CS224W: Analysis of Networks **Fall 2018**

Homework 3

Due 11:59pm PDT November 15, 2018

This problem set should be completed individually.

General Instructions

These questions require thought, but do not require long answers. Please be as concise as possible. You are allowed to take a maximum of 1 late period (see the information sheet at the end of this document for the definition of a late period).

Submission instructions: You should submit your answers via Gradescope and your code via the SNAP submission site. Register for Gradescope at `http://gradescope.com` using your Stanford e-mail (if not SCPD) and include your student ID number with sign-up. Use the entry code **9ZZ2XY** to sign up for CS224W.

Submitting answers: Prepare answers to your homework in a single PDF file and submit it via Gradescope. Make sure that the answer to each sub-question is on a *separate, single page*. The number of the question should be at the top of each page. Please use the submission template files included in the bundle to prepare your submission. Failure to use the submission template file will result in a reduction of 2 points from your homework score.

Information sheet: Fill out the information sheet located at the end of this problem set or at the end of the submission template file, and sign it in order to acknowledge the Honor Code (if typesetting the homework, you may type your name instead of signing). This should be the last page of your submission. Failure to fill out the information sheet will result in a reduction of 2 points from your homework score.

Submitting code: Upload your code at `http://snap.stanford.edu/submit`. Put all the code for a single question into a single file and upload it. Failure to submit your code will result in reduction of all points for that part from your homework score.

Homework survey: After submitting your homework, please fill out the Homework 3 Feedback Form. Respondents will be awarded extra credit.

Questions

1 Spectral clustering and the Stochastic Blockmodel [35 points – Javier]

In this question we will continue our exploration of Spectral Clustering from Homework 2. We will then explore the Stochastic Blockmodel (SBM) covered in Lecture 8 and under which conditions is Spectral Clustering able to detect the community structure induced by the SBM. We will be following some of the ideas from the paper by Christopher Moore,"The Computer Science and Physics of Community Detection: Landscapes, Phase Transitions, and Hardness" `https://arxiv.org/pdf/1702.00467.pdf`.

1.1 Network analysis of political blogs using Spectral Clustering [15 points]

We will begin by using the spectral algorithm derived in parts 4.1 and 4.2 from Homework 2 to cluster the network specified by the file `polblogs.txt`. The `polblogs` network is one where the nodes are political blogs, and there is an undirected edge between two nodes i and j if either blog i links to blog j or blog j links to blog i. The blogs are labeled as liberal (label 0) or conservative (label 1).

Implement the spectral algorithm for normalized cut minimization. Use the algorithm to partition the `polblogs` network into two sets of nodes S and $\bar{S}$. Compute and report the sizes of S and $\bar{S}$ and the modularity of the cluster assignments given by the algorithm.

We've provided you with a second graph (`polblogs-rewired.txt`) that has the same set of nodes but with $1,000$ additional edges added to the graph. Compute and report the sizes of S and $\bar{S}$ and the modularity of the cluster assignments given by the algorithm on the graph in `polblogs-rewired.txt`.

How does the `polblogs-rewired` modularity value compare to that of `polblogs`? Why does this difference occur?

Suggestion: You need to work directly with the adjacency matrix, which we can't do with `snap.py`. We recommend but don't require using `numpy`, and you can use `np.linalg.eigh` to compute the eigenvectors and eigenvalues of the Laplacian. You can find an old `numpy` tutorial from CS231 here.

Sanity Check: When run on the dataset `sample.txt`, the spectral clustering algorithm produces clusters of size 12 and 53 with modularity 0.09.

1.2 The Stochastic Blockmodel [10 points]

As we saw in lecture, we can use a stochastic blockmodel for link prediction. If we are able to infer the latent structure of the blockmodel, we can predict new links within communities or between communities by the probabilities in the estimated blockmodel. The way to estimate the blockmodel from a given graph is by setting priors on the distribution of the model's parameters and iteratively update them via Markov Chain Monte Carlo methods.

Another way to view the stochastic blockmodel, however, is as a generative model. This is what we will explore in this part of the assignment. For simplicity, we will focus on the symmetric stochastic blockmodel (SSBM).

