Problem 1: (Weighted MINCUT and MAXCUT)

Let \( G(V, E) \) be an undirected weighted graph, with \( w_{ij} > 0 \) the weight associated with every edge \((i,j) \in E\). The weight of a cut \((C, \overline{C})\) is now the sum of the weights of edges across the cut, i.e., \( \delta(C, \overline{C}) = \sum_{i,j \in E(C, \overline{C})} w_{ij} \). We now try and extend our MAXCUT and MINCUT algorithms to this setting.

Part (a)

Let \( W = \sum_{(i,j) \in E} e_{ij} \) be the total weight of all edges in the graph. Modify the MAXCUT algorithm presented in class to return a cut \((C, \overline{C})\) with expected weight satisfying:

\[
\mathbb{E}[\delta(C, \overline{C})] \geq \frac{W}{2}
\]

Part (b)

Next suppose we modify the CONTRACT algorithm to pick edges proportional to their weights. Show that any minimum weight cut \((C, \overline{C})\) is returned by CONTRACT with probability \( \geq 2n(n-1)/n \).

Problem 2: (Recursive Randomized Selection)

Given a unsorted array \( S = \{x_1, x_2, \ldots, x_n\} \), with corresponding sorted array \( \{y_1, y_2, \ldots, y_n\} \), a selection algorithm is one that finds the median element \( y_{n/2} \) (or more generally, the \( k \)-th largest element \( y_k \) for any \( k \in \{1, 2, \ldots, n\} \). One way to do so is by first sorting the array, and then returning \( y_k \) for any \( k \) – this takes time \( O(n \log n) \). However, consider the following simple randomized algorithm to find \( y_k \) for a given \( k \):

QUICKSELECT\((S, k)\)
- Given array \( S \) of \( n \) elements, we want to output the \( k \)-th largest element \( y_k \).
- Choose a random pivot \( \sigma \), and partition \( S \) into two parts:
  \[
  S_\ell = \{y_i \in S | y_i < \sigma\}, \quad S_h = \{y_i \in S | y_i > \sigma\}
  \]
- If \( |S_\ell| = k - 1 \), return \( \sigma \)
- If \( |S_\ell| > k \), then run QUICKSELECT\((S_\ell, k)\); else run QUICKSELECT\((S_h, k - |S_\ell| - 1)\)

It is easy to see that this will find \( y_k \) – we now want to show that QUICKSELECT has a running time of \( O(n) \).

Part (a)

To build some intuition as to why this works, assume in given an array \( S \) of size \( n \), the two arrays \( S_\ell, S_h \) were guaranteed to be of size at most \( \alpha n \), for some \( \alpha \in [1/2, 1) \). Argue that the runtime of QUICKSELECT would then obey: \( T(n) = T(\alpha n) + O(n) \). Solve this to show \( T(n) = O(n) \).

Part (b)

Given any array of size at most \( n \), argue that after splitting about the pivot, the sets \( S_\ell \) and \( S_h \) both have size less than \( 3n/4 \) with probability at least \( 1/2 \). Using this, find an upper bound on
the expected number of times an array of size $n$ needs to be split about random pivots before the sub-array containing $y_k$ is of size $\leq 3n/4$.

**Hint:** Consider an alternate algorithm, where you pick a pivot, check to make sure that both $S_L$ and $S_H$ are less than $3|S|/4$, and then split – if not, you keep the array $S$ as before and again pick a random pivot. Prove the above result for this modified algorithm. Convince yourself that QUICKSELECT can only be faster.

**Part (c)**

Let’s define the algorithm to run in *phases*, where in phase $i$, the size of the sub-array containing $y_k$ is between $(3/4)^{j-1}n$ and $(3/4)^j n$. Also let $X_j$ denote the number of splits required in phase $j$ (so for example, $X_1$ is the expected number of splits required to go from the original array $S$ to one of size $3n/4$).

Argue that $T(n) \leq \sum_{\text{phase } j} c(3/4)^{j-1} n X_j$ for some constant $c$. Finally, via linearity of expectation, prove that $E[T(n)] = O(n)$.

