# Problem Statement

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 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.

# Solution

## Maximum Number of Questions

We need a maximum number of questions to generate answers for all possible questions. Let us fix the range of date of birth of eligible students to be 01-01-1971 to 31-12-2003.

From this date, the minimum number of questions attempted by any student will be 1011971 and maximum number of questions will be 31122003.

## Categorization of Questions in Reference Answer Key

First we will divide the maximum number of questions to topic categories one to five.

Date corresponding to maximum case would be 31-12-2003

The ratio of questions belonging to each topic category is given as10:4:3:2:1.

For the case where the question number is maximum, that is 31122003 the category wise split up questions would look like the following. This maximum case answer key can be considered as the reference answer key for evaluation.

1. Total number of questions = 31122003 (Maximum case)

* Number of questions in **Category 1** = 15561001.5
* Number of questions in **Category 2** = 6224400.6
* Number of questions in **Category 3** = 4668300.45
* Number of questions in **Category 4** = 3112200.3
* Number of questions in **Category 5** = 1556100.15

2. Now let us round the numbers. This gives the category wise split up as

* Number of questions in **Category 1** = 15561001
* Number of questions in **Category 2** = 6224401
* Number of questions in **Category 3** = 4668300
* Number of questions in **Category 4** = 3112200
* Number of questions in **Category 5** = 1556100

Total questions = 31122002

3. If the total doesn't add up to the actual number of questions attempted, then let us add to or subtract from category 5 to get the correct total.

* Number of questions in **Category 5** = 1556101

Total questions = 31122003

# Answer Key and Response

We can generate answer key for each category based on the maximum case. Using this response to questions can be evaluated.

Answer key can be represented using a matrix whose rows correspond to question numbers and columns correspond to answer options.

A one in the answer key matrix will represent the correct answer and a -1 would represent a wrong answer.

A single row of the answer key matrix will look like the following:

Answer Key Matrix = A

| | a | b | c | d |
|---|---|---|---|---|
| Q_Num_1 | [1 | -1 | -1 | 1] |

Consider question number 1, represented as Q_Num_1.

Here a b c d are the options. Since a question can have multiple correct answers, elements in columns a and b have the value 1.

Elements of column b and c are -1 since they represent the wrong answer.

A similar approach can be followed to store the responses as well.

For example is the response to the above Q_Num_1 can be represented as below. Here the option chosen or marked by the student is represented with 1 and other options are 0.

Response Matrix = R

| | a | b | c | d |
|---|---|---|---|---|
| Q_Num_1 | [1 | 0 | 0 | 0] |

Numpy arrays can be used for both answer key and response matrices.

# Calculation of Marks

Each correct response carry 1 mark. The negative marks increases per wrong answer as a penalty p=0.5*n, where n represents the n'th wrong answer.

1)So if we multiply corresponding elements of the answer key matrix and response matrix and this would give us another matrix, say the marks matrix M.

M = np.multiply(A,R)

2)If we take the sum of all 1's, this would give the marks for correct answers.

positive marks = count of 1's

3)In a similar manner, if we multiply corresponding elements of the answer key matrix and response matrix and count all -1's, this would give the number wrong answer (n).

From this the negative marks can be calculated by:

negative marks = n*(n + 1)/4

This calculation is to be done for each of the 5 categories separately.

# Generation of answer key

We can pre-prepare answer key for each category, based on the maximum number of questions.

