EECE 230X Introduction to Computation and Programming
Open Enrollment Section
Final Exam

November 25, 2023

- The duration of this exam is 2 hours, starting at 5:00 PM.
- The exam consists of 4 problems with 5 parts in total.
- **The solution of each part should be submitted in a separate Python file**. **Each of the 5 parts has a separate submission box**. In particular, submit
  - Part (a) of Problem 1 as `Prob1a.py` in the submission box corresponding to Problem 1.a
  - Part (b) of Problem 1 as as `Prob1b.py` in the submission box corresponding to Problem 1.b
  - Problem 2 as `Prob2.py` in the submission box corresponding to Problem 2
  - Problem 3 as `Prob3.py` in the submission box corresponding to Problem 3
  - Problem 4 as `Prob4.py` in the submission box corresponding to Problem 4.
- **It is your responsibility to make sure that your work is properly submitted within the allotted exam time**
- You are allowed to access all materials available on the course Moodle page during the exam.
- You are expected to submit your own work. **Cheating will not be tolerated**. Examples of cheating include seeking someone else's help to write your code or relying on Large Language Models (LLM) such as ChatGPT. In fact, **submitting your own work increases your chances of passing the exam** for the following reasons. All submissions will be checked for plagiarism. If needed, we might arrange live meetings with students to confirm the authenticity of their submissions. Also, it is not hard to check if a submission is generated by an LLM. Keep in mind that you don't have to solve all the exam problems to pass the exam. However, detection of cheating on even a single problem will result in a failing grade, regardless of the merit of the rest of your work.
- The problems are of varying difficulty. Below is a rough ordering estimate of the problems in order of increasing difficulty.
  - Level 1 (60 points): Part 1.a (nonefficient) and Problem 2
  - Level 2 (50 points): Part 1.a (efficient), Part 1.b, and Problem 3
  - Level 3 (30 points): Problem 4
- Plan your time wisely. Do not spend too much time on any one problem. Read through all of them first and attack them in the order that allows you to make the most progress.
- It's alright if you don't solve all the problems. Aim to solve as much as you can within the given time.
- Good luck!

Problem 1 (40 points). Kayaking Adventure

A *kayak* is a small long boat that people sit inside and move using long paddles.

In MSFEA Town, there is a kayaking group of people known for their regular trips. The group has a number of kayaks, each capable of seating two people. It is a well known that a large weight difference between the two occupants of a kayak can cause it to crash.

Karim, the leader and owner of the kayaking group, understands this risk. To ensure the safety of the trips, he carefully assigns members to kayaks to balance the weights, therefore reducing the risk of accidents. In this problem, you will help Karim in his safety analysis.

A kayak is *stable* if the absolute difference in weights between the two kayak's occupants is at most 5 `Kg`.

a) **(30 points) One stable kayak.** You are asked to implement the function `oneStable(L)` which given a list L that contains the weights in Kg of all the kayaking group members, returns `True` if it is possible to assign 2 members to a single kayak in such a way that the kayak is stable. Otherwise, it should return `False`.

For instance, on `L = [100,70,90, 73,96]`, the function should return `True` (choose 70 and 73). However, on `L = [105,63,90, 70,96]`, the function should return `False`.

See also the examples in the below test program.

Any correct solution is 20/30 points. To get full grade do it in log-linear time (recall that log-linear means $O(n \log n)$, where n is the input length).

Test program/output:

```
print(oneStable([]))                     | False
print(oneStable([75]))                   | False
print(oneStable([75,70]))                | True
print(oneStable([75,69]))                | False
print(oneStable([75,70,100]))            | True
print(oneStable([85,70,100]))            | False
print(oneStable([100,70,90, 73,96]))     | True
print(oneStable([105,63,90, 70,96]))     | False
```

Submit your solution as a file named Prob1a.py in the submission box on Moodle designated for Problem 1.a. Ensure you include your name as a comment at the beginning of the file.

b) **(10 points) All kayaks stable.** In this part, Karim would like to check if all team members could be seated on stable kayaks. Assume that the team has $2n$ members and the number of kayaks is n, for some integer $n \geq 1$.

Implement the function `allStable(L)` that, given an even-length list L containing the weights of all group members, checks if they can be all seated on stable kayaks. If so, the function should return `True`, and `False` otherwise.

For instance, on `L = [100,70,96, 73]`, the function should return `True` since the members of weights 100 and 96 could be stead on the first kayak and those of weights 70 and 73 could be seated on the second kayak.

However, on `L = [100,70,90,73]`, the function should return `False`.

See also the examples in the below test program.

Any correct solution is worth full grade.

Test program/output:

```
print(allStable([75,70]))                  | True
print(allStable([75,69]))                  | False
print(allStable([100,70,96,73]))           | True
print(allStable([100,70,90,73]))           | False
print(allStable([100,80,70,96,73,75]))     | True
print(allStable([100,82,70,96,73,75]))     | False
```

Submit your solution as a file named Prob1b.py in the submission box on Moodle designated for Problem 1.b. Ensure you include your name as a comment at the beginning of the file.

Problem 2 (40 points). Classes and Dictionaries: AUB Football League

In this problem, you are asked to design a Python class `AUBFL` that models the AUB Football League. The code you write will serve for managing the league rankings, team points, and playing matches.

Include in the class `AUBFL` the data attribute `Points`, which is a dictionary, where each key is the team name and its value is the number of points it has.

