Indian Institute of Information Technology and Management – Kerala (IIITM-K)

M.Sc. in Computer Science

Data Structures and Algorithms

(Mid-Term Exam 2)

Submitted By:

Ravi Prakash

M.Sc. Computer Science (Cyber Security)

Sem. I

Dated on: January 1st, 2021

Questions

Following question was provided in the question paper:

> *"There are 1000 students attempting x questions in a competitive examination, where x is your birthdate coded as ddmmyyyy format. For example if your birthday was on 11-12-2000, then x=11122000. Each student can score one mark per right answer, and a penalty of-0.5 marks per wrong answer. The negative marks increases per wrong answer as a penalty p=0.5*n, where n represents the n-th wrong answer. The questions are categorised into 5 topics, with a number of questions in the categories in the ratio 10:4:3:2:1. All the questions are multiple choice questions (MCQ) type, with possibly more than one correct answer."*
>
> *Write a program to automatically read the answers, assign marks, and rank the students based on their performance in each of the five topic categories. Your aim should be to reduce time and space complexity, at the same time ensure accurate results.*
>
> ***(100 points)***

Before the Solutions

Before the solution, I would like to inform that:

1. At least one key algorithm is written in every solution with its sample execution, as well.

2. All of the images and diagrams are self created by me.

3. A running instance of provided codes can be accessed here (notebook link below):

   https://colab.research.google.com/drive/1qudJ5l8q_SBA7Kutgi-0r9Syr91t7aTC?usp=sharing

   (this notebook contains exactly the same code that is submitted in this report)

Solutions

Solution A:

"There are 1000 students attempting x questions in a competitive examination, where x is your birthdate coded as ddmmyyyy format. For example if your birthday was on 11-12-2000, then x=11122000. Each student can score one mark per right answer, and a penalty of-0.5 marks per wrong answer. The negative marks increases per wrong answer as a penalty p=0.5*n, where n represents the n-th wrong answer. The questions are categorised into 5 topics, with a number of questions in the categories in the ratio 10:4:3:2:1. All the questions are multiple choice questions (MCQ) type, with possibly more than one correct answer."

Write a program to automatically read the answers, assign marks, and rank the students based on their performance in each of the five topic categories. Your aim should be to reduce time and space complexity, at the same time ensure accurate results.

Prerequisites & Objective

Following are important to know about, before starting with the solution:

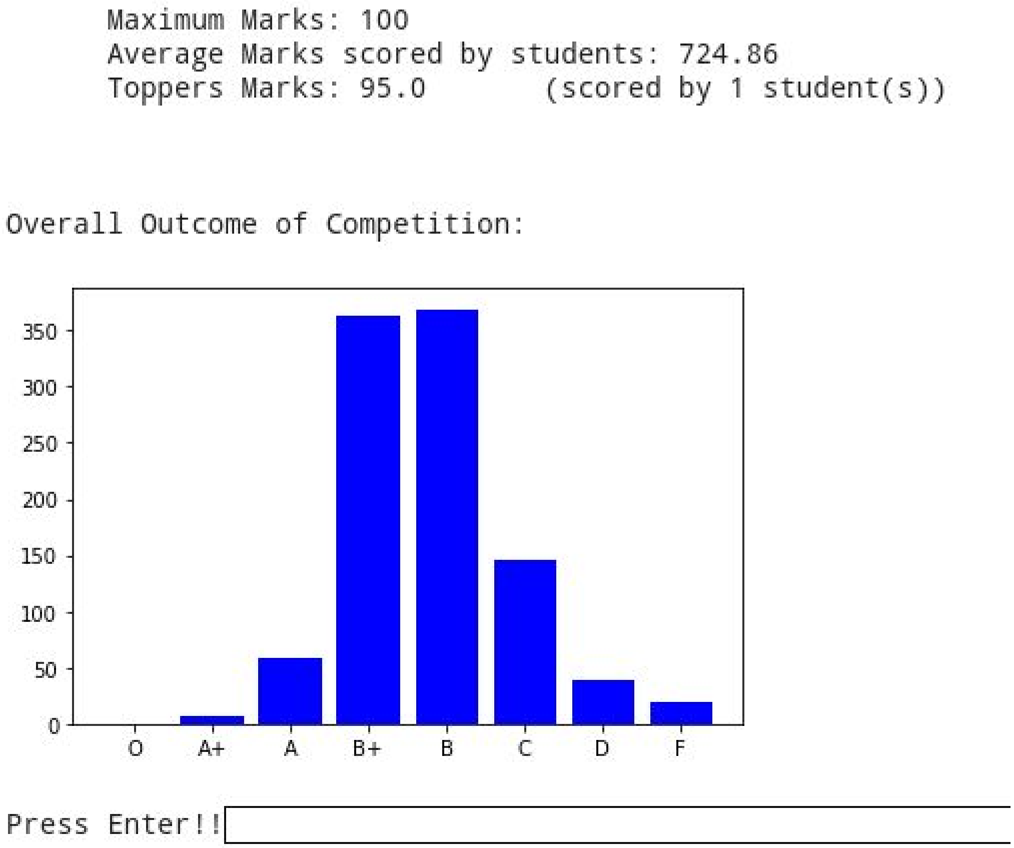
1. Recursion and how it reduces the time complexity.
2. Tree data structures and how links to the subtrees are stored.
3. Percentile VS Percentage.