Then this category wise answer key can be used to evaluate the response of student.

```
#Category 1


import numpy as np

#generate an answer key with 506002 questions
#Randomly choose an option as correct and represent it using 1
#numpy array for answer key {DS}
A_Cat1 = np.random.randint(2, size=(15561002, 4))

#Mark the remaining options as wrong using -1
A_Cat1[A_Cat1 == 0] = -1

#print anwer key for category 1
#print(A_Cat1)

#Category 2

#generate an answer key with 202401 questions
#Randomly choose an option as correct and represent it using 1
A_Cat2 = np.random.randint(2, size=(6224401, 4))
```

```
#Mark the remaining options as wrong using -1
A_Cat2[A_Cat2 == 0] = -1

#Category 3

#generate an answer key with 151800 questions
#Randomly choose an option as correct and represent it using 1
A_Cat3 = np.random.randint(2, size=(4668300, 4))

#Mark the remaining options as wrong using -1
A_Cat3[A_Cat3 == 0] = -1.

#Category 4

#generate an answer key with 101200  questions
#Randomly choose an option as correct and represent it using 1
A_Cat4 = np.random.randint(2, size=(3112200 , 4))

#Mark the remaining options as wrong using -1
A_Cat4[A_Cat4 == 0] = -1.

#Category 5

#generate an answer key with 50600 questions
#Randomly choose an option as correct and represent it using 1
A_Cat5 = np.random.randint(2, size=(1556101, 4))

#Mark the remaining options as wrong using -1
A_Cat5[A_Cat5 == 0] = -1
```

# Dealing with Response

First we need the DOB to get the number of questions attempted.
Now these  questions are to be divided into 5 categories based on the ratio 10:4:3:2:1

# Catergory Wise Number of Responses

We need to find the category wise number of questions for different date of births. This will help us to generate  sample response matrices for the five categories, for the given date of birth, that is the total number of questions.

```
#Finding category wise questions
def questions_per_cat(n_total,cat):
 q_per_cat = []
 #number of questions in category 1
 n1 = round(n_total * (10/20))
 q_per_cat.append(n1)
```

```
#number of questions in category 2
n2 = round(n_total * (4/20))
q_per_cat.append(n2)

#number of questions in category 3
n3 = round(n_total * (3/20))
q_per_cat.append(n3)

#number of questions in category 4
n4 = round(n_total * (2/20))
q_per_cat.append(n4)

#number of questions in category 5
n5 = round(n_total * (1/20))


#check if category wise numbers add up to n_total
#if not adjust the number in category 5

diff = n_total - (n1 + n2 + n3 + n4 + n5)
if diff != 0:
 n5 = n5 + diff
else:
 n5 = n5
q_per_cat.append(n5)
return int(q_per_cat[cat-1])
```

# Generate Response

Now we need to generate the response matrix for each category and append it with zeros, so as to match the dimensions of answer key matrix.

This will help us to do operations between answer key matrix and response matrix.

```
#Generate response
#numpy array for response {DS}
def gen_response_cat(num_cat,A_cat):
 a = np.random.randint(2, size=(num_cat, 4))
 R = np.zeros(A_cat.shape)
 R[:a.shape[0],:a.shape[1]] = a
 return R
```

# Calculating Marks

To calculate the marks the below steps can be followed:
* Multiply corresponding elements of answer key and response for each category
* From this resultant matrix, count the number of 1's to get marks for correct answers
* Count the number of -1s to get number of wrong answers
* Using the formula negative marks = n*(n + 1)/4, calculate negative marks
* Add marks of right and wrong answers(negative) to get category wise total marks of a student

```
def calc_marks_cat1(A,R):
 M = np.multiply(A,R)
 pos = np.count_nonzero(M == 1)
 n = np.count_nonzero(M == -1)
 neg = n*(n + 1)/4
 cat_total = pos - neg
 return cat_total
```