Recall from lecture that in the symmetric case, the probabilities of an edge existing between two nodes only depends on whether or not the nodes are in the same community or if they belong to different communities (see Figure 1). Communities arise in a graph where $\theta_{\text{in}} > \theta_{\text{out}}$. When these values are equal, this simply corresponds to the Erdös-Rényi model. Lastly, when $\theta_{\text{in}} < \theta_{\text{out}}$ it is called disassortative case.

Write code to generate a random graph via a SSBM given the number of nodes n, a distribution on community membership π, and probabilities of within community links θ_{in} and between community links θ_{out}. For each of the following parameter configurations below, generate a graph using your SSBM code, compute its adjacency matrix and visualize it to verify that the block structure is the expected one. Tip: If you have your adjacency matrix stored as a 2D `numpy` array called `adj`, for instance, you can visualize it using `plt.imshow(adj, cmap='binary')` (this is part of the

$$\Pr[(i,j) \in E] = \begin{cases} \theta_{in}. & \text{if } c_i = c_j \\ \theta_{out}. & \text{if } c_i \neq c_j \end{cases}$$

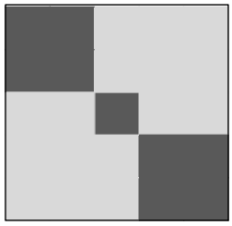

Figure 1: SSBM probabilities and adjacency matrix blockmodel

`matplotlib.pyplot` library).

- $n = 10$, $\pi = [0.5, 0.5]$, $\theta_{\text{in}} = 0.9$, $\theta_{\text{out}} = 0.3$
- $n = 100$, $\pi = [0.1, 0.2, 0.3, 0.4]$, $\theta_{\text{in}} = 0.6$, $\theta_{\text{out}} = 0.15$
- $n = 1000$, $\pi = [0.25, 0.25, 0.25, 0.25]$, $\theta_{\text{in}} = 0.2$, $\theta_{\text{out}} = 0.6$
- $n = 1000$, $\pi = [0.25, 0.35, 0.1, 0.3]$, $\theta_{\text{in}} = 0.7$, $\theta_{\text{out}} = 0.5$

1.3 Recovery requirements [10 points]

In our previous assignment, we implemented a normalized cut minimization algorithm to perform spectral clustering for community detection in graphs. Now that we are able to simulate graphs induced by a SBM, we wish to explore under which conditions is our spectral clustering algorithm able to recover the block structure present in the generated graph.

We will begin by generating a graph with a fairly simple structure using our code from the previous question. Create a graph on $1,000$ nodes, with 2 equally sized communities with a probability of an edge existing within one of the communities of 0.55 and the probability of an edge existing between different communities of 0.5.

From our stochastic blockmodel, we know the first half of the nodes belong to one community and the second half of them belong to the other one. Now, perform spectral clustering on this graph and based on the signs of the values on the Fiedler vector, assign each of the nodes to one of two communities. You may reuse your code from part 1.1 to perform this task. Evaluate the accuracy of this assignment against what we know to be the **true** community assignment. Let C_1 and C_2 represent the ground truth communities we used to define our SBM from. Note $C_1 = \{i \in \mathbb{N} | 1 \leq i \leq 500\}$, and $C_2 = \{i \in \mathbb{N} | 501 \leq i \leq 1,000\}$. Let $\hat{C}_1$ and $\hat{C}_2$ be the communities detected by the spectral clustering algorithm. We define the accuracy of this assignment as follows:

$$\text{acc}(C_1, C_2, \hat{C}_1, \hat{C}_2) = 2\frac{|C_1 \cap \hat{C}_1| + |C_2 \cap \hat{C}_2|}{|C_1| + |C_2|} - 0.5$$

Note that we multiply by 2 and subtract 0.5 from the above expression to get accuracy values in the range $[0, 1]$ since for a totally random community assignment $\frac{|C_1 \cap \hat{C}_1| + |C_2 \cap \hat{C}_2|}{|C_1| + |C_2|}$ will be 0.5 in expectation. Also note that you should account for the fact that the detected communities are expected to be the same up to a permutation. That is, there is no guarantee that the first community detected by your spectral clustering algorithm corresponds to the first block in the stochastic blockmodel. Since we are only dealing with 2 communities, you just need to evaluate

the accuracy of the assignment on the 2 possible permutations and return the highest value. That is, your global accuracy will be:

$$\max\{\text{acc}(C_1, C_2, \hat{C}_1, \hat{C}_2), \text{acc}(C_1, C_2, \hat{C}_2, \hat{C}_1)\}$$