The objective of this problem:

1. To evaluate and rank the students as better as possible
2. It can be used as a result declaration portal
3. It has NOTHING to do with scanning input for any question as it is not an exam conduction platform.

** The sample outputs are given in this report where in actual implementation, you can find the implementation for the 19071999 questions' solution in each attempted answerSheet.*

A unique and best feature included in this solution is the overall assessment of the exam. The output for the same is given below:



```
Maximum Marks: 100
Average Marks scored by students: 724.86
Toppers Marks: 95.0      (scored by 1 student(s))


Overall Outcome of Competition:
```

```
Press Enter!!
```

Fig. 1.1

This can easily be helpful in detecting the overall stats of the exam as in case of large scale evaluation, this feature ***helps in syllabus planning*** and adjusting the ***difficulty level***. Of the exam.

Why is This the Best Solution?

I have implemented the best solution because:

1. Uses **recursion** to evaluate the marks.
2. Reducing the the **time complexity exactly** to just **half** as compared to the general approaches i.e. **O(n/2)**
3. **Ranks** each **individual in the best and unique way** using **percentile**.
4. **Ranks** students **in a grouped manner** too, using **grades**.
5. Tells grades, marks and percentage of each student.
6. Many more operations are possible to be performed as the storage of marks is best.
7. Gives overall stats of the conducted exams.

Let’s look at the actual program flow and storage mechanism.
We have actually splitted the whole problem into the little parts, and then solved them as modules. At the end, everything is integrated at one place.

Main Program Flow

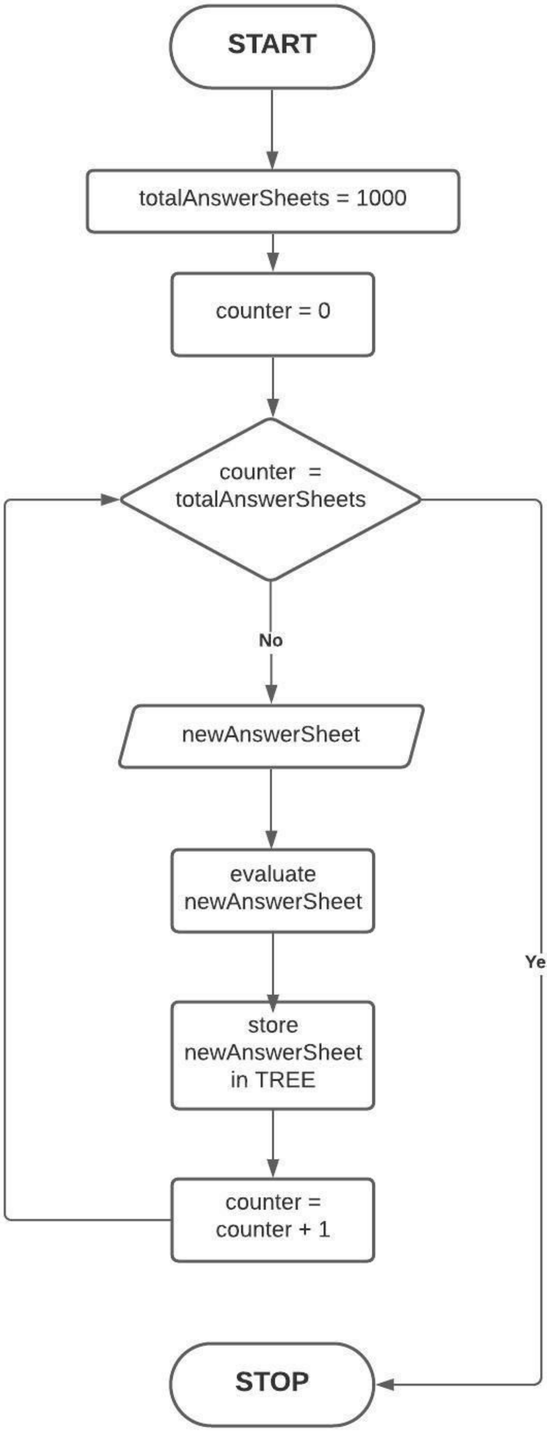


Fig. 1.2

The process of evaluation of marks is following:

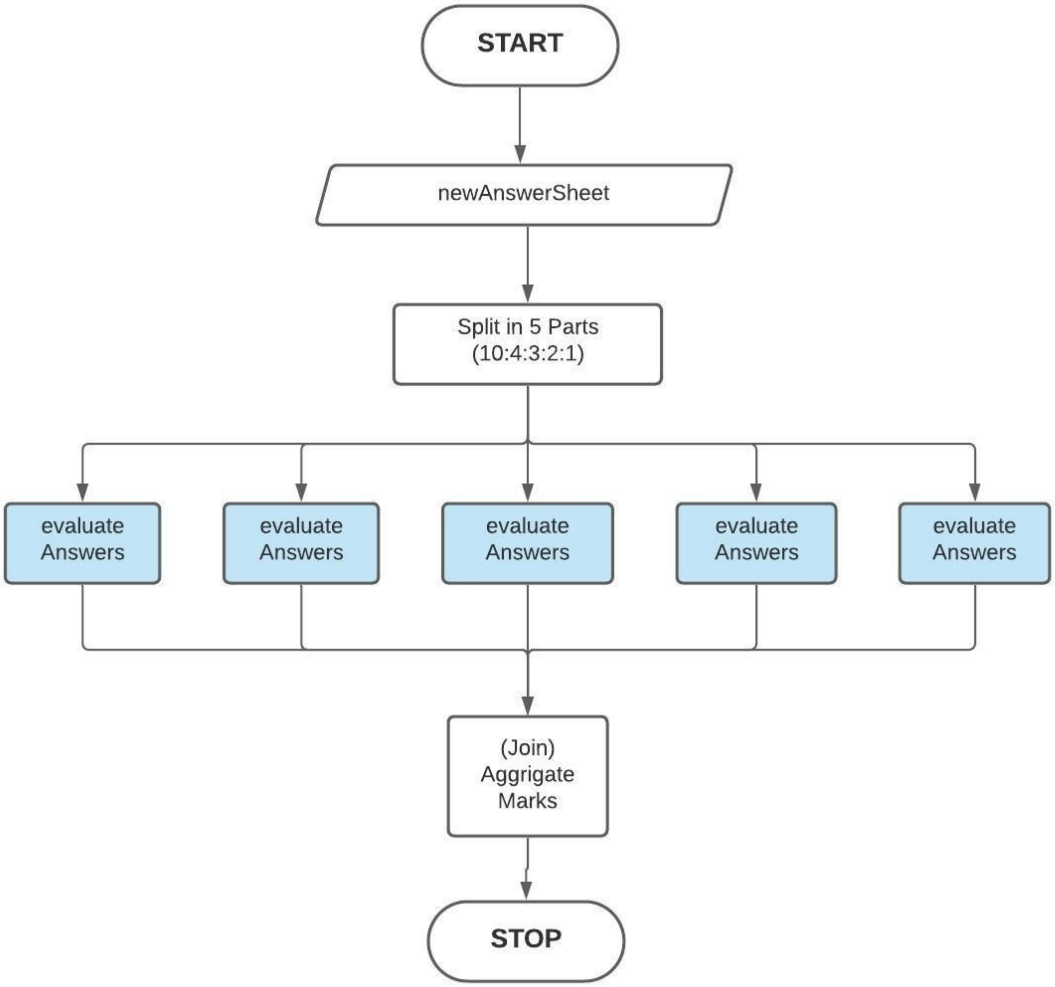


Fig. 1.3

Following algorithm implements the algorithm that is heighted above (*Fig 1.3*):

```
module is evaluateStudent (ratio):
     if length(ratio) == 1:
          evalutedMarksList = evaluateTopic(ratio[0])
     ratio = [10,4,3,2,1]
     evalutedMarksList = evaluateTopic(ratio[0]) +
                                  evaluateStudent(ratio[1 to
                               lastIndex]))
     return evalutedMarksList
```

A single topic is evaluated as below:

```
module is  evalutedMarksList (questionRatio):
     answerList = ratioToQuestionProvider (questionRatio)
     topicMarks = evaluateTopic(answerList)
     return topicMarks
```

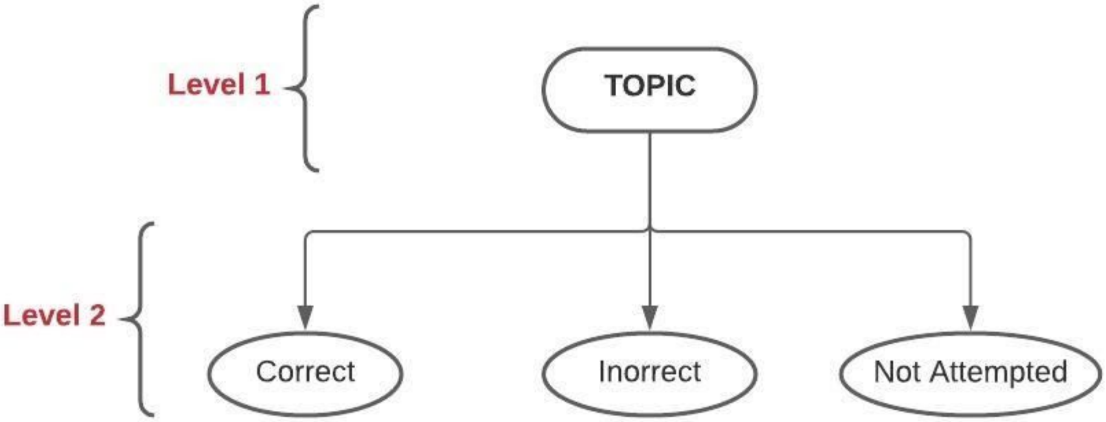


Fig. 1.4

Above figure (*Fig 1.4*) gives an idea of how the solution for every topic is stored in the tree.