# StudentExam Class to Bring Together Required Methods

A StudentExam class can be created to consolidate related methods we will be using inorder to get date fo birth, name , responses, calculate marks etc. This will make it easy to conduct exam and to evaluate student responses

```
import os
class StudentExam:
   #constructor
  def __init__(self,name='Student',dob=0):
    self.name = name
    self.dob = dob

   #for student name
  def set_name(self,name):
    self.name = name

   #for student DOB
  def get_dob(self):
    print("Provide DOB in ddmmyyyy format. \n" "If your DOB is 1-Jan-2000, enter 01012000")
    dob = int(input("Enter DOB: "))
    self.dob = dob

  #Category wise question number for given DOB
  def q_num(self):
```

```
  self.n1 = questions_per_cat(self.dob,1)
  self.n2 = questions_per_cat(self.dob,2)
  self.n3 = questions_per_cat(self.dob,3)
  self.n4 = questions_per_cat(self.dob,4)
  self.n5 = questions_per_cat(self.dob,5)

#Response of the student
 def response(self):
  self.R1 = gen_response_cat(self.n1,A_Cat1)
  self.R2 = gen_response_cat(self.n2,A_Cat2)
  self.R3 = gen_response_cat(self.n3,A_Cat3)
  self.R4 = gen_response_cat(self.n4,A_Cat4)
  self.R5 = gen_response_cat(self.n5,A_Cat5)

#Category wise marks of the student
 def marks(self):
  self.cat1 = calc_marks_cat1(A_Cat1,self.R1)
  self.cat2 = calc_marks_cat1(A_Cat2,self.R2)
  self.cat3 = calc_marks_cat1(A_Cat3,self.R3)
  self.cat4 = calc_marks_cat1(A_Cat4,self.R4)
  self.cat5 = calc_marks_cat1(A_Cat5,self.R5)

 #Print student name and category wise marks to a text file
 def print_student_marks(self):
  writepath = 'report.txt'
  mode = 'a' if os.path.exists(writepath) else 'w'
  with open(writepath, mode) as f:
   print(self.name, self.cat1,self.cat2,self.cat3,self.cat4,self.cat5,sep= ",",file=f)
  f.close()
```

# Conduct the Exam

Let us test our code by conductiong a mock exam. Here we will be conducting the test for 10 students. Ten names, DOB and response set will be generated and the test scores for each category will be calculated.

```
#Generate random DOBs and names
import datetime
import random

#Generate random DOB
def gen_dob():
 start_date = datetime.date(1971, 1, 1)
 end_date = datetime.date(2003, 1, 1)
 time_between_dates = end_date - start_date
 days_between_dates = time_between_dates.days
 random_number_of_days = random.randrange(days_between_dates)
 yyyymmdd = str(start_date + datetime.timedelta(days=random_number_of_days))
 ddmmyyyy = yyyymmdd[8:] + yyyymmdd[5:7] + yyyymmdd[:4]
 return int(ddmmyyyy)

#Generate student names
def gen_names(i):
   name = 'Student'+str(i)
   return name
```

```
#Mock Test
#Conduct exam for 10 students
for i in range(10):
 stu = StudentExam(gen_names(i),gen_dob())
 stu.q_num()
 stu.response()
 stu.marks()
 stu.print_student_marks()
```