Now, you will explore the theoretical limits on when it is possible to detect the community structure, by varying the probabilities θ_{in} and θ_{out}. Keeping $\theta_{\text{out}} = 0.3$ fixed, sweep over 50 values for θ_{in} within a reasonable range. Decide what range of values to look at based on the properties of the SSBM. For each of the θ_{in} values:

- Generate a graph with the SSBM
- Run your spectral clustering algorithm
- Evaluate the accuracy value

Plot your accuracy values against θ_{in}. Do the same but keeping $\theta_{\text{in}} = 0.7$ fixed and sweeping over 50 values for θ_{out} within a reasonable range. Decide what range of values to look at based on the properties of the SSBM.

Lastly, let's verify that the jump in detection accuracy coincides with the theoretical results for weak and exact recovery. Let $\theta_{\text{in}} = \frac{a}{n}$, and $\theta_{\text{out}} = \frac{b}{n}$. Note that a and b represent the expected number of nodes in each community. In lecture we showed that for exact recovery, we require

$$\frac{a+b}{2} > 1 + \sqrt{ab}$$

whereas for weak recovery we require the signal to noise ratio to be greater than 1, i.e.

$$\frac{(a-b)^2}{k(a+(k-1)b)} > 1$$

Where k denotes the number of communities we wish to detect. In our case, $k = 2$.

For each of your experiments, compute the values of θ_{in} (respectively θ_{out}) which correspond to the weak and exact detection thresholds. Add two vertical lines at these values to each of your plots and label them accordingly.

As a sanity check: one of the accuracy plots should be increasing, while the other one should be decreasing. Our reference implementation for this question takes about 150 seconds to run.

What to submit

Page 1:
- Number of nodes in S and $\bar{S}$ in the clusters found by the normalized cut minimization spectral algorithm on the two graphs.
- Modularity of the cluster assignment in each graph.
- 1-2 sentence analysis of the difference in modularity observed and why it occurred.

Page 2:
- A plot of the adjacency matrix for each of the 4 requested graphs

Page 3:
- The accuracy value of the spectral clustering community detection on the specified graph
- A plot of accuracy against values of θ_{in}, as well as the thresholds for exact and weak recovery
- A plot of accuracy against values of θ_{out}, as well as the thresholds for exact and weak recovery

2 Node2Vec [20 points – Alex Wang]

In this question we'll be getting some hands on experience with node2vec in order to solidify our intuition behind the algorithm. For more details, you can find the research paper on node2vec here.

To complete this problem you will need to clone or download the node2vec repository onto your computer. We recommend you develop in the 'src' folder. Take a look at `main.py` and their README to get a grasp of how to run the node2vec code.

You will also need to install NetworkX.

2.1 Visualizing the Graph [1 points]

Load the 'karate.edgelist' into an undirected graph and create a visualization of the resultant graph (an easy way to do this is with NetworkX). Take a look at node 33. What sort of structural role does it seem to play in the Karate graph?

2.2 DeepWalk [2 Points]

Recall that node2vec works by carrying out a number of random walks from each node in the graph, where the walks are parameterized by p and q. More precisely, after having just traversed the edge from node node t to node v, the unnormalized transition probability of travelling from node v to a neighboring node x is given by:

$$\alpha_{pq}(t,x) = \begin{cases} \frac{1}{p} & d_{tx} = 0 \\ 1 & d_{tx} = 1 \\ \frac{1}{q} & d_{tx} = 2 \end{cases}$$

Where d_{tx} is the shortest path distance between nodes t and x.

DeepWalk is a similar algorithm to node2vec that also learns representations for each node through random walks. However, unlike node2vec, at each node the next neighbor to walk to is chosen uniformly at random.

You can read more about DeepWalk here.

One can think of DeepWalk as a specific case of node2vec. Explain briefly but specifically (1-2 sentences) why this is true, and how one can use node2vec to simulate DeepWalk.

2.3 Finding Clusters [6 Points]

The main advantage of node2vec is its flexibility in the type of similarities it can infer between nodes in a graph.

Without calculating the shortest path distance between any nodes in the Karate graph, give a list of the five nodes with the lowest shortest path distances to node 33 (there may be more than five nodes that are direct neighbors of node 33 - any 5 will do). Explain clearly how you generated this list, how you set p and q, and the reasoning behind your methodology.

