Some notes for Math 103

CHAPTER 1: **Plurality method:** The candidate with the most first-place votes wins. **Borda count method:** If there are N candidates, each first-place vote counts N points, each second place vote counts $N-1$ points, ..., each Nth place vote counts 1 point. The candidate with the most points wins. **Plurality-with-elimination method:** The candidate(s) with the fewest first place votes is eliminated. Then the preference table is revised and the procedure is repeated. The last remaining candidate is the winner. **Method of pairwise comparisons:** For each pair of candidates X and Y, X is awarded 1 point if a majority of voters prefer X over Y, $\frac{1}{2}$ point if X and Y are tied, and 0 points if a majority of voters prefer Y over X. The candidate with the most points wins.

Majority criterion: A candidate with a majority of the first place votes should win. **Condorcet criterion:** A Condorcet candidate is one prefered by a majority of voters over any other candidate. A Condorcet candidate should win. **Monotonicity criterion:** If candidate X would win an election and if some votes are changed in favor of X, then candidate X should still win the election. **Independence of Irrelevant Alternatives criterion:** If candiate X would win an election and if another candidate drops our, candidate X should still win the resulting election.

Ranking methods: **Extended plurality method** - rank candidates in order of the number of first place votes. **Extended Borda count method** and **Estended method of pairwise comparisons** - rank candidates in order of the number of points. **Extended plurality with elimination method** - rank candidates in the reverse of the order in which they are eliminated. **Recursive methods** - use any method to determine the winner. Then eliminate the winner and use the method to determine second place, etc.

CHAPTER 2: A **weighted voting scheme** with players $P_1, ..., P_N$ is given by an $n+1$-tuple $[q, w_1, ..., w_n]$ where $(w_1 + ... + w_N)/2 < q \leq w_1 + ... + w_N$ and $w_1 \geq w_2 \geq ... \geq w_N$. Player P_i has w_i votes and q votes are required to pass a motion. A **coalition** is a subset of $\{P_1, ..., P_N\}$. A coalition is **winning** (respectively, **losing**) if the sum of the weights of its members is $\geq q$ (respectively, $< q$). A player is **critical** for a coalition if that coalition is a winning coalition, but the coalition obtained by removing that player is losing. The **Banzhaf power index** of player P_i is the number of times P_i is critical divided by the number of times any player is critical. A **sequential coalition** is an ordering $P_{i_1}, ..., P_{i_N}$ of the N players. Player P_{i_j} is **pivotal** if $\{P_{i_1}, ..., P_{i_{j-1}}\}$ is a losing coalition but $\{P_{i_1}, ..., P_{i_j}\}$ is winning. The **Shapley-Shubik power index** of player P_i is the number of times P_i is pivotal divided by $N!$.

CHAPTER 3: **Divider chooser method:** Divider divides object into two pieces of equal value and the chooser chooses one. **Lone divider method** (for $N = 3$): Divider divides object into three pieces of equal value. The two choosers list the pieces they would find acceptable. If at least two pieces are listed, it is possible to give everyone an acceptable piece. If not, one piece which is not listed is given to the divider, the two remaining pieces are combined and then divided by the two choosers (using the divider chooser method). **Lone chooser method** (for arbitrary N): The $N-1$ dividers divide the object into $N-1$ pieces of equal value. Each divider divides its piece into N pieces of equal value. The chooser takes one of these pieces from each divider. **Last diminisher method:** The players play in a fixed order, dropping out when they get a piece of the object. The first player cuts a piece of the object of size $\frac{1}{N}$. Each successive player may pass or may reduce the size of the piece (diminish it). If no player diminishes the piece, the player who cut the piece gets it. If at least one player diminishes it, the last diminsher gets the piece. This continues until only two players remain. They then use the divider chooser method to finish. **Method of sealed bids:** Players bid for every item. Each item is awarded to the highest bidder (who must pay the price bid for it). Each player is paid for its "fair share", that is the total amount it bid divided by N. The surplus amount collected over the amount paid is divided equally among the players. **Method of markers:** Objects labelled $1, 2, ..., K$ are arranged in a row. Each of N players indicates N intervals that they regard as equal value by placing $N-1$ markers, one at the end of each of the first $N-1$ intervals. The interval from 1 to the left most first marker is awarded to the player who placed that marker. That player's other markers are removed and the player with the leftmost second marker is awarded their second interval. This continues until every player is awarded an interval. Some items may be left over.

