Candidate name:	Adler Chua Yu Cheng
Centre number:	
Index number:	1301017
Programming language used:	Python 3.6.0

Question 1

Evidence 1 *Program code*

```
def ReadLog():
    file_handle = open("WEBLOG.txt")
    log_data = []
    for line in file_handle:
        name, date = line.strip().split("|")
        date = date.split(":")[0]
        for i in range(len(log_data)):
            if log_data[i][0] == name:
                for j in range(len(log_data[i][1])):
                    if log_data[i][1][j] == date:
                        break
                else:
                    log_data[i][1].append(date)
                break
        else:
            log_data.append([name, [date]])
    file_handle.close()
    return log_data
```

Evidence 2 *Program code*

```
def ProcessLog():
    file_handle = open("SUMMARY.txt", "w")
    log_data = ReadLog()
    for data in log_data:
        file_handle.write("{0:<20}{1}".format(data[0], ",".join(data[1]))
\
                          + "\n")
    file_handle.close()
```

Evidence 3 *Screenshot*

```
199.32.43.22          01/Jul/1995,02/Jul/1995,03/Jul/1995
abc.xom.com           02/Jul/1995,03/Jul/1995
fds.err.saa           02/Jul/1995
computing.com         01/Jul/1995,03/Jul/1995,05/Jul/1995
```

Evidence 4 *Program code*

```
def ProcessLog():
    file_handle = open("SUMMARY.txt", "w")
    log_data = ReadLog()
    for data in log_data:
        file_handle.write("{0:<20}{1}".format(data[0], ",".join(data[1]))
\
                          + "\n")
    file_handle.close()

    high = 0
    accessed_by = []
    for i in range(len(log_data)):
        if len(log_data[i][1]) > high:
            high = len(log_data[i][1])
            accessed_by = [log_data[i][0]]
        elif len(log_data[i][1]) == high:
            accessed_by.append(log_data[i][0])
    print("Highest frequency (days): ", high)
    print("Accessed by:")
    for name in accessed_by:
        print(name)
```

Evidence 5 *Screenshot*

```
>>> ProcessLog()
Highest frequency (days):  3
Accessed by:
199.32.43.22
computing.com
```

Question 2

Evidence 6 *Program code*

```
def get_choice():
    while True:
        choice = input("\n1. Read file\n"\
                       "2. Linear Search\n"\
                       "3. Binary Search\n"\
                       "4. End\n")
        if choice.isdigit():
            if int(choice) in range(1, 5):
                return int(choice)
        print("\nInvalid input. Please select an option from 1 to 4"\
              ", corresponding to the desired option.")

def menu():
    while True:
        choice = get_choice()
```

```
        if choice == 4:
            break
        elif choice == 1:
            pass
        else:
            try:
                file_data
            except NameError:
                print("Please read file data using option 1 first!")
            else:
                if choice == 2:
                    pass
                else: #choice = 3
                    pass
```

Evidence 7 *Program code*

#NOTE: Changes from previous parts of code are bolded.

```
def menu():
    while True:
        choice = get_choice()
        if choice == 4:
            break
        elif choice == 1:
            file_data = read_file()
        else:
            try:
                file_data
            except NameError:
                print("Please read file data using option 1 first!")
            else:
                if choice == 2:
                    pass
                else: #choice = 3
                    pass

def read_file(x):
    #x is either 1 or 2 and represents which file is to be opened
    #i.e.: x = 1 for CUPS-SOLD1.txt; x = 2 for CUPS-SOLD2.txt
    if x == 1:
       file_handle = open("CUPS-SOLD1.txt")
    elif x == 2:
        file_handle = open("CUPS-SOLD1.txt")
    else:
        raise ValueError
    file_data = file_handle.read().strip().split("\n")
    file_handle.close()
    return file_data
```

