Code

Table of Sorting Speeds

Module 4: Speed Test for Sorting Algorithms

Client.py
Python Code

Paste your Client.py Python code in the box on the right side.

Include the following attributes and behaviors:

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

Code:
```
#Name: Delostrico, Glenn
#Date:  1/29/2023

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 it must return a boolean
    # __eq__ means "equals" and it must return a boolean
    # __le__ means "less than or equal to" and it must return 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
        # return self.__client_id < other.__client_id sort by client id

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

    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

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

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

SortingActualSpeed.py Python Code

Paste your Python code. Include the following processes:

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

```
# Name: Glenn Delostrico
# Date: 1/29/2023

from BubbleSort import BubbleSort
from SelectionSort import SelectionSort
from InsertionSort import InsertionSort
from ShellSort import ShellSort
from Quicksort import Quicksort
from MergeSort import MergeSort
from Client import Client
from datetime import date
import time

print("Name: ", "Glenn Delostrico")
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 call record
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 a call object based on the data from the line
        clt = Client(client_id, f_name, l_name, phone, email)
        # append call object to the list
        clients.append(clt)
```

Screenshot of Running Code

Run your SortingSpeed.py code and take a screenshot of the output. The output screenshot must show your name and the date to be accepted.

Please note. We are looking to avoid plagiarism. As a result, if the running code does not show your name and the current date, then this Course Project will be given a zero until you submit a PowerPoint that shows your name and the current date in the output screenshot.

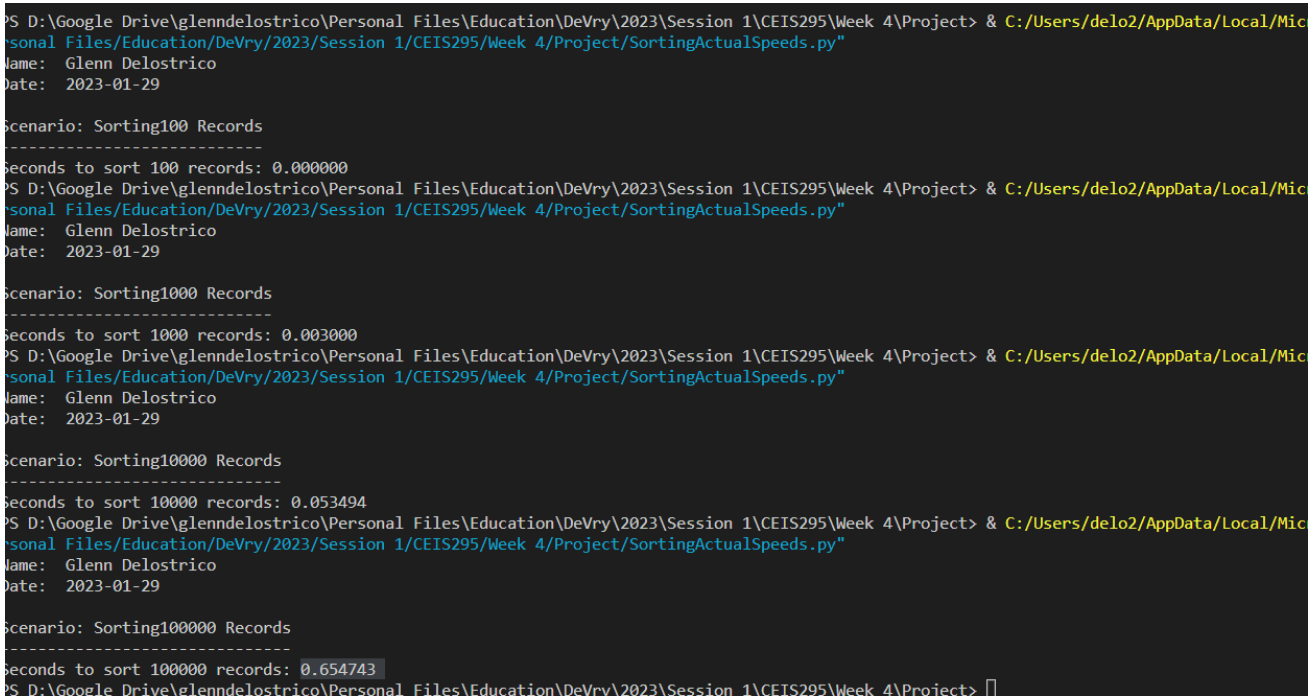


Table of Sorting Speeds

Create a table showing the time that it takes for the following sorting algorithms using different data sizes:

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

Table of Real World Sorting Speeds				
	100 Records	1,000 Record	10,000 Recor	100,000 records
Bubble Sort	0.001147	0.146502	14.061904	1406.1904
Selection Sort	0.001001	0.095047	9.456682	945.6682
Insertion Sort	0.0005	0.069533	7.032623	703.2623
Shell Sort	0.000499	0.003998	0.079421	7.9421
Quicksort	0.0005	0.006511	0.043555	4.3555
Merge Sort	0	0.003	0.053494	0.654743

Python Code – Client.py

Python Code - SearchingActualSpeed.py

Screenshot of Running Code

Module 5: Speed Test for Searching Algorithms

Client.py
Python Code

Paste your Client.py Python code in the box on the right side.

Include the following attributes and behaviors:

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

