**Collaborators :** None

**Sources** :

1. <https://www.perplexity.ai/>

**Q4 Dealing with error in randomized computation**

**4-a)** If you have a zero-error algorithm for with failure probability (quite high!), show how to convert it to a zero-error algorithm for with failure probability at most The “slowdown” should only be a factor of a few thousand.

Algorithm for :

1. Run repeatedly for times.
2. If any run produces a non-"?" result, return that result.
3. If all runs produce "?", return "?".

Analysis:

* Probability of all runs failing:
* We want:
* Taking logarithms:

Therefore, we need runs to achieve the desired failure probability. The slowdown factor is 3290, which is indeed "a factor of a few thousand" as required.

**4-b)** Alternatively, show how to convert to an algorithm for which: (i) always outputs the correct answer, meaning ; (ii) has expected running time only a few powers of worse than that of . (Hint: look up the mean of a *geometric random variable*.)

Algorithm for :

1. Repeatedly run until a non-"?" result is obtained.
2. Return this result.

Analysis:

* The number of runs follows a geometric distribution with (success probability). This means that the probability that Algorithm returns the result after runs is
* Expected number of runs (Average number of runs Algorithm takes to return the result)

The expected running time of is 10 times that of , which is only a few powers of worse, as required.

**4-c)** If you have a no-false-negatives algorithm for with failure probability (quite high!), show how to convert it to a no-false-negatives algorithm for with failure probability at most The “slowdown” should only be a factor of a few thousand.

Algorithm for :

1. Run for times.
2. If any run outputs , return .
3. Otherwise, return .

Analysis:

* This preserves the no-false-negatives property.
* For fails only if all runs output .
* Probability of failure:

Solving for as in part (a), we get . The slowdown factor is 3290, which is "a factor of a few thousand" as required.

**4-d)** If you have a two-sided error algorithm for with failure probability , show how to convert it to a two-sided error algorithm for with failure probability at most The “slowdown” should only be a factor of a few dozen thousand. (Hint: look up the Chernoff bound.)

Algorithm for :

1. Run for times ( odd).
2. Return the majority vote of the outputs.

Analysis using Chernoff bound:

Let be the number of correct outputs in runs.

(as the success probability is )

We want: (to get the majority votes for successful outcome)

Using the Chernoff bound with for

We need: Solving this inequality:

Therefore, we can use (to make it odd) runs of . The slowdown factor is 41589, which is "a factor of a few dozen thousand" as required.