Here following property are held by the topic subtree:

1. **Overall marks** scored in a specific topic are stores in **level 1**
2. Leaves only hold the **number count** of each *correct, incorrect & not attempted* solutions.
3. The answer type **counter** is helpful in **comparing every student based on the topics and later ranking them** on the basis of the **percentile**.

In the same way, i have evaluated the marks of all 5 topics for every student and stored them in a separate **topic sub-tree**.

Hecne, the whole subtree for a single student's marks is given below:

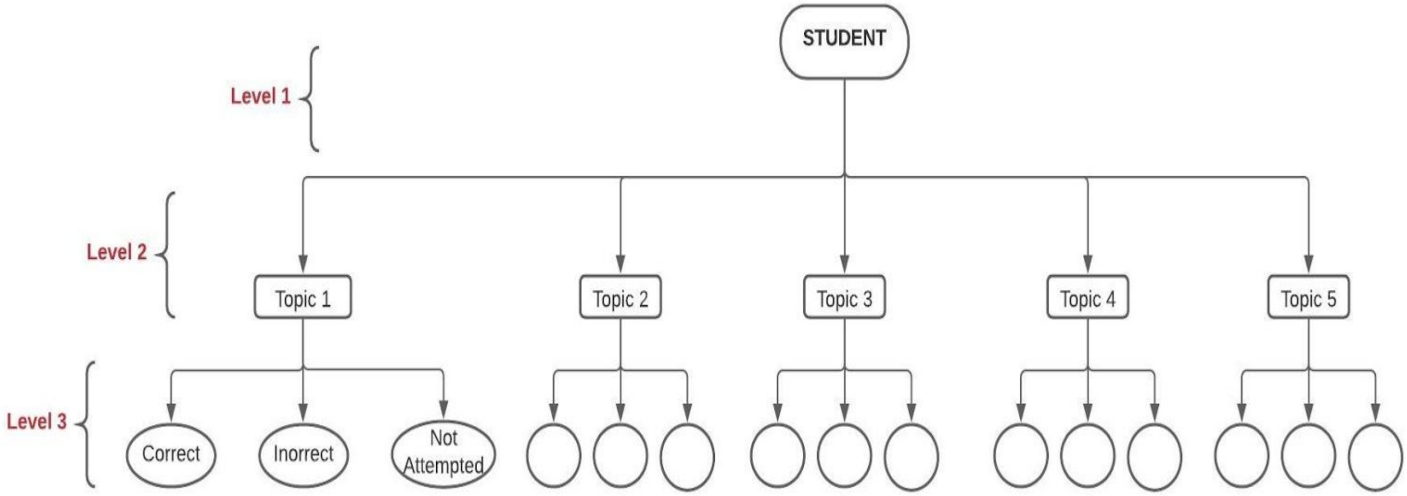


Fig. 1.5

Here, the values stored in every level are as follows:

Level 1:

1. List of total **marks** in **every topic**
2. Overall **grade** of the student, that groups multiple students.
3. **Priority level of student** based on the correct, incorrect and left questions
    a. *used in case of equal mark clash between multiple students*
    b. helps in ranking each student, **individually**
    c. help to find the **percentile**

Level 2:

1. **Overall marks** in one particular **topic**

Level 3:

1. Leaves only hold the **number count** of each *correct, incorrect & not attempted* solutions.

Once we have understood the concept of storing each student, we can view the structure of the whole ***forest*** *where we can store beyond 1000 students' data.*

Storage Tree

Following is the structure for the whole storage:

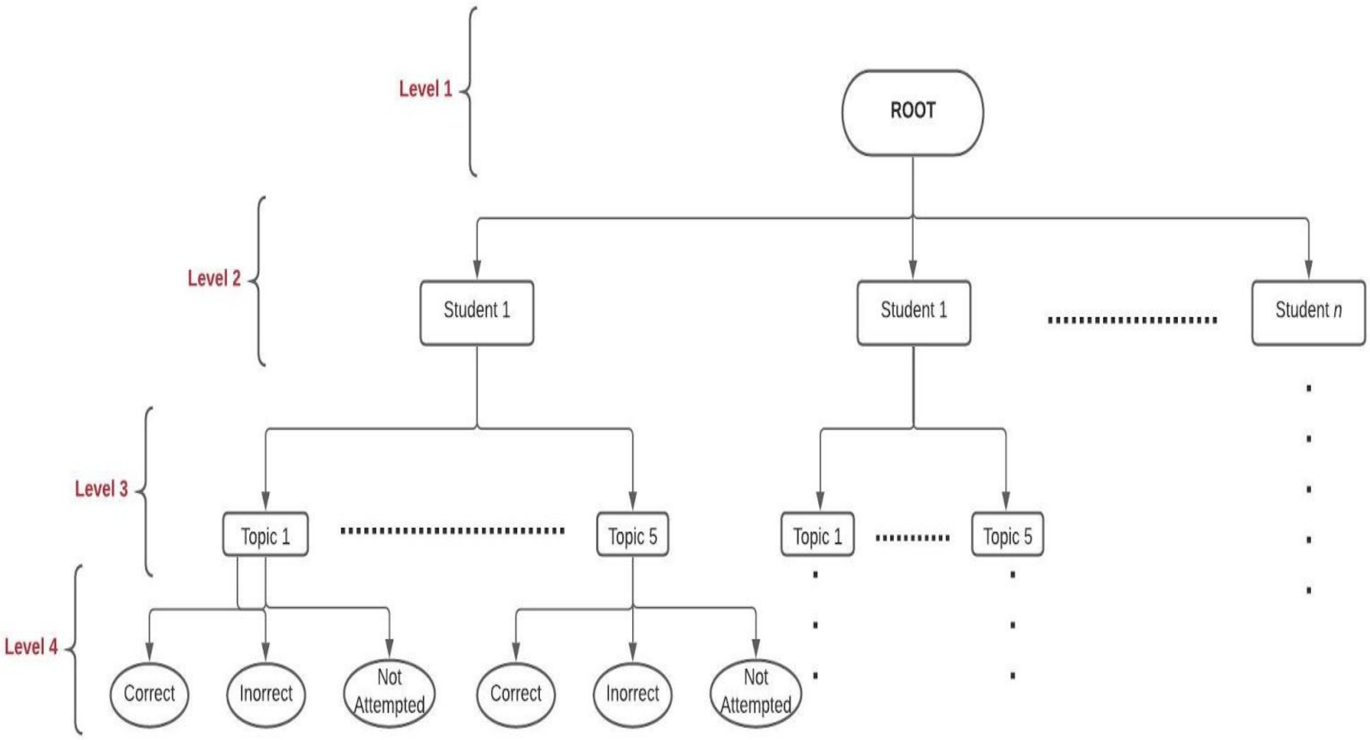


Fig. 1.6

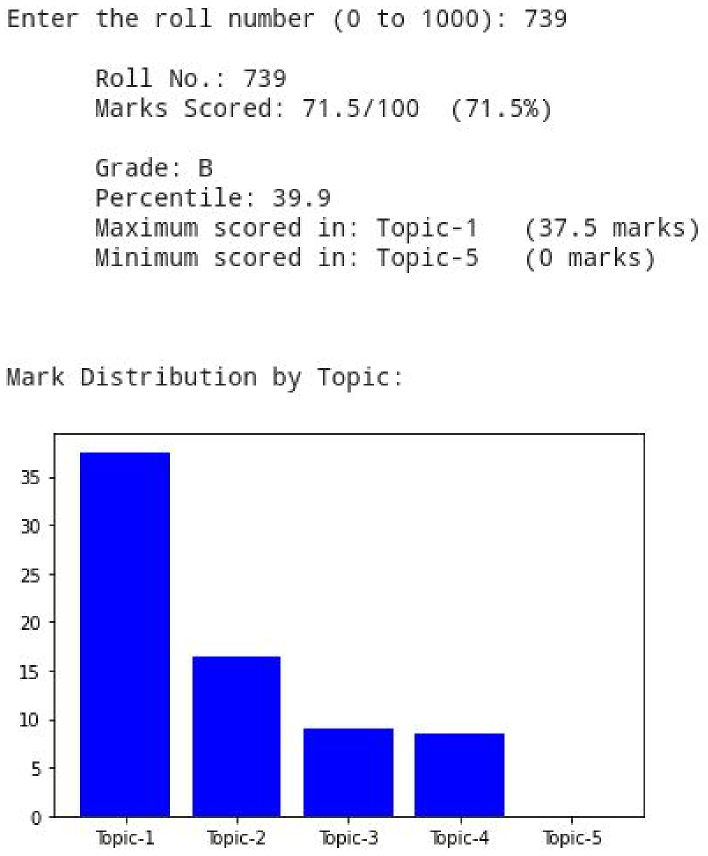
Here, in above figure (*Fig. 1.6*) *Level 2, Level 3, and Level 4* collectively create a subtree **student** (as in *Fig 1.5*). Hence, the abovementioned 3 levels hold exactly the same type of the data that is there in the *Level 1, Level 2 and Level 3* of the student tree.