```
#Name: Delostrico, Glenn
#Date:  1/29/2023

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 it must return a boolean
    # __eq__ means "equals" and it must return a boolean
    # __le__ means "less than or equal to" and it must return 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
        # return self.__client_id < other.__client_id sort by client id
        return self.__client_id < other.__client_id

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

    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
        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
        return str(self.__client_id) + ", " + self.__last_name + ", " +
self.__first_name
        # return str(self.__client_id) + ", " + self.__last_name + ", " +
self.__first_name // sort by client id
```

SearchingActualSpeed.py Python Code

Paste your Python code. Include the following processes:

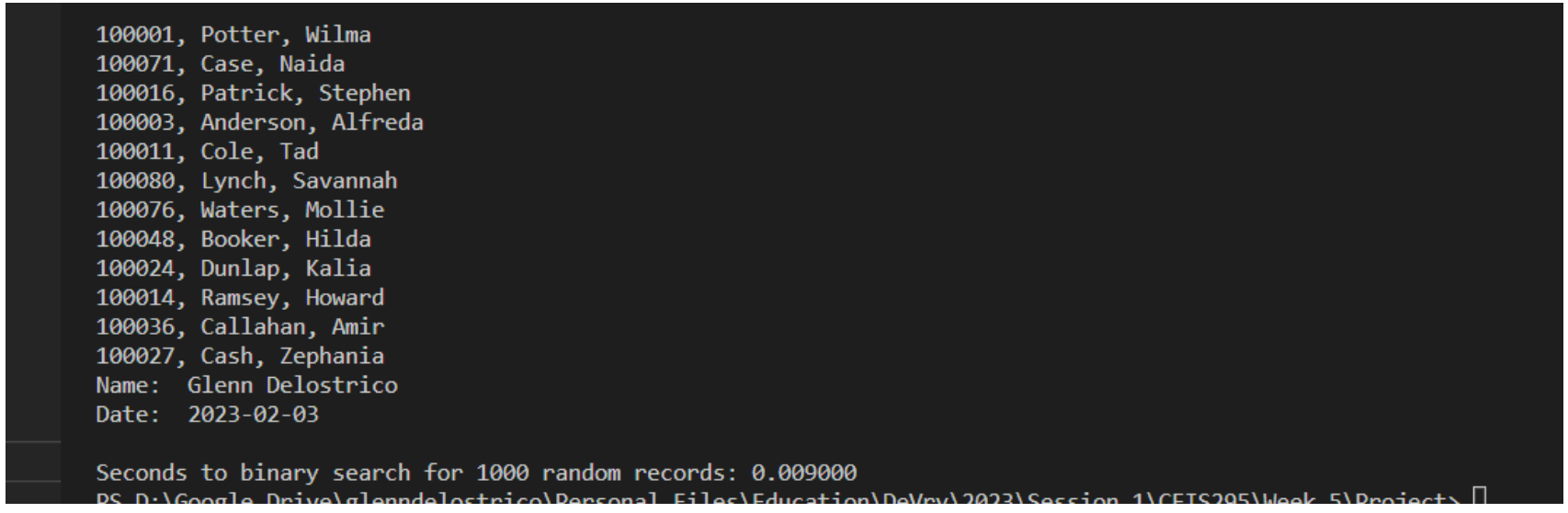
- Read records into application
- Test the following searching algorithm speeds:
  - Linear Search
  - Binary Search
- Test these two algorithms against different size datasets
  - 100 records
  - 1,000 records
  - 10,000 records
  - 100,000 records