Evidence 8 *Program code Screenshots X 2*

#NOTE: Changes from previous parts of code are bolded.

```
def menu():
```

```
    while True:
        choice = get_choice()
        if choice == 4:
            break
        elif choice == 1:
            x = 1
            file_data = read_file(x)
        else:
            try:
                file_data
            except NameError:
                print("\nPlease read file data using option 1 first!")
            else:
                sales_figure = input("\nSales figure to find: ")
                if choice == 2: #linear search
                    found = LinearSearch(file_data, sales_figure)
                    print("Sales figure " + sales_figure + " was ", \
                          end = '')
                    if found == -1:
                        print("not ", end = '')
                    print("found in the file ", end = '')
                    if x == 1:
                        print("CUPS-SOLD1.txt")
                    else: #x = 2
                        print("CUPS-SOLD2.txt")
                else: #choice = 3 #binary search
                    pass

def LinearSearch(array, target):
    for i in range(len(array)):
        if array[i] == target:
            return 0
    return -1
```

```
>>> menu()

1. Read file
2. Linear Search
3. Binary Search
4. End
1

1. Read file
2. Linear Search
3. Binary Search
4. End
2

Sales figure to find: 167
Sales figure 167 was found in the file CUPS-SOLD1.txt

1. Read file
2. Linear Search
3. Binary Search
4. End
4

>>> menu()

1. Read file
2. Linear Search
3. Binary Search
4. End
1

1. Read file
2. Linear Search
3. Binary Search
4. End
2

Sales figure to find: 405
Sales figure 405 was not found in the file CUPS-SOLD1.txt

1. Read file
2. Linear Search
3. Binary Search
4. End
4
```

Evidence 9 *Program code Screenshots X 2*

#NOTE: Changes from previous parts of code are bolded.

```
def menu():
    while True:
        choice = get_choice()
```

```
        if choice == 4:
            break
        elif choice == 1:
            x = 1
            file_data = read_file(x)
        else:
            try:
                file_data
            except NameError:
                print("\nPlease read file data using option 1 first!")
            else:
                sales_figure = input("\nSales figure to find: ")
                if choice == 2: #linear search
                    found = LinearSearch(file_data, sales_figure)
                else: #choice = 3 #binary search
                    try:
                        sorted_file_data
                    except NameError:
                        sorted_file_data = quick_sort(file_data)
                        found = BinarySearch(sorted_file_data, \
                                             sales_figure)
                print("Sales figure " + sales_figure + " was ", \
                      end = '')
                if found == -1:
                    print("not ", end = '')
                print("found in the file ", end = '')
                if x == 1:
                    print("CUPS-SOLD1.txt")
                else: #x = 2
                    print("CUPS-SOLD2.txt")

def BinarySearch(array, target):
    low = 0
    high = len(array) - 1
    while low <= high:
        middle = (low + high) // 2
        if target == array[middle]:
            return 0
        elif target < array[middle]:
            high = middle - 1
        else:
            low = middle + 1
    return -1


def quick_sort(array):
    if len(array) < 2:
        return array
    left = []
    right = []
    pivot = array[0]
    for i in range(1, len(array)):
        if array[i] < pivot:
            left.append(array[i])
        else:
```

```
            right.append(array[i])
    return quick_sort(left) + [pivot] + quick_sort(right)
>>> menu()

1. Read file
2. Linear Search
3. Binary Search
4. End
1

1. Read file
2. Linear Search
3. Binary Search
4. End
3

Sales figure to find: 366
Sales figure 366 was found in the file CUPS-SOLD1.txt

1. Read file
2. Linear Search
3. Binary Search
4. End
4

>>> menu()

1. Read file
2. Linear Search
3. Binary Search
4. End
1

1. Read file
2. Linear Search
3. Binary Search
4. End
3

Sales figure to find: 123
Sales figure 123 was not found in the file CUPS-SOLD1.txt

1. Read file
2. Linear Search
3. Binary Search
4. End
4
```