Note: We won't be grading the correctness of your outputted nodes, but rather the correctness of the rational behind your methodology – your approach to this question should be clearly and concisely justified..

2.4 Finding Structural Similarity [6 Points]

Given the fact that the node with the highest degree in the Karate graph is node 34, and without calculating the degrees of any other nodes, give a list of the five nodes that are also likely to have moderately high degrees. Explain clearly how you generated this list, how you set p and q, and the reasoning behind your methodology.

2.5 Neighborhood Similarity [5 Points]

Intuitively, node2vec works by creating a representation of a 'neighborhood' around each node through random walks, then generating vector representations of each node that preserve the neighborhood similarity.

Setting $p = 100$ and $q = 0.01$, find, in the Karate graph, the node whose node2vec representation has the closest l_2 (Euclidean) distance from node 33. What are the degrees of node 33 and this node? Is this what you would expect? Give a high level explanation of this result (it may help to refer to your visualization of the Karate graph from part 1).

What to submit

Page 4:
- A plot of the graph
- The structural role node 33 plays (1 sentence)

Page 5:
- An explanation of the relationship between DeepWalk and node2vec (1-2 sentences)

Page 6:
- A list of 5 node IDs
- An explanation and justification of your methodology

Page 7:
- A list of 5 node IDs
- An explanation and justification of your methodology

Page 8:
- The nodeID of the most similar node to node 33 under the given parameters
- The degrees of the above node and node 33
- An explanation of this result (3-4 sentences)

3 Decision-based Cascades: A Local Election [25 points – Shuyang]

It's election season and two candidates, Candidate A and Candidate B, are in a hotly contested city council race in sunny New Suburb Town. You are a strategic advisor for Candidate A in charge of election forecasting and voter acquisition tactics.

Based on careful modeling, you've created two possible versions of the social graph of voters. Each graph has 10,000 nodes, where nodes are denoted by an integer ID between 0 and 9999. The edge lists of the graphs are provided in the homework bundle. Both graphs are **undirected**.

Given the hyper-partisan political climate of New Suburb Town, most voters have already made up their minds: 40% know they will vote for A, 40% know they will vote for B, and the remaining 20% are undecided. Each voter's support is determined by the last digit of their node id. If the last digit is 0–3, the node supports A. If the last digit is 4–7, the node supports B. And if the last digit is 8 or 9, the node is undecided.

The undecided voters will go through a 10-day decision period where they choose a candidate based on the majority of their friends. The **decision period** works as follows:

1. The graphs are initialized with every voter's initial state (A, B, or undecided).
2. In each iteration, every undecided voter decides on a candidate. Voters are processed in increasing order of node ID. For every undecided voter, if the majority of their friends support A, they now support A. If the majority of their friends support B, they now support B. "Majority" for A means that strictly more of their friends support A than the number of their friends supporting B, and vice versa for B (ignoring undecided friends).
3. If a voter has an equal number of friends supporting A and B, we assign support for A or B in alternating fashion, starting with A. In other words, as the voters are being processed in increasing order of node ID, the first tie leads to support for A, the second tie leads to support for B, the third for A, the fourth for B, and so on. This alternating assignment happens at a *global level* for the whole network, across all rounds. (Keep a single global variable that keeps track of whether the current alternating vote is A or B, and initialize it to A in the first round. Then as you iterate over nodes in order of increasing ID, whenever you assign a vote using this alternating variable, change its value afterwards.)
4. When processing the updates, use the values from the current iteration. For example, when updating the votes for node 10, you should use the updated votes for nodes 0–9 from the current iteration, and nodes 11 and onwards from the previous iteration.
5. There are 10 iterations of the process described above.
6. On the 11th day, it's election day, and the votes are counted.

Note that only the undecided voters go through the decision process. The decision process does not change the loyalties of those voters who have already made up their minds.

3.1 Basic Setup and Forecasting [4 points]

Start your work with the starter code provided. Read in the two graphs and assign the initial vote configurations to the network. Then, perform the 10 iterations of the voting process. Which

candidate wins in Graph 1, and by how many votes? Which candidate wins in Graph 2, and by how many votes?

For sanity check, neither of these two numbers (how many votes) is larger than 300.