```
# Name: Glenn Delostrico
# Date: 2/3/2023

from LinearSearch import LinearSearch
from BinarySearch import BinarySearch
from Quicksort import Quicksort
from Client import Client
from datetime import date
import time
import sys
import random

print("Name: ", "Glenn Delostrico")
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 call record
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 a call object based on the data from the line
        clt = Client(client_id, f_name, l_name, phone, email)
        # append call object to the list
        clients.append(clt)
```

Screenshot of Running Code

Run your SearchingSpeed.py code and take a screenshot of the output. The output screenshot must show your name and the date to be accepted.

Please note. We are looking to avoid plagiarism. As a result, if the running code does not show your name and the current date, then this Course Project will be given a zero until you submit a PowerPoint that shows your name and the current date in the output screenshot.



Python Code – BinarySearchTreeActualSpeed.py

Screenshot of Running Code

Table of Speeds

Module 6: Speed Test for Binary Search Tree

BinarySearchTreeActualSpeed.py Python Code

Paste your Python code. Include the following processes:

- Read records into application
- Test the following process speeds:
  - Add records to beginning of data structure
  - Add records to end of data structure
  - Add records to middle of data structure
  - Retrieve records from beginning of data structure
  - Retrieve records from end of data structure
  - Retrieve records from middle of data structure

```
# Name: Glenn Delostrico
# Date: 2/7/2023

from BinarySearchTree import BinarySearchTree
from Client import Client

from datetime import date
import time
import random
import sys

print("Name:", "Glenn Delostrico")
print("Date:", date.today())
print()

clients = []

input_file_name = 'ClientData.csv'
# read call record
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 a call object based on the data from the line
        clt = Client(client_id, f_name, l_name, phone, email)
        # append call object to the list
        clients.append(clt)