Level 1:

1. Stores a **link** to every student's data.
2. Stores overall marks of every student, stored in sequence of the Roll Number.
    a. Gives **quick access** to each students' overall score.
    b. Helps to find **overall stats** with a **time complexity** of **O(1)**

The solution goes a single level down to to compare all the students hence **complexity** of **topic-wise comparison between students** is **O(log n)**.

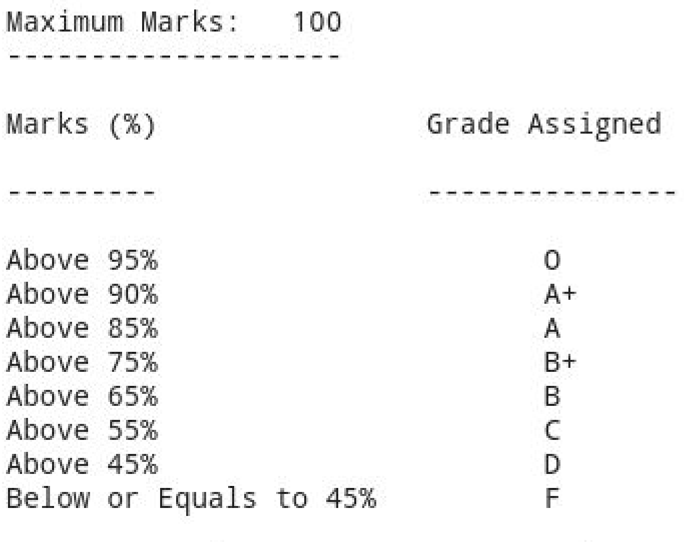
A sample result of one student with many other stats is shown below:



```
Enter the roll number (0 to 1000): 739

      Roll No.: 739
      Marks Scored: 71.5/100  (71.5%)

      Grade: B
      Percentile: 39.9
      Maximum scored in: Topic-1   (37.5 marks)
      Minimum scored in: Topic-5   (0 marks)


Mark Distribution by Topic:
```

Fig. 1.7

The grading criteria can also be seen with the implemented solution:



```
Maximum Marks:   100
--------------------

Marks (%)                Grade Assigned

---------                ---------------

Above 95%                       O
Above 90%                       A+
Above 85%                       A
Above 75%                       B+
Above 65%                       B
Above 55%                       C
Above 45%                       D
Below or Equals to 45%          F
```

Fig. 1.8

Limitations

"Imperfection is the only Perfection"

There always exists a scope of improvement in every thing as nothing can be perfect, indeed. Few of the limitation of my solution are given below:

1. Solution **can not trace exactly one particular solution** and check whether it was correct, incorrect or was not attempted. (tho' in checking the result, one ain't allowed to do so)
2. Can not perform the operations like:
    a. *Update marks of some questions only*
3. Consumes slightly extra memory to give additional details as overall exam stats, that is not asked in the problem.

The implementation (in Python language) of the final code is given on the next page.

