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 then graph. Modify the MAXCUT algorithm presented in class to return a cut $(C, \overline{C})$ with expected weight satisfying: $\mathbf{E}[\delta(C, \overline{C})] \geq \frac{W}{2}$

Solution: For every edge $(i, j) \in E$, let X_{ij} denote the indicator r.v. that is 1 if $(i, j) \in E(C, \overline{C})$. Then, we have $\delta(C, \overline{C}) = \sum_{(i,j)\in E} w_{ij} \cdot X_{ij}$, and thus by linearity of expectation, $\mathbf{E}[\delta(C, \overline{C})] = \sum_{(i,j)\in E} w_{ij} \cdot \mathbf{E}[X_{ij}]$. Now to compute $\mathbf{E}[X_{ij}] = \mathbf{P}[X_{ij} = 1]$, we use the principle of deferred decisions. For any edge (i, j), suppose we first assign node i to either C or $\overline{C}$ – then in order for (i, j) to be in the cut, we must assign node j to the other set, which happens with probability $1/2$. Thus, we get that $\mathbf{E}[\delta(C, \overline{C})] = \sum_{(i,j)\in E} w_{ij} \cdot 1/2 = W/2$.

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 $\geq \frac{2}{n(n-1)}$.

Solution: Suppose we are given any multigraph G with n nodes and with a minimum cut $(C, \overline{C})$ with weight w_k. Then for any vertex of the G, we have that $d(v) \geq w_k$ (else it is smaller than the minimum cut, which is a contradiction). This implies that the total weight satisfies $W \geq nw_k/2$. Using this, we have that:

$$\mathbf{P}[\text{Randomly picked edge lies in } E(C, \overline{C})] \leq \frac{w_k}{nw_k/2} = \frac{2}{n}.$$

Returning to the CONTRACT algorithm, in order for the minimum cut $(C, \overline{C})$ to be preserved, we require that no edge in $E(C, \overline{C})$ is picked in the $n - 2$ random edge selections. Thus, we have:

$$\begin{aligned}
\mathbf{P}[\text{CONTRACT}(G, 2) \text{ preserves } (C, \overline{C})] &\geq \Pi_{i=0}^{n-3} \left(1 - \frac{2}{n-i}\right) \\
&= \left(\frac{n-2}{n}\right)\left(\frac{n-3}{n-1}\right)\left(\frac{n-4}{n-2}\right)\cdots\left(\frac{2-1}{2+1}\right) \\
&= \frac{2}{n(n-1)}
\end{aligned}$$

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 $y_{\frac{n}{2}}$ (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 σ, and partition S into two parts:

$$S_\ell = \{y_i \in S | y_i < \sigma\} \quad , \quad S_h = \{y_i \in S | y_i > \sigma\}$$

- If $|S_\ell| = k - 1$, return σ
- If $|S_\ell| > k$, then run QUICKSELECT(S_ℓ, 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_ℓ, S_h were guaranteed to be of size at most αn, 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 $T(n) = O(n)$.

Solution: Our algorithm splits the array into two parts which have size at most $n-1$. Therefore, $\lceil \frac{n}{2} \rceil \leq max(S_l, S_h) < n$. Hence, the maximum execution time, α, will be in $[1/2, 1)$. Since, partitioning S into S_l and S_h requires $O(n)$ time, we have $T(n) = T(\alpha n) + O(n)$, which yields to geometric series:

$$T(n) = T(\alpha n) + O(n) = O(n) + O(\alpha n) + O(\alpha^2 n) + O(\alpha^3 n) + \cdots = O(\frac{1}{1-\alpha} n) = O(n).$$

Part (b)

Given any array of size at most n, argue that after splitting about the pivot, the sets S_ℓ 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.

Solution: Note that, for any k, $\mathbf{P}(|S_l| = k) \geq \frac{1}{n}$. Therefore,

$$p = \mathbf{P}(|S_l|, |S_k| \leq \frac{3n}{4}) = \mathbf{P}(\frac{n}{4} \leq |S_l| \leq \frac{3n}{4}) \geq \frac{\frac{3n}{4} - \frac{n}{4}}{\frac{1}{n}} = \frac{1}{2}.$$

Let N be the number of partitions an array of size n needs to be split about random pivots before the sub-array containing y_k is of size $\leq 3n/4$. Since $N \sim Geom(p)$, we have $\mathbf{E}[N] = \frac{1}{p} \geq 2$.

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 $\mathbf{E}[T(n)] = O(n)$.

Solution: First, note that QUICKSELECT uses $\leq cn$ operations outside of the recursive call for some constant $c > 0$. By the definition of the *phase*, $(3/4)^{j-1}n$ is an upper bound on the array size during phase j. Also, $c(3/4)^{j-1}n$ is the amount of work that we do on each phase j sub-problem, therefore $c(3/4)^{j-1}n \cdot X_j$ is the amount of work in phase j overall. Hence, $T(n) \leq \sum_{\text{phase } j} c(3/4)^{j-1}nX_j$.
Using the fact that $\mathbf{E}[X_j] = 2$,and the linearity of expectation, we get:

$$\mathbf{E}[T(n)] \leq \mathbf{E}[\sum_{\text{phase } j} c(3/4)^{j-1}nX_j] = cn \sum_{\text{phase } j} (3/4)^{j-1}\mathbf{E}[X_j] = 2cn \sum_{\text{phase } j} (3/4)^{j-1} \leq 2cn\frac{1}{1-3/4} = 8cn.$$

So, $\mathbf{E}[T(n)] = O(n)$.