Include the following method attributes:

- The initializer `__init__(self, Teams)`, which takes the value of an array of strings `Teams` as an input argument. The strings in `Teams` are names of the teams. The initializer should initialize the data attribute `self.Points` to a dictionary whose keys are the strings in `Teams` and values are all 0 since the league has not started yet.

  Make sure that `Teams` is of type `list` and all of its entries are of type `str`. If this is not the case, the initializer should stop the execution of the program with an error message `"Invalid Input!"`

- `__str__`, which when invoked on an instance of `AUBFL`, casts it into a string containing each team with its number of points on a separate line separated by the character ":", i.e., of the form :
  `team1: points1`
  `team2: points2`
  ...

  See the below test program.

- `play(self,team1,team2,score1,score2)`, which takes two strings `team1` and `team2`, and two integers `score1` and `score2` as input argument, where `score1` and `score2` are the scores of `team1` and `teams2` after playing a game. It should update the points of `team1` and `team2` according to the scores. In particular, **if the scores are not equal, the winner gets 3 points. If they are equal, they both get 1 point.**

  This method should assert that `team1` and `team2` are in the league and that `score1` and `score2` are integers. If this is not the case, it should stop the program with the error message `"Invalid Input!"`

- `add(self, team)`, which takes a string `team` as an input argument and adds it to the league with 0 points.

  If the team is already in the league, it should stop the program with the error message `"Invalid Input!"`

- `winner`, which when invoked on an instance of `AUBFL`, returns the name of the winner, i.e., the team with the largest number of points. If more than one team has the maximum number of points, this method should return the string `"No Winner"`.

  Any correct solution is worth full grade.

Test program:

```
League = AUBFL(["CSE", "CIVE", "PremedStudentSociety"])
print(League)
League.play("CSE", "PremedStudentSociety", 3, 2)
print(League)
League.add("OSB")
print(League)
League.play("CIVE", "OSB", 1, 1)
print(League)
League.play("OSB", "CSE", 1, 1)
print (League)
print("The winner is  " +League.winner())
```

Output:

```
CSE:0
CIVE:0
PremedStudentSociety:0
```

```
CSE:3
CIVE:0
PremedStudentSociety:0

CSE:3
CIVE:0
PremedStudentSociety:0
OSB:0

CSE:3
CIVE:1
PremedStudentSociety:0
OSB:1

CSE:4
CIVE:1
PremedStudentSociety:0
OSB:2

The winner is CSE
```

Submit your solution as a file named Prob2.py in the submission box on Moodle designated for Problem 2. Ensure you include your name as a comment at the beginning of the file.

Problem 3 (30 points). Pattern

In this problem, you are asked to implement a function `pattern(k)` which given an integer $k \geq 0$, returns a string as shown in the below examples. Guess the pattern for $k \geq 6$. You are advised to use recursion. Any correct solution is worth full grade.

Test Program:

```
for k in range(0,6):
   print("pattern("+str(k)+"): " + pattern(k))
```

Output:

```
pattern(0): 1
pattern(1): 1
pattern(2): 1001
pattern(3): 10010001
pattern(4): 1001000100001001
pattern(5): 10010001000010010000010010001
```

Submit your solution as a file named Prob3.py in the submission box on Moodle designated for Problem 3. Ensure you include your name as a comment at the beginning of the file.

Problem 4 (30 points). Check if sorted list has three distinct elements.

In this part, we are given a list consisting of integers sorted in nondecreasing order. We are interested in checking if the list has at least 3 distinct values. In particular, implement the function `threeDistinct(L)`, which given a list L consisting of integers sorted in nondecreasing order, returns `True` if `L` has at least 3 distinct values, and `False` otherwise.

The use of **for or while loops is strictly forbidden in this part**. Also, the **use of all methods associated with the list type** is **strictly forbidden**. You are asked to solve it using **recursion**. Namely, implement the function `threeDistinct` as a wrapper function which calls a recursive function `threeDistinctRecursive` with appropriate input parameters.

Examples:

- For `L= [10,10,10,21,21]`, `threeDistinct` should return `False`.
- For `L= [10,10,10,16,21,21]`, `threeDistinct` should return `True` since `L` has 3 distinct values
- For `L= [10,10,10,16,19,19,19,21,21]`, `threeDistinct` should return `True` since `L` has 4 distinct values

See also the examples in the below test program.

You are not asked to check if `L` is sorted in nondecreasing order, and you are not supposed to give any guarantee on the behavior of your function if this is not the case.

Any correct solution is worth 15/30 points. Faster solutions are worth more points. To get full grade, do it in $O(\log n)$ time, where n is the length of `L`.

Test program/output:

```
print(threeDistinct([]))                            | False
print(threeDistinct([1]))                           | False
print(threeDistinct([1,1]))                         | False
print(threeDistinct([1,10]))                        | False
print(threeDistinct([1,1,10]))                      | False
print(threeDistinct([1,1,10,10,10]))                | False
print(threeDistinct([1,1,3,10,10,10]))              | True
print(threeDistinct([1,1,3,10,10,10,10,10,10]))     | True
print(threeDistinct([1,1,1,1,1,3,3,4,10]))          | True
```

Submit your solution as a file named Prob4.py in the submission box on Moodle designated for Problem 4. Ensure you include your name as a comment at the beginning of the file.