Python Code

# heighlighted libraries are not required in case of a real implemenations

```
## Libries
import matplotlib.pyplot as plt
from random import randint
import numpy as np
%precision 2

## for formatted output in python-notebook
from IPython.display import clear_output


############################
### Competition Data Tree ###
############################

####  Data of One Topic  ####
## holds total questions' and their aggr. marks
class answerCount:
  def __init__(self):
    self.__marksCount = None
    self.__marks = None

  def setMarks(self, gotList):
    self.__marksCount = len(gotList)
    self.__marks = sum(gotList)

  def getCount(self):
    return self.__marksCount
  def getMarks(self):
    return self.__marks

## holds all questions' outcome from a topic
class topic:
  def __init__(self):
    self.topicMarks = None
    self.correctAns = answerCount()
    self.incorrectAns = answerCount()
    self.NA = answerCount()
```

```
################################
### Result of One Competitor ###
################################

####  Result of One Candidate/Student  ####
class competitor:
  def __init__(self):
    self.grade = ''
    self.bonus = 0

    ## actual marks + pointes based on
    ## right/wrong/not-attempted que.
    self.bonus = 0
    self.marksByTopic = []
    self.topics = []

  def addTopic(self,topicMarkList):
    newTopic = topic()

    newTopic.topicMarks = sum(topicMarkList)
    newTopic.correctAns.setMarks(list(filter(lambda x: (x>0),
topicMarkList)))
    newTopic.incorrectAns.setMarks(list(filter(lambda x: (x<0),
topicMarkList)))
    newTopic.NA.setMarks(list(filter(lambda x: (x==0),
topicMarkList)))

    self.topics.append(newTopic)
    return newTopic.topicMarks

  def setBonus(self):
    bonus = 0
    for index in range(len(self.topics)):
      topic = self.topics[index]
      rights = topic.correctAns.getMarks()
      wrongs = topic.incorrectAns.getMarks()
      left = topic.NA.getMarks()

      bonus += (index+1)*(rights-wrongs)
      if left != 0 and wrongs < left+right:
        bonus += left
    return bonus


  def getTopic(self,topicNum):
    if topicNum in list(range(1,6)):
      return self.topics[topicNum-1]
    return False

  def addResult(self,result):
    for topicMarks in result:
      self.marksByTopic.append(self.addTopic(topicMarks))
    self.bonus = self.setBonus()
    return sum(self.marksByTopic)

  def setGrade(self, g):
    self.grade = g



################################
### Complete Database Holdes ###
################################

## creates a list of marks awarded for every question grouped by
the 5 topics
## creates a list of marks awarded for every question grouped by
the 5 topics
## creates a list of marks awarded for every question grouped by
the 5 topics
class competitiveExam:
  def __init__(self, answerKey = None):
    self.__answerKey = self.__putSolutions(answerKey)
    self.scores = None
    self.competitors = []


######################################################
                 PRIVATE FUNCTIONS
######################################################

  ## stores actual solution
  def __putSolutions(self,ansKey):
    while not ansKey:
```

```
   try:
    ansKey = eval(input("Enter the correct solutions as a
list: "))
   except:
    continue
  return ansKey


 ## marks for every question of curent section
 def __marksInTopicRcrcv(self, answerSheet, init = 0, stop =
None):
  unitTopicQ = len(self.__answerKey)//20
  init *= unitTopicQ
  if stop is None:
   stop = len(answerSheet) - unitTopicQ*19
  else:
   stop *= unitTopicQ

  markList = []
  incorrectAns = 0

  for index in range(init, stop,1):
   if answerSheet[index] == self.__answerKey[index]:
    markList.append(float(1))
   elif answerSheet[index] == None:
    markList.append(float(0))
   else:
    incorrectAns += 1
    markList.append(-incorrectAns/2)

  return markList


 def __topicMarks(self, answerSheet, init, topic):
  topicRes = []
  indxList = [10,14,17,19]

  if topic == 5:
   topicRes.append(self.__marksInTopicRcrcv(answerSheet,
init))
  else:
   stop = indxList[topic-1]
```

```
      topicRes.append(self.__marksInTopicRcrcv(answerSheet,
init, stop))
      topicRes += self.__topicMarks(answerSheet, stop, topic+1)

    return topicRes


  ####    EXAM STATS    ####
  def __getGrade(self, pCent):
    if pCent > 95:
      return 'O'
    elif pCent > 90:
      return 'A+'
    elif pCent > 85:
      return 'A'
    elif pCent > 75:
      return 'B+'
    elif pCent > 65:
      return 'B'
    elif pCent > 55:
      return 'C'
    elif pCent > 45:
      return 'D'
    else:
      return 'F'

  def __findGrade(self, part, total):
    pCent = self.__getPercent(part, total)
    return self.__getGrade(pCent)

  def __getPercent(self, part, total):
    return part*100/total

  def __getPercentile(self, loc):
    gotMarks = self.scores[loc]
    studentsBehind = len(self.scores[self.scores < gotMarks])

    dupMarksLoc = list(np.where(self.scores == gotMarks)[0])
    if len(dupMarksLoc) > 1:
      getWithExtra = lambda x: gotMarks +
self.competitors[x].bonus
      thisWithExtra = getWithExtra(loc)
      dupMarksLoc.remove(loc)
      dupComparision = np.array(list(map(getWithExtra,
dupMarksLoc)))
      studentsBehind += len(dupComparision[dupComparision <
thisWithExtra])

    percentile = studentsBehind*100 / len(self.scores)

    return percentile


######################################################
              PUBLIC FUNCTIONS
######################################################


  # gives a list of the marks for all of the questions
  # grouped by the topic viz.  [[T1], [T2],...., [T5]]
  def addCompetitor(self, answerSheet):