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 $(1 - x) \leq e^{-x}$.

Solution: The probability of not finding cut $(C, \overline{C})$ after k runs is $(1 - \frac{1}{n^2})^k$. Using the hint, we get:

$$\left(1 - \frac{1}{n^2}\right)^k \leq \left(e^{-\frac{1}{n^2}}\right)^k = e^{-\frac{k}{n^2}} \leq \frac{1}{2} \Rightarrow k \geq n^2 ln2.$$

Similarly, in a more general case,

$$(1-p)^k \leq e^{-kp} \Rightarrow k \geq ln2/p.$$

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 $\Theta(t^2/n^2)$.

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 $1/2$.

Solution: Suppose we run CONTRACT until the number of vertices in the multigraph is t (for some $t = o(n)$ we choose later), and then use a deterministic min-cut algorithm. The total time for this is $O(n^2 + t^3) = O(\max\{n^2, t^3\})$. Moreover, the probability that this returns the true min-cut is $\Theta(t^2/n^2)$ – thus in order to ensure the min-cut is found with probability $1/2$, we need to repeat $O(n^2/t^2)$ times. Thus the total running time is $O((n^2/t^2)\max\{n^2, t^3\}) = O(\max\{n^4/t^2, n^2 t\})$. To minimize this, we set $t = \Theta(n^{2/3})$, which gives us a total running time of $O(n^{8/3})$.

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 α 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 α-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 α-CONTRACT on the original graph G, and at each recursive step, we do 2 independent runs of α-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 $(C, \overline{C})$?

Solution: Suppose we number all the subgraphs at the k^{th} recursive step as $\{1, 2, \ldots, 2^k\}$. Let X_i be an indicator r.v. for whether the i^{th} subgraph retains the minimum cut. Then clearly $\mathbf{E}[X_i] = 1/2^k$ (since it has to survive k successive independent runs of α-CONTRACT, one at each recursive step). By linearity of expectation, we see that:

$$\mathbf{E}[\text{Sub-graphs retaining } (C, \overline{C}) \text{ after } k \text{ recursions}] = 1$$

Part (b)

Suppose instead of doing 2 independent runs of α-CONTRACT on each subgraph, we instead ~~ran it once, and just duplicated the resulting subgraph.~~ run it k times, and then make 2^k copies of the resulting subgraph. Now what is the expected number of sub-graphs which retain the minimum cut $(C, \overline{C})$? Why is this way of doing it different from part (a)?

Solution: Suppose again we number all the subgraphs at the k^{th} recursive step as $\{1, 2, \ldots, 2^k\}$, and define X_i to be the indicator r.v. for whether the i^{th} subgraph retains the minimum cut. Then again $\mathbf{E}[X_i] = 1/2^k$, and by linearity of expectation, we have $\mathbf{E}[\text{Sub-graphs retaining } (C, \overline{C})] = 1$.

The point of this question is to show that although getting the correct expectation for a random variable (typically for the problems we study, runtime/probability of success) is necessary for good algorithm design, it is far from sufficient. Although the procedure in parts (a) and (b) gave the same number of expected sub-graphs preserving the min-cut, the procedure in part (b) has a much higher variance in the number compared to part (a). What we often require is a more detailed understanding of the r.v. in question – this is what we do in the next part, and more generally, using tail inequalities.

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 α-CONTRACT for each subgraph and duplicate the output~~ where we do 2^k runs of α-CONTRACT and then make 2^k copies of the resulting sub-graph – 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 α-CONTRACT for each subgraph – the function $p(k)$ obeys $p(k+1) = 1 - \left(1 - \frac{p(k)}{2}\right)^2$.

Solution: The first recursion is easy to see – it corresponds exactly to doing k successive independent runs of α-CONTRACT. For the second recursion (for the procedure in part (a)), observe that the probability of starting with a graph and *not preserving* the minimum cut after $k+1$ recursions is the same as not preserving it along both of the independent sub-graphs created at the first step of the recursion. However, for either of these subgraphs, the min-cut is preserved with probability $\frac{1}{2} \cdot p(k)$. Thus we get $p(k+1) = 1 - \left(1 - \frac{p(k)}{2}\right)^2$.

Note: The problem in the homework had another way of duplicating subgraphs in part (b), wherein we duplicated each subgraphs once at each stage. That procedure also ends up having the same expected number of subgraphs at each stage, but does not obey the recursion for the probability of survival given in part (c). In particular, it satisfies the recursion $p(k+1) = \frac{1}{2} - \frac{1}{2}(1-p(k))^2$, *which also satisfies* $p(k) = \Theta(1/k)$ *– one way to see this is that this procedure is identical to running* α*-CONTRACT once on the original graph, and then switching to the procedure in part (a).*

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.*

Solution: See lecture notes.