Question 1: Minimum Cuts in a Tree

Problem: How many different minimum cuts are there in a tree with n nodes (i.e., $n-1$ edges)?

Solution: A tree is defined as a connected acyclic graph. A tree with n nodes has exactly $n-1$ edges.

1. **Min-Cut Size:** By definition, removing any single edge from a tree disconnects the graph into exactly two connected components. Therefore, every edge defines a cut of size 1. Since a connected graph cannot have a cut of size 0, the minimum cut size is exactly 1.
2. **Distinctness:** Each of the $n-1$ edges connects a unique pair of subtrees. Removing edge e_i creates a partition (A_i, B_i) that is distinct from the partition created by removing any other edge e_j.
3. **Counting:** Since every edge corresponds to a distinct minimum cut, and there are $n-1$ edges, there are exactly $n-1$ minimum cuts.

$$\text{Total Min Cuts} = \text{Number of Edges} = n-1$$

Answer: $n-1$

Question 2: Probability in Karger's Algorithm

Problem: Let $p = \frac{1}{\binom{n}{2}}$. Analyze the probability $Pr[\text{out} = (A, B)]$ relative to p.

Solution: The fundamental analysis of Karger's Random Contraction Algorithm provides a lower bound for the success probability. Let k be the size of the minimum cut. Since the min-cut is k, the degree of every vertex is at least k, meaning the graph has at least $nk/2$ edges. The probability that a specific minimum cut (A, B) survives the contraction process is:

$$Pr[\text{out} = (A, B)] \geq \frac{2}{n(n-1)} = \frac{1}{\binom{n}{2}} = p$$

This inequality holds for **every** minimum cut in **every** graph.

Therefore, the correct statement is: "For every graph G with n nodes and every min cut (A, B) of G, $Pr[\text{out} = (A, B)] \geq p$."

Question 3: Randomized Selection Probability

Problem: What is the probability that after one iteration of Randomized Select, the subarray size is $\leq \alpha n$ (where $0.5 < \alpha < 1$)?

Solution: Let the sorted elements of the array be $z_1, z_2, \ldots, z_n$. The algorithm picks a pivot p uniformly at random. To ensure the remaining subarray has size at most αn, the pivot must partition the array such that neither the left side nor the right side is larger than αn. This occurs if the pivot falls in the "middle" of the sorted sequence.

- **Bad Pivot (Left):** If p is among the smallest $(1-\alpha)n$ elements, the right partition will be larger than αn.
- **Bad Pivot (Right):** If p is among the largest $(1-\alpha)n$ elements, the left partition will be larger than αn.

The "Good Pivot" range is the indices between $(1-\alpha)n$ and αn.

$$\text{Size of Good Range} = \alpha n - (1-\alpha)n = (2\alpha - 1)n$$

Since the pivot is chosen uniformly from n elements:

$$Pr(\text{Good Pivot}) = \frac{(2\alpha - 1)n}{n} = 2\alpha - 1$$

Answer: $2\alpha - 1$

Question 4: Recursion Depth of RSelect

Problem: If every recursive call reduces the input size to at most an α fraction ($0 < \alpha < 1$), what is the maximum recursion depth?

Solution: Let N_k be the input size at depth k. We are given the recurrence inequality:

$$N_k \leq \alpha \cdot N_{k-1}$$

Unrolling this from the initial size n:

$$N_k \leq n \cdot \alpha^k$$

The recursion terminates when the problem size reduces to 1 (base case).

$$1 = n \cdot \alpha^k$$

Taking the logarithm of both sides:

$$\log(1) = \log(n \cdot \alpha^k)$$

$$0 = \log(n) + k \log(\alpha)$$

$$-k \log(\alpha) = \log(n)$$

$$k = -\frac{\log(n)}{\log(\alpha)}$$

(Note: Since $\alpha < 1$, $\log(\alpha)$ is negative, so the depth is positive).

Answer: $-\frac{\log n}{\log \alpha}$

Question 5: Global Min Cut via s-t Min Cut

Problem: Minimum number of calls to an s-t min cut subroutine needed to find the global min cut.

Solution: The global minimum cut partitions the vertices V into two disjoint sets, S and $V \setminus S$.

1. We fix an arbitrary vertex s. This vertex must belong to one of the two partitions (without loss of generality, assume $s \in S$).
2. The other partition $V \setminus S$ is non-empty, so there exists at least one vertex $t \in V \setminus S$.
3. We iterate through all possible candidate vertices for t (all $v \in V, v \neq s$).
4. By computing the Minimum $s - t$ Cut for every $t \in V \setminus \{s\}$, we are guaranteed that at least one iteration will have t in the correct opposing set ($V \setminus S$).
5. For that specific pair, the min $s - t$ cut will correspond exactly to the global min cut.

Since we fix s and iterate through all other nodes, the number of calls is $n - 1$.

Answer: $n - 1$

Optional Problem 1: Lower Bound for Randomized Sorting

Problem: Prove that the worst-case expected running time of every randomized comparison-based sorting algorithm is $\Omega(n \log n)$.