    marklist = self.__topicMarks(answerSheet, 0, 1)

    newCompetitor = competitor()
    finalMarks = newCompetitor.addResult(marklist)


newCompetitor.setGrade(self.__findGrade(finalMarks,len(self.__an
swerKey)))

    self.competitors.append(newCompetitor)
    if self.scores is None:
      self.scores = np.array([finalMarks])
    else:
      self.scores = np.append(self.scores, finalMarks)

  def getCompetitor(self,rNo):
    score = self.scores[rNo]
    competitorData = self.competitors[rNo]
    return (competitorData,score)



######################################################
######################################################
```

```
  def gradingSys(self):
    print("\nMaximum Marks: 
",len(self.__answerKey),"\n"+"-"*20,"\n")
    print("Marks (%) \t\t Grade 
Assigned\n","\n"+"-"*9,"\t\t","-"*15,"\n")
    for percent in [95, 90, 85, 75, 65, 55, 45]:
      print("Above {}% \t\t\t{}".format(percent, 
self.__findGrade(percent+1,100)))
    print("Below or Equals to {}% \t\t{}".format(45, 
self.__findGrade(45,100)))

  def competitionStats(self):
    if len(self.competitors) > 0:
      MM = len(self.__answerKey)
      omitGrade = lambda x: x.grade
      gradelist = list(map(omitGrade, self.competitors))

      gradeDict = {}
      for grd in ['O','A+','A','B+','B','C','D','F']:
        gradeDict[grd] = len(list(filter(lambda x: (x==grd), 
gradelist)))

      #############
      print("""
      Maximum Marks: {}
      Average Marks scored by students: {}
      Toppers Marks: {} \t(scored by {} student(s))
      """.format(MM, sum(self.scores)/MM, max(self.scores),
                 len(list(filter(lambda x: 
(x==max(self.scores)), self.scores)))))

      print("\n\nOverall Outcome of Competition: \n")
      keys, values = gradeDict.keys(), gradeDict.values()
      plt.bar(range(len(values)), values, color='b')
      plt.xticks(range(len(values)), keys)
      plt.show()
    else:
      print("Results will be out soon!! Keep checking the 
portal!")

  def competitorStats(self):
    if len(self.competitors) > 0:
      MM = len(self.__answerKey)
      unitSecMarks = MM//20
      mmByTopic = list(unitSecMarks*i for i in [10,4,3,2,1])

      index = int(input("Enter the roll number (0 to {}): 
".format(len(self.competitors))))
      if index >= len(self.competitors):
        print("Results will be out soon!! Keep checking the 
portal!")
        return False

      thisCompetitor,finalScore = self.getCompetitor(index)
      grades = thisCompetitor.grade
      marksByTopic = thisCompetitor.marksByTopic

      print("""
      Roll No.: {}
      Marks Scored: {}/{}  ({}%)\n
      Grade: {}
      Percentile: {}
      Maximum scored in: Topic-{}   ({} marks)
      Minimum scored in: Topic-{}   ({} marks)
      """.format(index, finalScore, MM, 
self.__getPercent(finalScore,MM),
                 grades, self.__getPercentile(index),
                 marksByTopic.index(max(marksByTopic))+1, 
max(marksByTopic),
                 marksByTopic.index(min(marksByTopic))+1, 
min(marksByTopic)))

      topics = ['Topic-1', 'Topic-2', 'Topic-3', 'Topic-4', 
'Topic-5']
      topicPCent = []

      for i in range(len(marksByTopic)):
        
topicPCent.append(self.__getPercent(marksByTopic[i],mmByTopic[i]
))

      print("\n\nMark Distribution by Topic: \n")
      ## topicPCent can be kept as it is from above for loop
      topicPCent = marksByTopic
      plt.bar(range(len(topicPCent)), topicPCent, color='b')
      plt.xticks(range(len(topicPCent)), topics)
      plt.show()
      return True
    else:
      print("Results will be out soon!! Keep checking the
portal!")
      return False

######################################################
                     DRIVER CODE
######################################################

################          MAIN MENU          ################
def menu():
  print("""
         Main-Menu
        -----------\n
        1) Evaluate an Answersheet
        2) Upload & Evaluate All 100 Answersheets
(autometically)
        3) Overall Exam Stats
        4) Check Your Result
        5) Grading Criteria
        ** Any other key to exit!
        """)
  return input("\nEnter your choice: ")


################        MAIN FUNCTION        ################
if __name__ == '__main__':
  # 20 questions with all ans as 1  ---->  replace 20 by num of
questions
  thisCompetitiveExam = competitiveExam(answerKey = [1]*20)

  while True:
    clear_output()
    ch = menu()

    ## Decision
    if ch == '1':
      clear_output(wait=True)
      ans = [1,0]*10
      if thisCompetitiveExam.addCompetitor(ans):
        print("Graded Successfully!/n")
      input("\nPress Enter!!")

    elif ch == '2':
      clear_output(wait=True)
      ####
      numOfQue = 19071999   ## 100
      ansKey = [1]*numOfQue
      thisCompetitiveExam = competitiveExam(answerKey = ansKey)
      for rNo in range(1000):
        answerSheet = []
        for i in range(numOfQue):
          num = randint(1,1000)%100
          if num%10 == 0:
            answerSheet.append(0)
            #num = ''
          else:
            answerSheet.append(1)
        thisCompetitiveExam.addCompetitor(answerSheet)
      ####
      print("Results are ready!\n")
      input("\nPress Enter!!")

    elif ch == '3':
      clear_output(wait=True)
      thisCompetitiveExam.competitionStats()
      input("\nPress Enter!!")

    elif ch =='4':
      clear_output(wait=True)
      thisCompetitiveExam.competitorStats()
      input("\nPress Enter!!")

    elif ch =='5':
      clear_output(wait=True)
      thisCompetitiveExam.gradingSys()
      input("\nPress Enter!!")

    else:
      clear_output(wait=True)
      del(thisCompetitiveExam)
      print("Good Bye!")
      break
```