CHAPTER 4: The **standard divisor** is the population of the entire country divided by the number of representatives. The **standard quota** for a state is the population of the state divided by the standard divisor. The **standard lower quota** is obtained by rounding down, the **standard upper quota** is obtained by rounding up. **Hamilton's method:** Assign each state its standard lower quota, then assign the remaining representatives one at a time to the states with the largest fractional parts. **Jefferson's method:** Find a number D, the **modified divisor**, such that if the **modified quota** (the population of the state divided by the modified divisor) is rounded down and the results summed, the result is the desired number of representatives. **Adams' method:** Replace "down" by "up" in Jefferson's method. **Webster's method:** Replace rounded down by the usual rounding procedure in Jefferson's method. **Quota rule:** The number of representatives assigned to a state should be the standard lower quota or the standard upper quota.

CHAPTER 5: A **graph** is a picture consisting of dots, called **vertices**, and lines, called **edges**. Each edge connects two vertices or connects a vertex to itself (in which case the edge is called a **loop**). The **degree** of a vertex is the number of edges at that vertex (with a loop counting twice towards the degree). The sum of the degrees of all of the vertices

of a graph is equal to twice the number of edges. Consequently, the number of vertices of odd degree is even. Two vertices are **adjacent** if there is a edge joining them. Two edges are **adjacent** if they share a common vertex. A path is a sequence of distinct edges, each of which is adjacent to the next. (A vertex can appear on a path more than once, but an edge can be part of a path only once.) A **circuit** is a path that starts and ends at the same vertex. A graph is **connected** if any two of its vertices can be joined by a path. Otherwise it is **disconnected**. A **bridge** in a connected graph is an edge such that the graph obtained by removing it is disconnected. An **Euler path** in a graph is a path which contains every edge of the graph. An **Euler circuit** in a graph is a circuit which contains every edge of the graph. A graph has an Euler circuit if and only if it is connected and every vertex has even degree. A graph has an Euler path if and only if it is connected and there are at most two vertices of odd degree (hence either 0 or 2 vertices of odd degree). **Fleury's algorithm** gives a procedure for finding an Euler circuit in a connected graph in which every vertex has even degree: Start at any vertex and travel along any edge subject to the condition that you do not travel over a bridge for the untraveled part of the graph unless there is no alternative. Stop when you cannot travel any further. You will have returned to your starting point and completed an Euler circuit. Any connected graph may be **Eulerized** by adding copies of existing edges in such a way that all vertices of the resulting graph have even degree.

CHAPTER 6: A **Hamilton circuit** in a graph is a circuit which contains each vertex exactly once. A graph may or not have a Hamilton circuit. The **complete graph on N vertices**, denoted K_N is the graph with N vertices and with no loops in which every two vertices are connected by exactly one edge. K_N has $(N(N-1))/2$ edges and $(N-1)!$ Hamilton circuits (where we regard two circuits that differ only in their starting points as the same). A **weighted graph** is a graph in which each edge has been labeled with some number. The **weight of a path** in a weighted graph is the sum of the weights of the edges of the path. An **optimal Hamilton circuit** in a complete weighted graph is a Hamilton circuit of smallest possible weight. We have studied algorithms for finding Hamilton circuits. The **brute force algorithm** consists of listing all Hamilton circuits, computing the weight of each and choosing the circuit of minimal weight. This produces an optimal Hamilton circuit, but is an **inefficient algorithm** in the sense that, for large values of N, the number of computations involved is so large that the computation cannot be completed in a reasonable amount of time. There are **approximate algorithms** which can be implemented in a reasonable amount of time, but do not necessarily produce the optimal Hamilton circuit. The **nearest neighbor algorithm** starts at any vertex, follows the edge of smallest weight from that vertex to another vertex, then follows the edge of smallest weight from that vertex to a vertex which has not been visited yet. The algorithm continues in this way until all vertices have been visited. Finally, one returns to the original vertex. The **repetitive nearest neighbor algorithm** consists of applying the nearest neighbor algorithm starting at each vertex of the graph and choosing the shortest Hamilton circuit produced in this way. The **cheapest link algorithm** consists of choosing the edge of smallest weight in the entire graph, then choosing the remaining edge of smallest weight, then choosing the remaining edge of smallest weight which does not complete a circuit or produce a vertex of degree greater than 2. One then continues in this way until $N-1$ edges have been chosen. At this point you will have created a path containing all N vertices. Connect the ends of this path to form a Hamilton circuit.