Evidence 10 *Program code*

#NOTE: Changes from previous parts of code are bolded.

```
def menu():
    while True:
```

```
        choice = get_choice()
        if choice == 4:
            break
        elif choice == 1:
            x = 2
            file_data = read_file(x)
        else:
            try:
                file_data
            except NameError:
                print("\nPlease read file data using option 1 first!")
            else:
            sales_figure = input("\nSales figure to find: ")
            if choice == 2: #linear search
                found, count = LinearSearch(file_data, sales_figure)
                if found == 0:
                    print("\nNumber of day(s) with this sales"\
                          " figure reported: ", count)
            else: #choice = 3 #binary search
                try:
                    sorted_file_data
                except NameError:
                    sorted_file_data = quick_sort(file_data)
                    found = BinarySearch(sorted_file_data, \
                                         sales_figure)
            print("Sales figure " + sales_figure + " was ", \
                  end = '')
            if found == -1:
                print("not ", end = '')
            print("found in the file ", end = '')
            if x == 1:
                print("CUPS-SOLD1.txt")
            else: #x = 2
                print("CUPS-SOLD2.txt")

def LinearSearch(array, target):
    count = 0
    for i in range(len(array)):
        if array[i] == target:
            count += 1
    if count == 0:
        found = -1
    else:
        found = 0
    return found, count
```

Evidence 11 *Screenshots X 3*

Normal:

```
>>> menu()

1. Read file
2. Linear Search
3. Binary Search
4. End
1

1. Read file
2. Linear Search
3. Binary Search
4. End
2

Sales figure to find: 190

Number of day(s) with this sales figure reported:  5
Sales figure 190 was found in the file CUPS-SOLD2.txt

1. Read file
2. Linear Search
3. Binary Search
4. End
2

Sales figure to find:
Sales figure  was not found in the file CUPS-SOLD2.txt

1. Read file
2. Linear Search
3. Binary Search
4. End
4
```

Erroneous:

```
>>> menu()

1. Read file
2. Linear Search
3. Binary Search
4. End
1

1. Read file
2. Linear Search
3. Binary Search
4. End
2

Sales figure to find: a
Sales figure a was not found in the file CUPS-SOLD2.txt

1. Read file
2. Linear Search
3. Binary Search
4. End
4
```

Boundary:

```
>>> menu()

1. Read file
2. Linear Search
3. Binary Search
4. End
1

1. Read file
2. Linear Search
3. Binary Search
4. End
2

Sales figure to find: 0

Number of day(s) with this sales figure reported:  2
Sales figure 0 was found in the file CUPS-SOLD2.txt

1. Read file
2. Linear Search
3. Binary Search
4. End
4
```

QUESTION 3

Evidence 12 *Program code*

```
def CreateUpdateFile():
    file_handle = open("RESULTS.txt")
    team1, goal1, team2, goal2 = file_handle.readline().strip().split("
")
    file_handle.close()
    match_results = get_match_results(goal1, goal2)
    try:
        file_handle = open("NEWFILE.txt", "a")
    except FileNotFoundError:
        file_handle = open("NEWFILE.txt", "w")
    file_handle.write(",".join([team1, match_results[0], goal1, goal2])\
                      + "\n")
    file_handle.write(",".join([team2, match_results[1], goal2, goal1])\
                      + "\n")
    file_handle.close()

def get_match_results(goal1, goal2):
    goal1 = int(goal1)
    goal2 = int(goal2)
    if goal1 == goal2:
        return "D", "D"
    elif goal1 > goal2:
        return "W", "L"
    return "L", "W"
```

Evidence 13 *Program code*

```
def CreateUpdateFile():
    #resetting the update file
    file_handle = open("NEWFILE.txt", "w")
    file_handle.close()

    file_handle1 = open("RESULTS.txt")
    for line in file_handle1:
        team1, goal1, team2, goal2 = line.strip().split(" ")
        match_results = get_match_results(goal1, goal2)
        try:
            file_handle2 = open("NEWFILE.txt", "a")
        except FileNotFoundError:
            file_handle2 = open("NEWFILE.txt", "w")
        file_handle2.write(",".join([team1, match_results[0], goal1,
goal2])\
                           + "\n")
        file_handle2.write(",".join([team2, match_results[1], goal2,
goal1])\
                           + "\n")
        file_handle2.close()
    file_handle1.close()
```

