Problem 1: Quicksort with High Probability [25 points, 5 each]. We have examined the quicksort algorithm multiple times. The algorithm has a worst case running time of $O(n^2)$, but we saw in recitation that when we choose a pivot at random, it has an expected running time of $O(n \log(n))$. This problem will lead you through extending this probabilistic analysis to prove that quicksort runs fast with high probability rather than just in expectation. Here is the pseudocode for the algorithm with a random pivot. You are given an array $a$, and begin by calling quicksort with $l = 0$ and $h = \text{length}(a) - 1$. Note that quicksort is in place, so we don’t actually return a new array. It also uses a Partition helper function which we will need to define.

Consider a “lucky” pivot in round $j$ to be one which splits the subarray $a^{(j)}$ at round $j$ such that the maximum size of a subarray passed in the recursive quicksort calls is $(3/4)|a^{(j)}|$. That is, neither subarray is larger than $3/4$ the size of the previous subarray array. Note that we have a “lucky” round with probability $1/2$.
(a) Consider a specific element \( x \) of the input array to quicksort. In how many lucky rounds can \( x \) appear? 

[hint: if \( x \) is in a lucky round, then the size of the partition in which it appears shrinks by at least a factor of 3/4].

(b) Let \( m_x \) be this maximum number from part (a). What is the expected number of rounds before \( x \) participates in \( m_x \) lucky rounds?

(c) Call the expectation in part (b) \( \mu_x \). We will use a form of a concentration bound to prove that the number of rounds in which \( x \) participates is close to \( \mu_x \) with high probability. The form we use is simplified and specific to random coin tosses with probabilities 1/2 for each event, as is the case here. Use the fact that for \( M \) coin tosses with 0,1 outcomes, the probability that the number of ones is smaller than \( M/4 \) is at most \( e^{-M/8} \) to argue that the probability that in \( 4\mu_x \) rounds, \( x \) has fewer than \( m_x \) “lucky” rounds is less than or equal to \( 1/n^3 \).

(d) Now we take the union bound over all elements \( x \) of the input. Formally, for events \( A_1, \ldots, A_n \), the union bound is

\[
\Pr[A_1 \text{ OR } A_2 \text{ OR } \ldots \text{ OR } A_n] \leq \sum_{i=1}^{n} \Pr[A_i]
\]

Use this fact to show that the recursion tree has height no greater than \( 4\mu_x \) with probability at least \( 1 - 1/n^2 \). [hint: recursion tree of quicksort will have depth greater than \( 4\mu_x \) only if at least one \( x \) participates in more than \( 4\mu_x \) recursive calls]

(e) Finally, assuming that \( \Theta(n) \) work is done at each level of the recursion tree for quicksort, conclude that quicksort runs in \( O(n \log n) \) time with high probability (at least \( 1 - 1/n^2 \)) from the above observations.

**Problem 2: Minimum \((s,t)\text{-Cut} \) [20 points, 10 each].** You are given an undirected graph \( G = (V,E) \) with edge weights \( w_e \) for every \( e \in E \). An \((s,t)\)-cut is a partition of the graph into two sets of vertices \( X \) and \( V \setminus X \). The capacity of an \((s,t)\)-cut \( X \) is the sum of edge weights crossing that cut (that is, with exactly one endpoint in \( X \)). The minimum \((s,t)\)-cut is the cut with the minimum capacity.

(a) Write an integer program to compute the minimum \((s,t)\)-cut in \( G \).

(b) Relax your program from part a to a linear program, and take it’s dual. Interpret your result as an LP for the the maximum flow problem.

**Problem 3: Hall’s Theorem (Taken from DPV 7.30) [20 points points].** Suppose we have a bipartite graph \( G = (A \cup B, E) \) with a set \( A \) of employers and a set \( B \) of programmers. \( A \cap B = \emptyset, |A| = |B| \), and there is an edge between an employer \( a \in A \) and a programmer \( b \in B \) if \( b \) is qualified for the position with employer \( a \). Every employer wants to hire exactly one qualified programmer, and every programmer wants to be hired by exactly one employer for whose position they are qualified. Hall’s theorem says that this is possible (that is, there exists a perfect matching in the graph) if and only if for every subset \( X \subseteq A \), there are at least \( |X| \) vertices in \( B \) with at least one edge to some vertex in \( X \). (The set of such vertices in \( B \), each of which has at least one edge to some vertex in \( X \), is often called the neighborhood of \( X \), abbreviated \( N(X) \)).

Prove this theorem [hint: use the max-flow min-cut theorem].

**Problem 4: Generalized Max Flow (Taken from DPV 7.20) [15 points points].** You are given a directed network \( G = (V,E) \) with edge capacities \( c_e \) for every \( e \in E \). Instead of a single \((s,t)\) pair, you are given multiple pairs \((s_1,t_1),(s_2,t_2),\ldots,(s_k,t_k)\), where the \( s_i \) are sources of \( G \) and the \( t_i \) are sinks of \( G \). You are also given \( k \) demands \( d_1,d_2,\ldots,d_k \). The goal is to find \( k \) flows \( f^{(1)},f^{(2)},\ldots,f^{(k)} \) with the following properties:

- Every \( f^{(i)} \) is a valid flow from \( s_i \) to \( t_i \).
- For each edge \( e \), the total flow across that edge is less than its capacity, that is, \( f^{(1)}_e + f^{(2)}_e + \ldots + f^{(k)}_e \leq c_e \).
• Each flow routes at least its demand, that is, \( f(i) \geq d_i \).

• The total flow is maximized, that is, \( f^{(1)} + f^{(2)} + \ldots + f^{(k)} \) is maximized.

Can you formulate this problem as a simple max flow problem by transforming the graph? Can you formulate this problem as a linear program? Explain your answer (and give the formulation if you can).