**Problem 3: (Multi-stage MINCUT Algorithm)**

In class we saw the CONTRACT Algorithm for finding the MINCUT of a multigraph $G$ – we were given that each run of CONTRACT took time $O(n^2)$, and argued that if $G$ had a unique minimum cut $(C, \overline{C})$, then CONTRACT finds it with probability $\Omega(1/n^2)$.

**Part (a)**

Suppose CONTRACT returned $(C, \overline{C})$ with probability at least $1/n^2$ – show that $n^2 \ln 2$ independent runs of CONTRACT are sufficient to find cut $(C, \overline{C})$ with probability at least $1/2$.

More generally, convince yourself that if an algorithm is successful with probability at least $p$, then $\ln 2/p$ independent runs are sufficient to guarantee success with probability at least $1/2$.

**Hint:** Use $(1 - x) \leq e^{-x}$.

**Part (b)**

The above problem shows that the overall runtime of CONTRACT is $O(n^4)$ – on the other hand, we learnt in class that the best deterministic MINCUT algorithm had a runtime of $O(n^3)$. We also saw that if we ran CONTRACT until the number of vertices in the multigraph is $t$, then it takes time $O(n^2)$ (as long as $t = o(n)$) and preserves the minimum cut $(C, \overline{C})$ with probability $O(t^2/n^2)$.

Now consider running CONTRACT until the number of vertices in the multigraph is $t$, followed by a deterministic MINCUT algorithm for the $t$-node graph – as before, we can do this multiple times to improve the probability. Show that the best possible choice of $t$ results in a running time of $O(n^{8/3})$ for finding $(C, \overline{C})$ with probability at least $1/2$. 


Problem 4: (The FASTCUT Algorithm and the Branching Process)

Recall that in class, we briefly saw the FASTCUT algorithm, where given a graph, we first ran two independent executions of CONTRACT, stopping them when the resulting subgraph retained the minimum cut with probability $\geq 1/2$, and then proceeded recursively. We now try and understand why this algorithm works.

Part (a)
Assume we can choose $\alpha$ such that contracting the graph to $t = \alpha n$ nodes ensures that a minimum cut is preserved with probability exactly $1/2$ – let us call this the $\alpha$-CONTRACT step. Also assume the original graph $G$ had a unique minimum cut $(C, \overline{C})$.

Now suppose in the first recursive step, we do 2 independent runs of $\alpha$-CONTRACT on the original graph $G$, and at each recursive step, we do 2 independent runs of $\alpha$-CONTRACT for each input sub-graph. After $k$ recursions (where $k \in \{1, 2, \ldots, \log_{1/\alpha} n\}$), what is the expected number of sub-graphs which retain the minimum cut $(C, \overline{C})$?

Part (b)
Suppose instead of doing 2 independent runs of $\alpha$-CONTRACT on each subgraph, we instead ran it once, and just duplicated the resulting subgraph. Now what is the expected number of sub-graphs which retain the minimum cut $(C, \overline{C})$ after $k$ recursions? Why do you think this is different from part (a)?

Part (c)
Let $p(k)$ be the probability that the minimum cut $(C, \overline{C})$ survives in at least one subgraph if we stop after doing $k$ recursions (thus $p(0) = 1$).

Argue that in the procedure in part (b) – where we do one run of $\alpha$-CONTRACT for each subgraph and duplicate the output – the function $p(k)$ obeys $p(k + 1) = \frac{p(k)}{2}$, and thus $p(k) = 1/2^k$.

On the other hand, argue that the procedure in part (a) – where we do two independent runs of $\alpha$-CONTRACT for each subgraph – the function $p(k)$ obeys $p(k + 1) = 1 - \left(1 - \frac{p(k)}{2}\right)^2$.

Part (d)

(OPTIONAL) Try to show that the solution to the recursive equation $p(k + 1) = 1 - \left(1 - \frac{p(k)}{2}\right)^2$ obeys $p(k) = \Theta(1/k)$.

Hint: Note that $p(k) = \Theta(1/k)$ is same as saying $c_1/k \leq p(k) \leq c_2/k$ – now substitute this in the above recursive equation, and prove it holds by induction.