Evidence 14 *Screenshots X 2*

First 10:

```
    Lovepool,W,3,2
    Arsenic,L,2,3
    Totoham,D,2,2
    MadCity,D,2,2
    MadUnited,W,2,1
    Chelsand,L,1,2
    Stroke,L,0,1
    Newcast,W,1,0
    Everlong,W,1,0
    WestBeef,L,0,1

Last 10:
    Totoham,D,1,1
    Newcast,D,1,1
    WestBeef,L,0,2
    MadUnited,W,2,0
    Stroke,D,0,0
    Totoham,D,0,0
    Arsenic,W,3,2
    Everlong,L,2,3
    MadCity,W,4,3
    Lovepool,L,3,4
```

Evidence 15 *Program code*

```
def ComputeTeamStat(team_name):
    file_data = get_file_data()
    team_results = [0, 0, 0, 0, 0, 0]
    #team_results = [played, wins, draws, losses, goals_for,
goals_against]
    for match in file_data:
        if team_name in match:
            team1, goal1, team2, goal2 = match
            match_results = get_match_results(goal1, goal2)
            if team_name == team1:
                match_results = match_results[0]
                goals_for, goals_against = int(goal1), int(goal2)
            else: #team_name = team2
                match_results = match_results[1]
                goals_for, goals_against = int(goal2), int(goal1)
            team_results[0] += 1
            if match_results == 'W':
                team_results[1] += 1
            elif match_results == 'D':
                team_results[2] += 1
            else: #match_result = 'L'
                team_results[3] += 1
            team_results[4] += goals_for
            team_results[5] += goals_against
    #goal_difference
    team_results.append(team_results[-2] - team_results[-1])
```

```
    #points
    team_results.append(team_results[1] * 3 + team_results[2])

    print_header()
    played, wins, draws, losses, goals_for, goals_against, \
            goal_difference, points = team_results
    print("{0:<12}{1:^8}{2:^8}{3:^8}{4:^8}{5:^8}{6:^8}{7:^8}{8:^8}"\
          .format(team_name, played, wins, draws, losses, goals_for, \
                  goals_against, goal_difference, points))

def get_file_data():
    file_handle = open("RESULTS.txt")
    results = [x.split(" ") for x in
file_handle.read().strip().split("\n")]
    file_handle.close()
    return results

def print_header():
    print("{0:<12}{1:^8}{2:^8}{3:^8}{4:^8}{5:^8}{6:^8}{7:^8}{8:^8}"\
          .format("Team", "P", "W", "D", "L", "GF", "GA", "GD",
"Points"))
    print("=" * 76)
```

Evidence 16 *Screenshot*

```
>>> ComputeTeamStat('Everlong')
Team            P       W       D       L       GF      GA      GD    Points
============================================================================
Everlong        5       1       2       2       8       9       -1      5
```

