Programming Assignment

Note:

1. This assignment consists of two sections: Section A and Section B.
2. Solutions to the problems in Section A – will be checked in the Lab (Every Wednesday and Thursday)
3. Solutions to the problems in Section B have to submit in your Github repository. You need to also submit your pseudo-code and time complexity analysis file.
4. **Submission Deadline for Section B: 19th November 2018**
5. A deduction of 1 marks of the maximum mark available from the actual mark achieved by the student shall be imposed upon expiry of the deadline.

6) A further deduction of 1 marks of the maximum mark available from the actual mark achieved by the student shall then be imposed on each of the next subsequent working days;

Section A

Q.1: Given order of n matrices, find the minimum multiplication operations required for multiplying n matrices.

Q.2: From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra's algorithm and bellman's ford algorithm.

Q.3: Implement N Queen's problem using Back Tracking.

Q.4: Suppose you are managing the construction of billboards on the university road, a heavily travelled stretch of road that runs west-east for M miles. The possible sites for billboards are given by numbers $x_1, x_2, \dots\dots\dots x_n$ each in the interval $[0, M]$ (specifying their position along the roads, measured in miles from its western end). If you place billboard at location x_i, you receive revenue of $r_i > 0$.
Regulations imposed by the Delhi road authority require that no two billboards be within less than or equal to 5 $miles$ of each other. You had like to place billboards at a subset of the sites so as to maximize your total revenue, subject to this restriction.
Example: Suppose M=20, n=4

$$\{x_1, x_2, x_3, x_4\} = \{6,7,12,14\} \text{ and } \{r_1, r_2, r_3, r_4\} = \{5,6,5,1\}$$

Then the optimal solution would be to place billboards at x_1 and x_3 for total revenue of 10.
Give an algorithm that takes an instance of this problem as input and returns the maximum total revenue that can be obtained from any valid subsets of sites.

Q.5: Some of your friends have gotten into the burgeoning field of *time-series data mining,* in which one looks for patterns in sequences of events that occur over time. Purchases at stock exchanges --- what's being bought --- are one source of data with a natural ordering in time. Given a long sequence S of such events, your friends want an efficient way to detect certain "patterns" in them --- e.g. they may want to know if the four events

buy Yahoo, buy eBay, buy Yahoo, buy Oracle

occur in this sequence S, in order but not necessarily consecutively. They begin with a finite collection of possible *events* (e.g. the possible transactions) and a sequence S of *n* of these events. A given event may occur multiple times in S (e.g. Yahoo stock may be bought many times in a single sequence S). We will say that a sequence S' is a *subsequence* of S if there is a way to delete certain of the events from S so that the remaining events, in order, are equal to the sequence S'. So for example, the sequence of four events above is a sub-sequence of the sequence

buy Amazon, buy Yahoo, buy eBay, buy Yahoo, buy Yahoo, buy Oracle

Their goal is to be able to dream up short sequences and quickly detect whether they are subsequences of S. So this is the problem they pose to you: Give an algorithm that takes two sequences of events --- S' of length *m* and S of length *n*, each possibly containing an event more than once --- and decides in time O (*m* + *n*) whether S' is a subsequence of S.

Section B

Q.6: Suppose you're consulting for a bank that's concerned about fraud detection, and they come to you with the following problem. They have a collection of *n* bank cards that they've confiscated, suspecting them of being used in fraud. Each bank card is a small plastic object, containing a magnetic stripe with some encrypted data, and it corresponds to a unique account in the bank. Each account can have many bank cards corresponding to it, and two bank cards are equivalent if they correspond to the same account. It's very difficult to read the account number off a bank card directly, but the bank has a high-tech "**equivalence tester**" that takes two bank cards and, after performing some computations, determines whether they are equivalent.
Their question is the following: among the collection of n cards, is there a set of more thann/2 ofthem that are all equivalent to one another? Assume that the only feasible operations you can do with the cards are to pick two of them and plug them in to the equivalence tester. Show how to decide the answer to their question with only *O(n log n)* invocations of the equivalence tester.

Q.7: You are consultant working for a small computation-intensive investment company, and they have the following type of problem they want to solve over and over. A typical instance of the problem is the following. They are doing a simulation in which they look at n consecutive days of a given stock, at some point in the past. Let's number the days $i = 1,2, \dots n$; for each day i, they have a price $p(i)$ per share

for the stock on that day (For simplicity, the price was same during each day). Suppose during this time period, they wanted to buy 1000 shares on some day and sell all these shares on some (later) day. They want to know: When should they have bought and when should they have sold in order to have made as much money as possible. (If there was no way to make money during the n days, you should report this instead).
For example, suppose n = 3, p(1)=9, p(2)= 1, p(3)= 5. Then you should return
"Buy *on day 2 and selling on day 3"* means that they would have made $4 per share. Show how to find the correct numbers i and j in time $O(nlogn)$.