CHAPTER 7: If G is a graph, a **subgraph** of G is obtained from G by removing (or erasing) some edges and some vertices, with the condition that if a vertex is removed all of the edges connected to it must also be removed. A **tree** is a connected graph with no cycles. If a tree has N vertices, then it must have $N-1$ edges. If G is a graph, a **spanning tree** for G is a subgraph of G which is a tree and which contains all of the vertices of G. In a weighted graph, the **weight of a subgraph** is the sum of the weights of the edges of the subgraph. A **minimal spanning tree** for a connected weighted graph is a spanning tree of minimal weight. **Kruskal's algorithm** produces a minimal spanning tree for any connected graph G: Take the edge of lowest weight, then take the edge of second lowest weight, then take the remaining edge of lowest weight which does not create a circuit. Continue in this way until you have chosen $N-1$ edges (where G has N vertices). The result is a minimal spanning tree. The minimal spanning tree gives the shortest network connecting the vertices of G if the network must follow edges of the original graph. If new vertices can be added, it is possible to find shorter networks by adding **Steiner points**. A Steiner point X in a triangle with vertices A, B, and C is such that the angles $< AXB, < BXC$, and $< CXA$ are all 120 degrees. A triangle has a Steiner point in its interior if and only if all of its angles are less than 120 degrees. If X is a Steiner point for triangle ABC, then the sum of the lengths of XA, XB, and XC is minimal for any point in the interior of the triangle.

CHAPTER 8: In **scheduling a process**, M **tasks** are assigned to N **processors**, denoted $P_1, P_2, ..., P_N$. If task X must be completed before task Y can be started we write $X \rightarrow Y$. In this case we say that X **is incident to** Y and that Y **is incident from** X**.** This is called a **precedence relation**. Two tasks are said to be **independent** if there is no precedence relation between them. The **directed graph** (or **digraph**) of a process has vertices corresponding to each of the tasks and two additional vertices: **Start** and **End**. The vertices corresponding to tasks are connected by arrows corresponding to the precedence relations and arrows are drawn from **Start** to all tasks which are not incident from any other task and also from any task which is not incident to any other task to **End**. Each task is labeled with the time required to complete the task: thus the label $A(7)$ implies that it takes 7 units of time to complete task A. A

priority list for a process consists of an ordered list of tasks. Thus if there are M tasks, there are $M!$ priority lists. At any time each task is either **ineligible** if it is incident from some task which is not yet completed, **ready** if it is not ineligible and it has not yet been begun, **in process** if some processor has begun the task, or **completed**. Given the digraph of a process, a priority list, and the number of processors, the process can be scheduled: whenever a processor is free the highest priority ready task (if there is any ready task) is assigned to it. There are two natural algorithms for choosing a priority list. The **decreasing time algorithm** chooses the priority list which lists the tasks in decreasing order of required time. Thus the task requiring the longest time is listed first, then the task requiring the second longest time, and so on until the task requiring the shortest time is listed last. To describe the second algorithm, we must first define the **critical path** for a vertex X. This is the path from X to **End** that has the longest total processing time; this total processing time is called the **critical time** for X. Critical times may be computed (and recorded on the digraph of the process) by using the **backflow algorithm**: give **End** critical time [0]. Then for each vertex incident to **End** give that vertex critical time equal to the processing time for that vertex plus the critical time for **End** (which is [0]). Continue in this way: whenever critical times have been established for every vertex incident form a vertex X, give X critical time equal to its processing time plus the maximum of the critical times for the vertices incident from X. Once the critical times have been computed, the **critical path algorithm** chooses the priority list which lists the tasks in decreasing order of critical time.

CHAPTER 11: There are four types of **rigid motions** of the plane: **reflections, rotations, translations,** and **glide reflections.**

If A is a point in the plane and l is a line, the image of A under the reflection with **axis** l is the point A_1 such that l is the perpendicular bisector of the line segment joining the points A and A_1. A reflection is an **improper** motion, that is, it reverses left and right (or, equivalently, clockwise and counter-clockwise). A reflection is completely determined by its axis and the axis can be found if any pair of points A and A_1 such that A_1 is the image of A is known - the axis is the perpendicular bisector of the line segment joining A and A_1.