3.2 TV Advertising [8 points]

You have amassed a substantial war chest of \$9000, and you have decided to spend this money by showing ads on the local news. Unfortunately, only 100 New Suburb Townians watch the local news—those with ids 3000–3099. However, your ads are extremely persuasive, so anyone who sees the ad is immediately swayed to vote for candidate A regardless of his/her previous decision. You may spend \$1000 at a time on ads. The first \$1,000 reaches voters 3000–3009, the second \$1000 reaches voters 3010–3019, and so on. In other words, the total of $\$k$ in advertising would reach voters with ids from 3000 to $3000 + \frac{k}{100} - 1$. This advertising happens before the decision period. *After voters are persuaded by your ads, they never change their minds again.*

Simulate the effect of advertising spending on the two possible social graphs. First, read in the two graphs again and assign the initial configurations as before. Now, before the decision process, you purchase $\$k$ of ads and go through the decision process of counting votes.

For each of the two social graphs, plot $\$k$ (the amount you spend) on the x-axis (for values $k = 1000, 2000, \ldots, 9000$) and the number of votes you win by on the y-axis (that is, the number of votes for A less the number of votes for B). Put these on the same plot. What's the minimum amount you can spend to win the election in each of the two social graphs?

Note that the TV advertising affects all of the voters who see the ads and not just those who are undecided.

3.3 Wining and Dining the High Rollers [8 points]

TV advertising is only one way to spend your campaign war chest. You have another idea to have a very classy \$1000 per plate event for the high rollers of New Suburb Town (the people with the highest degree in the social graph). You invite high rollers in order of how many people they know, and everyone that comes to your dinner is *instantly persuaded to vote for candidate A regardless of his/her previous decision.* For each high roller you spend \$1000 to this persuasion. This event will happen before the decision period. When there are ties between voters with the same degree, the high roller with lowest node ID get chosen first.

Simulate the effect of the high roller dinner on the two graphs. First, read in the graphs and assign the initial configuration as before. Now, before the decision process, you spend $\$k$ on the fancy dinner and then go through the decision process of counting votes.

For each of the two social graphs, plot $\$k$ (the amount you spend) on the x-axis (for values $k = 1000, 2000, \ldots, 9000$) and the number of votes you win by on the y-axis (that is, the number of votes for A less the number of votes for B). What's the minimum amount you can spend to win the election in each of the two social graphs?

Note that wining and dining sways all the voters to vote for A and not just those who are undecided.

3.4 Analysis [5 points]

Plot the degree distributions on a log-log scale of the two graphs on the same plot (as in Question 1.1 on Problem Set 1). Although both graphs have roughly the same number of edges, degree distributions are actually very different. In 1–2 sentences, briefly summarize how this difference explains the results of Question 3.3.

What to submit

Page 9:
- Which candidate wins and by how many votes in each graph.

Page 10:
- Plot of winning margin in each graph as a function of $k (on the same plot)
- The minimum amount you can spend to win the election in each graph

Page 11:
- Plot of winning margin in each graph as a function of $k (on the same plot)
- The minimum amount you can spend to win the election in each graph

Page 12:
- Log-log plot of the degree distributions (on the same plot),
- 1–2 sentences on why the plot explains the results of Question 3.3.

4 Influence Maximization [20 points – Megha]

In class we discussed the influence maximization problem and the greedy hill-climbing approach to solving it. In the algorithm, we add nodes to the current seed set one at a time. At step 0, we have an empty set S_0. At step $i > 0$, we pick the node which maximizes the marginal gain: $S_i = S_{i-1} \cup \{\arg\max_u f(S_{i-1} \cup \{u\}) - f(S_{i-1})\}$, where $f(S)$ denotes the number of nodes influenced by the initially active set S (includes the set S itself).

As we showed in class the hill climbing algorithm cannot guarantee an optimal solution. In other words, there might exist a set T with $|T| = i$ such that $f(S_i) < f(T)$. It is also known that the greedy algorithm is a $(1 - (1 - \frac{1}{k})^k)$ - approximation for the influence maximization problem (its sub-optimality is bounded). In other words, for every $k \geq 1$, the greedy hill-climbing algorithm outputs a set S_k such that $f(S_k) \geq (1 - (1 - \frac{1}{k})^k) f(T)$ where T is the optimal set for influence maximization.