Q.8 : Your friends are starting a security company that needs to obtain licenses for n different pieces of cryptographic software. Due to regulations, they can only obtain these licenses at the rate of at most one per month. Each license is currently selling for a price of $\$100$. However, they are all becoming more expensive according to exponential growth curves: in particular, the cost of license j increases by a factor of r_j greater than 1 each month, where r_j is a given parameter. This means that if license j is purchased t months from now, it will cost $100.r_j(t)$. We will assume that all the price growth rates are distinct; that is $r_i \neq r_j$ for license $i \neq j$(even though they start at the same price of $100).
The question is that: Given that the company can only buy at most one license a month, in which order should it buy the license so that the total amount of money it spends is as small as possible.
Give an algorithm that takes the n rates of price growth $r_{1,}r_2 \dots \dots \dots r_n$ and computes an order in which to buy the license so that the total amount of spent is minimized. The running time of the algorithm should be polynomial in n.

Q.9: Your friend is working as a camp counselor, and he is in charge of organizing activities for a set of junior-high-school-age campers. One of his plans is the following mini-triathlon exercise: each contestant must swim 20 laps of a pool, then bike 10 miles, and then run 3 miles. The plan is to send the contestants out in a staggered fashion, via the following rule: the contestants must use the pool one at a time. In other words, first one contestant swims the 20 laps, gets out, and starts biking. As soon as this first person is out of the pool, a second contestant begins swimming the 20 laps; as soon as he/she's out and starts biking, a third contestant begins swimming... and so on.)

Each contestant has a projected *swimming time* (the expected time it will take him or her to complete the 20 laps), a projected *biking time* (the expected time it will take him or her to complete the 10 miles of bicycling), and a projected *running time* (the time it will take him or her to complete the 3 miles of running). Your friend wants to decide on a *schedule* for the triathalon: an order in which to sequence the starts of the contestants. Let's say that the *completion time* of a schedule is the earliest time at which all contestants will be finished with all three legs of the triathalon, assuming they each spend exactly their projected swimming, biking, and running times on the three parts. (Again, note that participants can bike and run simultaneously, but at most one person can be in the pool at any time.) What's the best order for sending people out, if one wants the whole competition to be over as early as possible? More precisely, give an efficient algorithm that produces a schedule whose completion time is as small as possible.

Q.10: (Planning a party).
Alice wants to throw a party and is deciding whom to call. She has n (which is at least 11) people to choose from and she has made up a list of which pairs of these people know each other. She wants to invite as many people as possible subject to the following two constraints:
1. Every person invited should know at least five other people that are invited.
2. Every person invited should not know at least five other people that are invited.
Design an efficient algorithm for maximizing the number of people she can invite. Remember to analyze the running time and correctness.
Hint: Maximizing the number of invitees is the same as minimizing the number of people Alice doesn’t invite. Obviously Alice might not be able to invite everyone. For example, if one of the n people knows less than five people out of the n potential invitees then the first constraint can never be satisfied for that person.

Q.11: Save Delhi City
Suppose you are **Shaktimaan** and you have decided to do something to save your favorite city (Delhi) against the attack of **Tamraj Kilvish**, since no one else surprisingly seems bothered about it, and are just suffering through various attacks by various different creatures.

Seeing your passion, **N** people of Delhi decided to come forward to try their best in saving their city. Now you have decided to strategize these **N** people into a formation of AT LEAST **K** people in a group. Otherwise, that group won't survive.

Let's demonstrate this by an example. Let's say that there were 10 people, and each group required at least 3 people in it for its survival. Then, the following 5 groups can be made:

- 10 - Single group of 10 members.
- 7, 3 - Two groups. One consists of 7 members, the other one of 3 members.
- 6, 4 - Two groups. One consists of 6 members, the other one of 4 members.
- 5, 5 - Two groups. One consists of 5 members, the other one of 5 members.
- 4, 3, 3 - Three groups. One consists of 4 members, the other two of 3 members.

Given the value of **N**, and **K** - find out the number of ways you can form these groups (anti-squads) to save Delhi city.