Solution:

To prove this, we utilize **Yao's Minimax Principle**, which allows us to establish a lower bound on the worst-case expected cost of a randomized algorithm by analyzing the average-case cost of a deterministic algorithm over a specific input distribution.

1. **Definitions:**
   - Let $\mathcal{A}$ be a randomized comparison-based sorting algorithm. We can view $\mathcal{A}$ as a probability distribution over a set of deterministic algorithms $\{D_1, D_2, \dots\}$.
   - Let $T(D, I)$ be the running time (number of comparisons) of deterministic algorithm D on input I.
   - We are looking for the worst-case expected running time: $\max_I E[T(\mathcal{A}, I)]$.
2. **Yao's Minimax Principle:** Yao's principle states that for any probability distribution $\mathcal{P}$ over the set of inputs:
   $$\max_I E_{\mathcal{A}}[T(\mathcal{A}, I)] \geq \min_D E_{I \sim \mathcal{P}}[T(D, I)]$$
   In words: The worst-case expected time of the best randomized algorithm is at least the average-case time of the best deterministic algorithm for the "worst" input distribution.
3. **Choosing the Input Distribution:** Let $\mathcal{P}$ be the uniform distribution over all $n!$ possible permutations of the input array.
4. **Deterministic Lower Bound:** Consider any deterministic comparison-based sorting algorithm D. It can be modeled as a decision tree.
   - The tree must have at least $n!$ leaves (one for each permutation) to correctly distinguish all inputs.
   - For a binary tree with L leaves, the average depth (average path length from root to leaf) is lower bounded by $\Omega(\log L)$.
   - Substituting $L = n!$, the average depth is $\Omega(\log(n!))$.
   - Using Stirling's approximation ($\log(n!) = \Theta(n \log n)$), the average running time of any deterministic algorithm D under the uniform distribution is $\Omega(n \log n)$.
5. **Conclusion:** Since $\min_D E_{I \sim \mathcal{P}}[T(D, I)] = \Omega(n \log n)$, by Yao's principle, the worst-case expected running time of any randomized algorithm is also $\Omega(n \log n)$.

Optional Problem 2: Deterministic Selection with Different Group Sizes

Problem: Analyze the running time of the deterministic selection algorithm (Median-of-Medians) if we use group sizes of 7 and 3.

Part A: Group Size of 7

Let $T(n)$ be the running time on an input of size n. The algorithm steps are:

1. Divide elements into $\lceil n/7 \rceil$ groups of 7. Find the median of each group. (Time: $O(n)$)
2. Recursively find the median of these medians, call it x. (Time: $T(n/7)$)

3. Partition the original array around x. (Time: $O(n)$)
4. Recurse on the appropriate subarray. (Time: $T(k)$, where k is the max subarray size)

Calculating the Recurrence:

- The number of groups is $\approx \frac{n}{7}$.
- The pivot x is the median of the $\frac{n}{7}$ group medians.
- Therefore, x is greater than $\frac{1}{2} \cdot \frac{n}{7} = \frac{n}{14}$ group medians.
- In each of these $\frac{n}{14}$ groups, there are 4 elements (the median itself and 3 elements greater than it) that are definitely $\geq x$.
- Total elements guaranteed $\geq x$: $4 \times \frac{n}{14} = \frac{2n}{7}$.
- Similarly, total elements guaranteed $\leq x$: $\frac{2n}{7}$.
- In the worst case, the algorithm recurses on the remaining elements: $n - \frac{2n}{7} = \frac{5n}{7}$.

The recurrence is:

$$T(n) \leq T\left(\frac{n}{7}\right) + T\left(\frac{5n}{7}\right) + O(n)$$

We check if the sum of fractions is less than 1:

$$\frac{1}{7} + \frac{5}{7} = \frac{6}{7} < 1$$

Since the sum is strictly less than 1, the running time is **linear**, i.e., $O(n)$.

Part B: Group Size of 3

Using the same logic for groups of 3:

- Number of groups: $\frac{n}{3}$.
- Pivot x is greater than $\frac{1}{2} \cdot \frac{n}{3} = \frac{n}{6}$ medians.
- In each such group, there are 2 elements (median and 1 greater) definitely $\geq x$.
- Total elements guaranteed $\geq x$: $2 \times \frac{n}{6} = \frac{n}{3}$.
- Max elements in recursive step: $n - \frac{n}{3} = \frac{2n}{3}$.

The recurrence is:

$$T(n) \leq T\left(\frac{n}{3}\right) + T\left(\frac{2n}{3}\right) + O(n)$$

Sum of fractions:

$$\frac{1}{3} + \frac{2}{3} = 1$$

Since the sum equals 1, this recurrence behaves like Merge Sort (specifically $T(n) = T(n/3) + T(2n/3) + cn$). The work per level is $O(n)$, and there are $\log n$ levels. Thus, the running time is $O(n \log n)$, not linear.

Optional Problem 3: Linear Time Weighted Median

Problem: Compute all weighted medians in $O(n)$ worst-case time.