Evidence 17 *Program code*

```
def quick_sort(array, index): #descending order
    if len(array) < 2:
        return array
    left = []
    right = []
    pivot = array[0]
    for i in range(1, len(array)):
        if array[i][index] > pivot[index]:
            left.append(array[i])
        else:
            right.append(array[i])
    return quick_sort(left, index) + [pivot] + quick_sort(right, index)

def GenerateTable():
    file_handle = open("TEAMS.txt")
    teams = file_handle.read().strip().split("\n")
    file_handle.close()
    file_data = get_file_data()
    results = []
    for team in teams:
```

```
        results.append(get_team_results(file_data, team))

    results = quick_sort(results, -2)
    results = quick_sort(results, -1)

    print_header()
    for team_results in results:
        team_name, played, wins, draws, losses, goals_for,
goals_against,\
                    goal_difference, points = team_results
        print("{0:<12}{1:^8}{2:^8}{3:^8}{4:^8}{5:^8}{6:^8}{7:^8}{8:^8}"\
              .format(team_name, played, wins, draws, losses, goals_for,
\
                      goals_against, goal_difference, points))


def get_team_results(file_data, team_name):
    #essentially ComputeTeamStat() but returning \
    #the results instead of printing them

    #also takes in file_data so that it does not need to be
    #created every time this function runs

    team_results = [0, 0, 0, 0, 0, 0]
    #team_results = [played, wins, draws, losses, goals_for,
goals_against]
    for match in file_data:
        if team_name in match:
            team1, goal1, team2, goal2 = match
            match_results = get_match_results(goal1, goal2)
            if team_name == team1:
                match_results = match_results[0]
                goals_for, goals_against = int(goal1), int(goal2)
            else: #team_name = team2
                match_results = match_results[1]
                goals_for, goals_against = int(goal2), int(goal1)
            team_results[0] += 1
            if match_results == 'W':
                team_results[1] += 1
            elif match_results == 'D':
                team_results[2] += 1
            else: #match_result = 'L'
                team_results[3] += 1
            team_results[4] += goals_for
            team_results[5] += goals_against
    #goal_difference
    team_results.append(team_results[-2] - team_results[-1])
    #points
    team_results.append(team_results[1] * 3 + team_results[2])

    return [team_name] + team_results
```

Evidence 18 *Screenshot*

```
>>> GenerateTable()
Team            P       W       D       L       GF      GA      GD    Points
=============================================================================
MadCity         5       3       2       0       13      9       4       11
MadUnited       4       3       1       0       8       3       5       10
Arsenic         5       3       0       2       13      11      2       9
Lovepool        5       3       0       2       10      10      0       9
Chelsand        4       2       1       1       10      7       3       7
Totoham         5       1       3       1       6       7       -1      6
Newcast         4       1       2       1       3       3       0       5
Everlong        5       1       2       2       8       9       -1      5
Stroke          5       0       1       4       5       12      -7      1
WestBeef        4       0       0       4       2       7       -5      0
```

QUESTION 4

Evidence 19 *Program code*

```
class Node():
    def __init__(self, Name, Score, LeftP, RightP):
        self._Name = Name
        self._Score = Score
        self._LeftP = LeftP
        self._RightP = RightP

    def get_Name(self):
        return self._Name

    def set_Name(self, new_Name):
        self._Name = new_Name

    def get_Score(self):
        return self._Score

    def set_Score(self, new_Score):
        self._Score = new_Score

    def get_LeftP(self):
        return self._LeftP

    def set_LeftP(self, new_LeftP):
        self._LeftP = new_LeftP

    def get_RightP(self):
        return self._RightP

    def set_RightP(self, new_RightP):
        self._RightP = new_RightP

class BinaryTree():
    def __init__(self):
        self._ThisTree = [None]
        for i in range(2, 21):
```

```
            self._ThisTree.append(Node('', '', i, 0))
        self._ThisTree.append(Node('', '', 0, 0))
        #additional None at the start to account for 1-indexing
        self._Root = 0
        #self._Root = 0 means nothing is currently in the tree
        self._NextFreePosition = 1
```

Evidence 20 *Program code*

```
    def AddNodeToTree(self, Name, Score):
        if self._NextFreePosition == 0:
            print("Tree is full. Unable to add to tree.")
            return
        self._ThisTree[self._NextFreePosition].set_Name(Name)
        self._ThisTree[self._NextFreePosition].set_Score(Score)
        temp = self._ThisTree[self._NextFreePosition].get_LeftP()
        self._ThisTree[self._NextFreePosition].set_LeftP(0)
        if self._Root == 0:
            self._Root = 1
        else:
            cur = self._Root
            while True:
                if cur == 0:
                    raise ValueError
                if Score > self._ThisTree[cur].get_Score():
                    if self._ThisTree[cur].get_RightP() == 0:
                        self._ThisTree[cur].set_RightP(\
                            self._NextFreePosition)
                        break
                    cur = self._ThisTree[cur].get_RightP()
                else:
                    if self._ThisTree[cur].get_LeftP() == 0:
                        self._ThisTree[cur].set_LeftP(\
                            self._NextFreePosition)
                        break
                    cur = self._ThisTree[cur].get_LeftP()
        self._NextFreePosition = temp
```