# Assigning Category Wise Ranks to Students

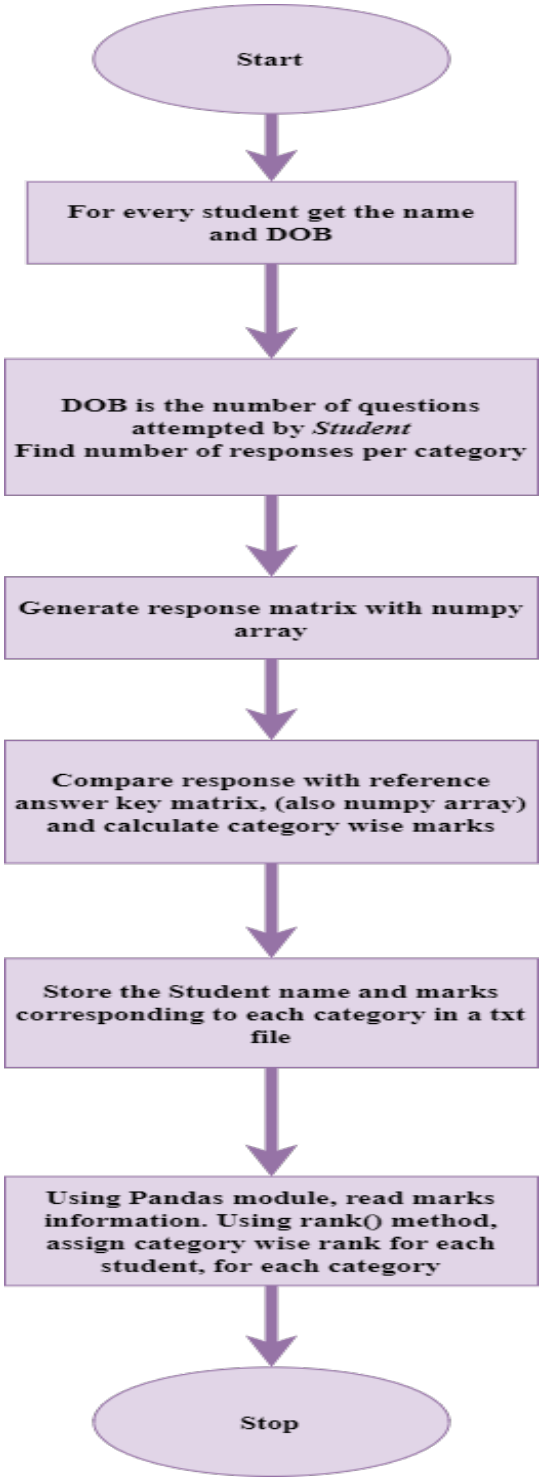
We can use the ranks() method from Pandas to find the rank of students for each category

```
# importing pandas package
import pandas as pd

# making data frame from csv file
df = pd.read_csv('report.txt', header =None)
df.columns = ['Name', 'Category1Marks','Category2Marks','Category3Marks','Category4Marks','Ca
tegory5Marks']
# creating a rank column and passing the returned rank series
df['Category 1 Rank'] = df.Category1Marks.rank(method='dense',ascending=False)
df['Category 2 Rank'] = df.Category2Marks.rank(method='dense',ascending=False)
df['Category 3 Rank'] = df.Category3Marks.rank(method='dense',ascending=False)
df['Category 4 Rank'] = df.Category4Marks.rank(method='dense',ascending=False)
df['Category 5 Rank'] = df.Category5Marks.rank(method='dense',ascending=False)

# sorting w.r.t name column
df.sort_values(by = 'Category1Marks')

# display after sorting w.r.t Name column
df
```

| | Name | Category1Marks | Category2Marks | Category3Marks | Category4Marks | Category5Marks | Category 1 Rank | Category 2 Rank | Category 3 Rank | Category 4 Rank | Category 5 Rank |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | Student0 | -1.823692e+13 | -2.918404e+12 | -1.640962e+12 | -7.289595e+11 | -1.824676e+11 | 8.0 | 8.0 | 8.0 | 8.0 | 8.0 |
| 1 | Student1 | -5.272105e+13 | -8.435483e+12 | -4.740780e+12 | -2.111349e+12 | -5.263731e+11 | 11.0 | 11.0 | 11.0 | 11.0 | 11.0 |
| 2 | Student2 | -6.911470e+10 | -1.106278e+10 | -6.243015e+09 | -2.784119e+09 | -6.987699e+08 | 3.0 | 3.0 | 3.0 | 3.0 | 3.0 |
| 3 | Student3 | -3.605861e+13 | -5.760856e+12 | -3.244656e+12 | -1.441410e+12 | -3.605051e+11 | 10.0 | 10.0 | 10.0 | 10.0 | 10.0 |
| 4 | Student4 | -3.057026e+13 | -4.891490e+12 | -2.749028e+12 | -1.222922e+12 | -3.059365e+11 | 9.0 | 9.0 | 9.0 | 9.0 | 9.0 |
| 5 | Student5 | -6.377442e+12 | -1.020628e+12 | -5.733825e+11 | -2.551768e+11 | -6.377203e+10 | 4.0 | 4.0 | 4.0 | 4.0 | 4.0 |
| 6 | Student6 | -1.237896e+13 | -1.984110e+12 | -1.113096e+12 | -4.956311e+11 | -1.239168e+11 | 7.0 | 7.0 | 7.0 | 7.0 | 7.0 |
| 7 | Student7 | -9.061634e+12 | -1.452089e+12 | -8.143009e+11 | -3.623176e+11 | -9.051147e+10 | 6.0 | 6.0 | 6.0 | 6.0 | 6.0 |
| 8 | Student8 | -5.654029e+13 | -9.050740e+12 | -5.078739e+12 | -2.263429e+12 | -5.654171e+11 | 12.0 | 12.0 | 12.0 | 12.0 | 12.0 |
| 9 | Student9 | -6.407190e+12 | -1.025936e+12 | -5.762567e+11 | -2.563541e+11 | -6.399294e+10 | 5.0 | 5.0 | 5.0 | 5.0 | 5.0 |
| 10 | stud10 | 1.000000e+00 | 2.000000e+00 | 3.000000e+00 | 4.000000e+00 | 5.000000e+00 | 2.0 | 1.0 | 1.0 | 2.0 | 1.0 |
| 11 | stud11 | 2.000000e+00 | 1.000000e+00 | 2.000000e+00 | 6.000000e+00 | 4.000000e+00 | 1.0 | 2.0 | 2.0 | 1.0 | 2.0 |