A rotation is determined by a point O, called the **rotocenter** and an angle (for example, 60 degrees clockwise). The image of a point A is obtained by drawing the line segment AO (connecting the points A and O), and then drawing another line segment A_1O of the same length such that the angle $< AOA_1$ is the given angle (60 degrees clockwise in our example.) If a point A has image A_1 under a reflection, then the perpendicular bisector of the line segment joining A and A_1 passes through the rotocenter of the rotation. Thus a rotation may be determined from two pairs of points A, A_1 and B, B_1 such that A_1 is the image of A and B_1 is the image of B. Rotations are proper motions, that is they preserve left and right (or, equivalently, clockwise and counter-clockwise). Rotation by 0 degrees (about any rotocenter) leaves every point in the plane fixed. We call this the **identity** motion.

A translation moves every point in a figure a fixed distance in a fixed direction. The distance and direction given by a **vector** which may be represented by an arrow. Two parallel arrows of the same length that are pointed in the same direction represent the same vector. Translations are proper motions. A translation is determined by giving a single pair of points A, A_1 such that A_1 is the image of A - the vector that describes this translation is the vector from A to A_1.

A glide reflection is a translation followed by a reflection in an axis parallel to the vector describing the translation. Glide reflections are improper motions. If the image of a point A under a glide reflection is A_1, then the midpoint of the line segment joining A and A_1 is on the axis of the reflection. Thus if two pairs of points A, A_1 and B, B_1 such that A_1 is the image of A and B_1 is the image of B are given, it is possible to determine the axis of the reflection and hence to determine the glide reflection.

A **symmetry** of an object or a shape is a rigid motion that moves the object back onto itself.

A square has 8 symmetries, four rotations and four reflections. More generally, a regular N-gon has $2N$ symmetries, N rotations and N reflections. (We call this D_N symmetry.) In fact, if a shape has symmetries that are reflections and symmetries that are rotations, then it has the same number each. A circle has infinitely many symmetries that are reflections (any line through the center may be an axis) and infinitely many rotations (you may rotate about the center through any angle). We say that a circle has D_∞ symmetry. A shape may have only rotation symmetries. If such a shape has N rotation symmetries, we say that it has Z_N symmetry. Thus a shape with no symmetry except the identity is said to have Z_1 symmetry.

A **pattern** is an infinite shaped made up of one or more infinitely repeating themes. A one-dimensional pattern is called a **border pattern**. We will usually write these horizontally (although the pattern could be arranged in any direction). Any border pattern has one basic translation symmetry. It may have (i) no reflection symmetry, (ii) only horizontal reflection symmetry, (iii) only vertical reflection symmetry, or (iv) both horizontal and vertical reflecti;on symmtetry. It may have either one rotation symmetry (the identity - rotation by 0 degrees) or two rotation symmetries (rotation by 0 degrees and rotation by 180 degrees). It may have either no glide reflection symmetry or one basic glide reflection symmetry (where the axis must be a line along the center of the pattern). There are 7 possible symmetry types of border patterns.

We call two-dimensional patterns **wallpaper patterns**. A wallpaper pattern must have translation symmetry in at

least two different (i.e., nonparallel) directions. It may have no reflection symmetries or may have reflection symmetries in $1, 2, 3, 4$ or 6 nonparallel directions. (There are no other possibilities.) It must have at least one rotation symmetry (the identity - rotation by 0 degrees). There may be also be 2 rotations (0 and 180 degrees), 3 rotations ($0, 120$ and 240 degrees), 4 rotations ($0, 90, 180, 270$ degrees), or 6 rotations ($0, 60, 120, 180, 240, 300$ degrees). There may be no glide reflections or glide reflections in $1, 2, 3, 4$, or 6 nonparallel directions. There are 17 possible symmetry types of wallpaper patterns.

CHAPTER 12: We have investigated several **recursive replaecment rules** that define sequences of shapes. For such a rule, there is a starting shape and a sequence of steps, each obtained from the previous step by applying a replacement rule. For example, for the **Koch snowflake** the starting shape is an equilateral triangle and the replacement rule is to divide each line segment on the boundary of the shape into thirds, leave the first and last thirds alone and replace the middle third by the other two sides of an equilateral triangle (directed outward).