Evidence 21 *Program code*

#NOTE: Changes from previous parts of code are bolded.

```
class Node():
    def __init__(self, Name, Score, LeftP, RightP):
        self._Name = Name
        self._Score = Score
        self._LeftP = LeftP
        self._RightP = RightP

    def get_Name(self):
        return self._Name

    def set_Name(self, new_Name):
        self._Name = new_Name
```

```
    def get_Score(self):
        return self._Score

    def set_Score(self, new_Score):
        self._Score = new_Score

    def get_LeftP(self):
        return self._LeftP

    def set_LeftP(self, new_LeftP):
        self._LeftP = new_LeftP

    def get_RightP(self):
        return self._RightP

    def set_RightP(self, new_RightP):
        self._RightP = new_RightP

    def __str__(self):
        return "{0:<20}{1:<7}{2:<7}{3}"\
               .format(self._Name, self._Score, self._LeftP,
self._RightP)

class BinaryTree():
    def __init__(self):
        self._ThisTree = [None]
        for i in range(2, 21):
            self._ThisTree.append(Node('', '', i, 0))
        self._ThisTree.append(Node('', '', 0, 0))
        #additional None at the start to account for 1-indexing
        self._Root = 0
        #self._Root = 0 means nothing is currently in the tree
        self._NextFreePosition = 1

    def AddNodeToTree(self, Name, Score):
        if self._NextFreePosition == 0:
            print("Tree is full. Unable to add to tree.")
            return
        self._ThisTree[self._NextFreePosition].set_Name(Name)
        self._ThisTree[self._NextFreePosition].set_Score(Score)
        temp = self._ThisTree[self._NextFreePosition].get_LeftP()
        self._ThisTree[self._NextFreePosition].set_LeftP(0)
        if self._Root == 0:
            self._Root = 1
        else:
            cur = self._Root
            while True:
                if cur == 0:
                    raise ValueError
                if Score > self._ThisTree[cur].get_Score():
                    if self._ThisTree[cur].get_RightP() == 0:
                        self._ThisTree[cur].set_RightP(\
                            self._NextFreePosition)
```

```
                        break
                    cur = self._ThisTree[cur].get_RightP()
                else:
                    if self._ThisTree[cur].get_LeftP() == 0:
                        self._ThisTree[cur].set_LeftP(\
                            self._NextFreePosition)
                        break
                    cur = self._ThisTree[cur].get_LeftP()
        self._NextFreePosition = temp

    def OutputData(self):
        print("Root: ", self._Root)
        print("NextFreePosition: ", self._NextFreePosition)
        if self._Root == 0:
            print("Tree is empty.")
        else:
            print("\n{0:<8}{1:<20}{2:<7}{3:<7}{4}"
                  .format("Index", "Name", "Score", "LeftP", "RightP"))
            for i in range(self._Root, len(self._ThisTree)):
                print("{0:<8}{1}".format(i, self._ThisTree[i]))
```

Evidence 22 *Program code*

```
file_handle = open("SCORES.txt")
file_data = []
for line in file_handle:
    temp = line.strip().split("|")
    temp[1] = int(temp[1])
    file_data.append(temp)
file_handle.close()

#bst: binary search tree
bst = BinaryTree()
for data in file_data:
    bst.AddNodeToTree(data[0], data[1])
bst.OutputData()
```