If required, the students can be sorted based on a particular category rank also.

# Algorithm – High Level View



# Data Structure Used

Numpy Array is used to represent answer key as well as student responses. Numpy arrays have several advantages over usual Python lists.

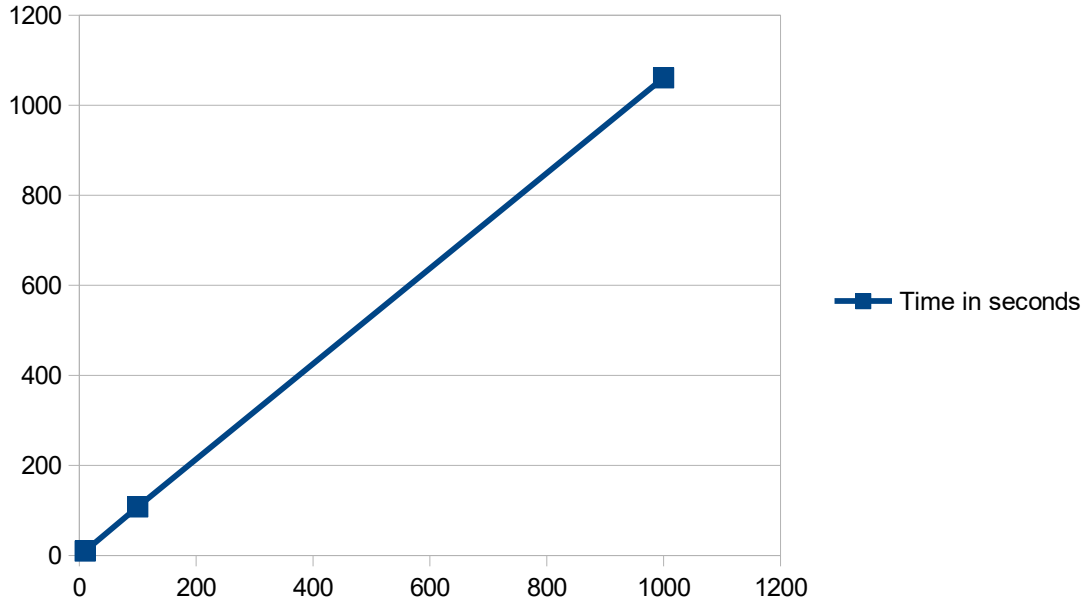
- Smaller memory consumption
- Better runtime behavior
- NumPy has optimized functions such as linear algebra operations built in

# Time Complexity

As mentioned earlier,Numpy Array is used to represent answer key as well as student responses. This will speed up our calculation of marks significantly.

The time taken for category wise marks calculation and then to find category wise rank:

- For 10 students : avg 10 seconds
- For 100 students: avg 107 seconds
- For 1000 students: avg 1061 seconds




From the time values, and graph it is clear that we are dealing with linear time complexity. Hence the time complexity of this approach is O(n).