A probability problem: We toss a series of coins independently. Each coin comes up heads with probability \( q \) and tails with probability \( 1-q \).

Q: What is the probability that we need at least \( t \) tosses to get \( k \) heads? Consider \( t \geq 2k \) and \( q \leq \frac{1}{2} \).

A: The probability that we have not achieved \( k \) heads in \( t \) tosses (for \( t > k \)) is:

\[
\sum_{i=k+1}^{t} \binom{t}{i} q^{i-1} (1-q)^{t-(i-1)}
\]

If we call this event \( A \) then:

\[
P(A) = \sum_{i=k+1}^{t} \binom{t}{i} q^{i-1} (1-q)^{t-(i-1)}
\]

\[
\text{If } t \geq 2k, \quad \binom{t}{k-1} \leq \binom{t}{k-1-i} \quad \forall i: 1 \leq i \leq k
\]

\[
\Rightarrow P(A) \leq k \cdot \binom{t}{k} (1-q)^t \quad \text{[Since } q \leq \frac{1}{2}]\]

Using \( \binom{t}{k} \leq \left( \frac{et}{k} \right)^k \) and setting \( t = ck^2 \) \((c > 1)\)

we get:

\[
P(A) \leq k \cdot \left( \frac{ec}{k} \right)^k (1-q)^{ck^2}
\]

Since \((1-q) < 1\) = \( C \) st \( \forall C > c^* : c(1-q)^c < e \)

[In fact \( c^* \) can be calculated easily as]

\[
\ln C + c \ln (1-q) < 1 \Rightarrow c \ln \frac{1}{1-q}
\]
2. Pivoting an array: We are given an array of integers. The array size is \( n \). We are given an index \( i \) \( (0 \leq i \leq n-1) \). We need to rearrange the array such that all elements less than \( A[i] \) are placed to the left of \( A[i] \) and all elements greater than it are placed to the right. (Note \( A[i] \) itself may have to move).

**Alg:**

Set \( p \leftarrow A[i] \)

Set \( l \leftarrow 0, r \leftarrow n-1 \)

\*While \( A[l] \leq p \) \& \& \( l++ \)

\*While \( A[r] > p \) \& \& \( r-- \)

Swap \( A[l] \) and \( A[r] \)

Repeat from \* until \( l = r \)

Clearly this takes \( O(n) \) time.

3. Recursive pivoting with priorities: Assign distinct priorities to each element in \( A \). Now choose \( i \) such that \( p_i \) is min. Pivot the array around \( A[i] \). If the new location of \( A[i] \) is \( j \) then recursively do the same thing for array \( A[0...j-1] \cup A[j+1...n-1] \).

Base case: Array of size 1 is already pivoted.

This procedure sorts the array.
4. Time taken for recursive pivoting: Note the treap structure in the algorithm.

Left subtree has keys ≤ k and its root has the min priority, i.e., a treap is formed.

Time taken to pivot an array of size k around min priority element: \( T(\text{find min prioriy}) + T(\text{pivot}) = O(k) \).

Note: Each level of the treap incurs \( O(n) \) pivoting time since the sum of lengths of arrays being pivoted is \( \leq n \).

\[ \therefore \text{Time is } O(n \log n) \text{ where } h \text{ is the height of the treap. (++)} \]

5. Randomized treaps: Give every location a random priority selected uniformly from \([0, 1]\).

Consider any leaf \( v \) of the treap. Consider the path from root to leaf. We call a node in this path good if the min priority node \( \text{deletes} \) lies in the middle third of its array (when seen in sorted order).

\[ \frac{1}{3} \]

Probability a node is good = \( \frac{1}{3} \)

Every time we get a good node, the largest subarray created is \( \frac{2}{3} \) the size of the previous array.

\[ \therefore \text{At most } \log_3 n \text{ good nodes are possible} \]
Noting that each node is labelled independently we get using (++) that

\[ P[\text{length of path} \geq c \cdot 3 \log_3 n] \leq 3 \cdot \log_3 n \cdot \frac{1}{m^3} \leq \frac{1}{m^2} \]

\[ \text{Let } A_j : \text{length of path to leaf } j \text{ is } \geq c \cdot 3 \log_3 n \]

\[ \text{Event } \text{ht of treap} > c \cdot 3 \log_3 n \]

\[ = \bigcup_{j=1}^{n} A_j \]

\[ = P\left[ \text{ht} > 3c \cdot \log_3 n \right] = P\left[ \bigcup_{j=1}^{n} A_j \right] \leq \sum_{j=1}^{n} P(A_j) \leq n \cdot \frac{1}{m^2} = \frac{1}{n} \]

\[ \therefore \text{Treap ht} \leq 3c \cdot \log_3 n \text{ w. prob } \geq 1 - \frac{1}{n} \]

Randomized quicksort takes time

\[ 3c \cdot n \log_3 n \text{ w. prob } \geq 1 - \frac{1}{n} \]

(from (+++))