|  |  |
| --- | --- |
| 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* |
| **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* |
| **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** |
| **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)** |
| **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:    Erroneous:    Boundary: |
| **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:    Last 10: |
| **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* |
| **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* |
| **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* |
| **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* |