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

# create Binary Search Tree for testing
my_bst = BinarySearchTree()
```

Screenshot of Running Code

Run your BinarySearchTreeActualSpeed.py code and take a screenshot of the output. The output screenshot must show your name and the date to be accepted.

Please note. We are looking to avoid plagiarism. As a result, if the running code does not show your name and the current date, then this Course Project will be given a zero until you submit a PowerPoint that shows your name and the current date in the output screenshot.

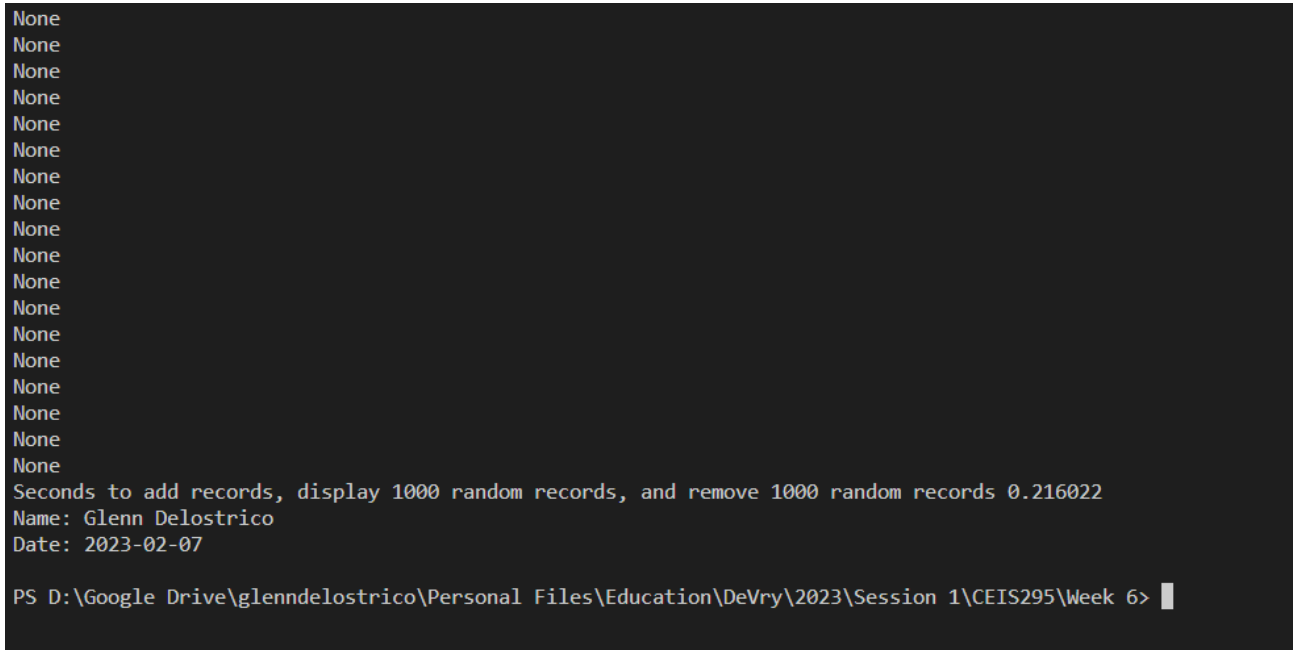


Table of Speeds

Update the table from last week that shows the times for the ArrayList and LinkedList. Add a new column and show the times for the BinarySearchTree for comparison.

- Printer Queue or Call Queue or Service Queue
  - Add many records to end of data structure
  - Pull all records off front of data structure
- Customer Service Center
  - Display random records
- Call Center
  - Add some records
  - Display random records
  - Remove some records

Table of Real World Speeds

	ArrayList (Unsorted)	ArrayList (sorted data)	LinkedList (sorted data)	BinarySearchTree
Scenario 1: Printer Queuee or Call Queue or Service Queue				
Add many records to end of data structure:	0.001498	0.002	0.003504	0.026426
Pull all records off front of data structure	1.93102	2.048328	0.001512	0.005499
Scenario 2: Customer Service Center				
Display random records	0.460364	0.090753	0.603216	0.090237
Scenario 3: Call Center				
Add some records, display random records, remove random records	0.92214	0.837739	1.156755	0.220965

Final Analysis

- Data Structures have Pros and Cons
  - ArrayLists:
    - Fast for processing data.
    - Slow for removing data.
    - Scenario to use: Service Center
  - Linked List:
    - Fast at adding and removing data at the ends
    - Slow for adding and removing data at the middle.
    - Scenario to use: Printer or Service Queue
  - Binary Search Tree:
    - Reasonably fast for adding and removing data if tree is balanced.
    - Reasonably fast for accessing data if tree is balanced
    - Scenario to use: Call Center

Challenges

- Typos with Python Code
- Keeping project files in the same directory
- Learning techniques to measure algorithm speeds
- Determining best scenarios for data structure

Career Skills

- Communication
  - Excel Tables for analysis
  - PowerPoint presentation
- Programming with Python
- Troubleshooting Python code
- Analysis

Conclusion

- Project covered fundamental topics in Data Structure
- Project covered algorithm speeds and analysis
- Hands on learning opportunity
- Practice on:
  - Python programming
  - Object Oriented programming
    - Importing classes
    - Client class