Evidence 23 *Screenshot*

```
Root:  1
NextFreePosition:  11

Index   Name                Score  LeftP  RightP
1       Tom Ng              40     3      2
2       Paul Ang            70     4      5
3       Pauline             35     9      10
4       Lim S C             60     7      0
5       Ma Zhu              80     6      8
6       Hnit Sun            75     0      0
7       Zeck                46     0      0
8       Thomas Teh          100    0      0
9       Crystal Ma          0      0      0
10      Mak Pei Pei         38     0      0
11                                 12     0
12                                 13     0
13                                 14     0
14                                 15     0
15                                 16     0
16                                 17     0
17                                 18     0
18                                 19     0
19                                 20     0
20                                 0      0
```

Evidence 24 *Program code*

#NOTE: Changes from previous parts of code are bolded.

```
class BinaryTree():
    def __init__(self):
        self._ThisTree = [None]
        for i in range(2, 21):
            self._ThisTree.append(Node('', '', i, 0))
        self._ThisTree.append(Node('', '', 0, 0))
        #additional None at the start to account for 1-indexing
        self._Root = 0
        #self._Root = 0 means nothing is currently in the tree
        self._NextFreePosition = 1

    def AddNodeToTree(self, Name, Score):
        if self._NextFreePosition == 0:
            print("Tree is full. Unable to add to tree.")
            return
        self._ThisTree[self._NextFreePosition].set_Name(Name)
        self._ThisTree[self._NextFreePosition].set_Score(Score)
        temp = self._ThisTree[self._NextFreePosition].get_LeftP()
        self._ThisTree[self._NextFreePosition].set_LeftP(0)
        if self._Root == 0:
            self._Root = 1
        else:
            cur = self._Root
            while True:
                if cur == 0:
                    raise ValueError
```

```
                if Score > self._ThisTree[cur].get_Score():
                    if self._ThisTree[cur].get_RightP() == 0:
                        self._ThisTree[cur].set_RightP(\
                            self._NextFreePosition)
                        break
                    cur = self._ThisTree[cur].get_RightP()
                else:
                    if self._ThisTree[cur].get_LeftP() == 0:
                        self._ThisTree[cur].set_LeftP(\
                            self._NextFreePosition)
                        break
                    cur = self._ThisTree[cur].get_LeftP()
        self._NextFreePosition = temp

    def OutputData(self):
        print("Root: ", self._Root)
        print("NextFreePosition: ", self._NextFreePosition)
        if self._Root == 0:
            print("Tree is empty.")
        else:
            print("\n{0:<8}{1:<20}{2:<7}{3:<7}{4}"
                  .format("Index", "Name", "Score", "LeftP", "RightP"))
            for i in range(self._Root, len(self._ThisTree)):
                print("{0:<8}{1}".format(i, self._ThisTree[i]))

    def RankList(self):
        if self._Root == 0:
            print("Tree is empty.")
        else:
            print("\n{0:<20}{1}"
                  .format("Name", "Score"))
            self.InReverseOrderTraversal(self._Root)

    def InReverseOrderTraversal(self, cur_index):
        if cur_index != 0:
            self.InReverseOrderTraversal(\
                self._ThisTree[cur_index].get_RightP())
            print("{0:<20}{1}"\
                  .format(self._ThisTree[cur_index].get_Name(), \
                          self._ThisTree[cur_index].get_Score()))
            self.InReverseOrderTraversal(\
                self._ThisTree[cur_index].get_LeftP())


file_handle = open("SCORES.txt")
file_data = []
for line in file_handle:
    temp = line.strip().split("|")
    temp[1] = int(temp[1])
    file_data.append(temp)
file_handle.close()

#bst: binary search tree
bst = BinaryTree()
```

```
for data in file_data:
    bst.AddNodeToTree(data[0], data[1])
bst.OutputData()
bst.RankList()
```

Evidence 25 *Screenshot*

```
Name                Score
Thomas Teh          100
Ma Zhu              80
Hnit Sun            75
Paul Ang            70
Lim S C             60
Zeck                46
Tom Ng              40
Mak Pei Pei         38
Pauline             35
Crystal Ma          0
```