Parts 1 and 2 of this problem ask you to construct examples where the greedy hill climbing returns a sub-optimal solution. In part 2, you will explore examples close to the non-optimal bound for the greedy algorithm for $k = 3$. Your answer should consist of: (1) For every node u its influence set X_u (you can describe the set or draw a directed graph where an edge from A to B indicates that node A influences node B with probability 1), (2) S_i, the set of nodes that a greedy hill climbing would choose after i iterations, and (3) T, the optimal set of i nodes.

For all the questions, you can assume: (1) The nodes in S are influencing themselves, i.e., the count of total influence $f(S)$ includes the nodes in S. (2) The influence set X_u contains all nodes that are influenced by node u, both directly and eventually. (3) When several nodes have the same level of marginal gain, we choose one of them at random.

4.1 Non-Optimal Hill-Climbing [7 points]

For $k = 2$, construct an example where $f(S_k) < f(T)$. That is, hill-climbing will only find a non-optimal solution. (Hint: the last step of the greedy approach is optimal given the $k - 1$ previous steps.)

4.2 Bounded Non-Optimal Hill-Climbing [7 points]

For $k = 3$, construct an example where $f(S_k) \leq 0.8f(T)$. That is, hill-climbing will only find a solution that is at most 80% of the optimal solution.
(Note: 0.8 is very close to the true lower bound for $f(S_k)$ when $k = 3$ which is approximately 0.70)

4.3 Optimality of Hill-Climbing [2 points]

Give a property of influence sets X_u such that $f(S_i) = f(T)$. In other words, what is a *sufficient* property of influence sets of nodes such that greedy hill-climbing always outputs the optimal solution? The property does not need to be a necessary one. It must be a property of X_u. Properties such as "the network has only i nodes" are not valid as correct answers.

There are several correct answers; we will accept all reasonable answers.

4.4 More Hill-Climbing... [4 points]

Assume that we stop hill-climbing after k steps and $|S_k| = |T| = k$. Recall that in the class we proved a bound in the form of

$$f(T) \leq f(S_k) + \sum_{i=1}^{k} \delta_i, \tag{1}$$

where $\delta_1, .., \delta_k$ are the largest k values of $\{f(S_k \cup \{u\}) - f(S_k)|u\}$ (See the course slides). Construct a family of examples for which $f(S_k) + \sum_{i=1}^{k} \delta_i - f(T)$ can be arbitrarily large.

To be more specific, given any number b you should exhibit a network (graph) such that $f(S_k) + \sum_{i=1}^{k} \delta_i - f(T) > b$.

Note: A *family* of examples is a set F of examples such that for any number b, there exists a network (graph) $E(b) \in F$ (corresponding to b) such that $f(S_k) + \sum_{i=1}^{k} \delta_i - f(T) > b$.

What to submit

Page 13:
- Submit an example: X_u, S, T.

Page 14:
- Submit an example: X_u, S, T.

Page 15:
- Submit a property and a brief explanation.

Page 16: • Submit a family of examples and a brief explanation.

Information sheet
CS224W: Analysis of Networks

Assignment Submission Fill in and include this information sheet with each of your assignments. This page should be the last page of your submission. Assignments are due at 11:59pm and are always due on a Thursday. All students (SCPD and non-SCPD) must submit their homeworks via Gradescope (http://www.gradescope.com). Students can typeset or scan their homeworks. Make sure that you answer each (sub-)question on a separate page. That is, one answer per page regardless of the answer length. Students also need to upload their code at http://snap.stanford.edu/submit. Put all the code for a single question into a single file and upload it. Please do not put any code in your Gradescope submissions.

Late Homework Policy Each student will have a total of *two* free late periods. *Homework are due on Thursdays at 11:59pm PDT and one late period expires on the following Monday at 11:59pm PDT.* Only one late period may be used for an assignment. Any homework received after 11:59pm PDT on the Monday following the homework due date will receive no credit. Once these late periods are exhausted, any assignments turned in late will receive no credit.

Honor Code We strongly encourage students to form study groups. Students may discuss and work on homework problems in groups. However, each student must write down their solutions independently i.e., each student must understand the solution well enough in order to reconstruct it by him/herself. Students should clearly mention the names of all the other students who were part of their discussion group. Using code or solutions obtained from the web (github/google/previous year solutions, etc.) is considered an honor code violation. We check all the submissions for plagiarism. We take the honor code very seriously and expect students to do the same.

Your name: _____

Email: _____ **SUID:** _____

Discussion Group: _____

I acknowledge and accept the Honor Code.

(Signed) _____