Solution:

We can adapt the standard deterministic selection algorithm. The key insight is that we can discard a portion of the array if we know the weighted median cannot be inside it, provided we account for the weights of the discarded elements.

Algorithm: `WeightedSelect(Array A, TargetWeight T)`

1. **Base Case:** If $|A|$ is small (e.g., 1 or 2), solve by brute force.
2. **Pivot Selection:** Use the deterministic Median-of-Medians algorithm to find the *unweighted* median of A, call it p. This takes $O(|A|)$ time.
3. **Partition:** Partition A around p into three sets:
   - $L = \{x \in A \mid x < p\}$
   - $E = \{x \in A \mid x = p\}$
   - $G = \{x \in A \mid x > p\}$
4. **Compute Weights:** Calculate the total weight of each set: $W_L = \sum_{x \in L} w_x$, $W_E = \sum_{x \in E} w_x$, $W_G = \sum_{x \in G} w_x$.
5. **Recurse:**
   - **Case 1:** If $W_L > T$, the weighted median must be in L.
     Recurse: `WeightedSelect(L, T)`
   - **Case 2:** If $W_L + W_E \geq T$, then p is the weighted median (or one of them).
     Return p.
   - **Case 3:** If $W_L + W_E < T$, the weighted median is in G. However, we effectively "removed" $W_L + W_E$ weight from the left. We must adjust the target.
     Recurse: `WeightedSelect(G,` $T - (W_L + W_E)$`)`

Complexity Analysis: Since we use the median-of-medians as the pivot p, we guarantee that $|L| \leq \frac{7}{10}n$ and $|G| \leq \frac{7}{10}n$ (approx). The recurrence is:

$$T(n) \leq T(0.7n) + O(n)$$

This geometric series sums to $O(n)$. Thus, the algorithm runs in **O(n)** worst-case time. To handle the initial call, we set $T = W_{total}/2$.

Optional Problem 4: Minimum Cuts in Directed Graphs

Problem: Is it true that every directed graph has only polynomially different minimum cuts? Prove it or give a counterexample.

Solution:

No, this is not true for directed graphs. A directed graph can have exponentially many global minimum cuts.

Counterexample: Consider a simple directed graph G with $n = k + 2$ vertices: a source s, a sink t, and k intermediate nodes $v_1, v_2, \ldots, v_k$.

- Add edges $s \to v_i$ with capacity 1 for all $i = 1 \ldots k$.
- Add edges $v_i \to t$ with capacity 1 for all $i = 1 \ldots k$.
- Add a back-edge $t \to s$ with capacity ∞.

Analysis:

1. A global minimum cut partitions vertices into $(S, \bar{S})$.
2. Due to the infinite edge $t \to s$, we cannot choose a cut where $t \in S$ and $s \in \bar{S}$.
3. Thus, valid finite cuts must have $s \in S$ and $t \in \bar{S}$.
4. For every intermediate node v_i, we have a choice:
   - Put $v_i \in S$: The edge $v_i \to t$ is cut (cost 1).
   - Put $v_i \in \bar{S}$: The edge $s \to v_i$ is cut (cost 1).
5. Regardless of the choice for each v_i, the cut size is exactly $1 \times k = k$.
6. Since there are k intermediate nodes and 2 choices for each, there are 2^k distinct cuts of size k.
7. Since $k = n - 2$, the number of minimum cuts is 2^{n-2}, which is **exponential** in n.

Optional Problem 5: Number of α-Minimum Cuts

Problem: How many α-minimum cuts can an undirected graph have? Prove the best upper bound.

Solution:

The upper bound is $O(n^{2\alpha})$.

Proof using Karger's Algorithm: Let k be the size of the minimum cut. An α-minimum cut has size at most αk.

1. In a graph with min-cut k, the minimum degree is at least k. Thus, the total number of edges $m \geq \frac{nk}{2}$.
2. Consider a specific α-minimum cut C with size $|C| \leq \alpha k$.
3. In Karger's contraction algorithm, when the graph has r vertices remaining, the probability of choosing an edge in C to contract is:

$$P(\text{edge in } C) = \frac{|C|}{\text{Edges remaining}} \leq \frac{\alpha k}{rk/2} = \frac{2\alpha}{r}$$

4. The probability that C survives this step is $1 - \frac{2\alpha}{r}$.
5. The algorithm stops contraction when 2α vertices remain (generalized version). The probability that C survives from n vertices down to 2α vertices is:

$$P(\text{survive}) \geq \prod_{r=2\alpha+1}^{n} \left(1 - \frac{2\alpha}{r}\right)$$

6. This product approximates to:

$$\approx \frac{\binom{2\alpha}{2\alpha}}{\binom{n}{2\alpha}} = \frac{1}{\binom{n}{2\alpha}} \approx n^{-2\alpha}$$

7. Since each run of the algorithm produces the specific cut C with probability at least $n^{-2\alpha}$, and the events for different cuts are disjoint in a single run output, the maximum number of such cuts is the reciprocal of the probability.

Result: There are at most $O(n^{2\alpha})$ cuts of size $\alpha \times$ min-cut.