1 Random vectors

a)

We define two vectors as $a = (a_1, ..., a_d)$, $b = (b_1, ..., b_d)$ and we note a random variable as $X_i = a_i \cdot b_i$, with $i \in [d]$. Also, we define the positive numbers of values X_i as a random variable Y. Since we choose independently and randomly values for a_i and b_i respectively between $\frac{1}{\sqrt{d}}$ and $-\frac{1}{\sqrt{d}}$, the probability that $X_i = a_i \cdot b_i$ is positive is $\frac{1}{2}$. So, the positive numbers of values X_i, Y is easy to understand that follows a Binomial distribution $Y \sim Bin(d, \frac{1}{2})$. Moreover, given that the total number of $a_i \cdot b_i$ is d then the negative numbers are $d-Y$. Therefore, we can reshape the question about the event of dot product not being $\frac{1}{10}$-close to orthogonal as the event in which the absolute value of the difference between positive and negative $a_i \cdot b_i$ out of all d possible products is at least $\frac{1}{10}$. In particular, $\frac{|Y-(d-Y)|}{d} \geq \frac{1}{10}$. The calculation of the event is the following:

$$Pr(\frac{|Y-(d-Y)|}{d} \geq \frac{1}{10}) = Pr(\frac{|2Y-d|}{d} \geq \frac{1}{10}) = Pr(|Y - \frac{d}{2}| \geq \frac{d}{20}) = Pr(\{Y \geq \frac{d}{2} + \frac{d}{20}\} \cup \{Y \leq \frac{d}{2} - \frac{d}{20}\})$$

Using the union bound rule:

$$Pr(\frac{|Y-(d-Y)|}{d} \geq \frac{1}{10}) \leq Pr(Y \geq \frac{d}{2} + \frac{d}{20}) + Pr(Y \leq \frac{d}{2} - \frac{d}{20}) \tag{1}$$

Using the Chernoff bound for each part of equation (1) we have:

$$Pr(Y \geq \frac{d}{2} + \frac{d}{20}) = Pr(Y \geq (1 + \frac{1}{10}) \cdot \frac{d}{2}) \leq (\frac{e^{\frac{1}{10}}}{(1+\frac{1}{10})^{1+\frac{1}{10}}})^{\frac{d}{2}} \approx 0.9976^d \tag{2}$$

$$Pr(Y \leq \frac{d}{2} - \frac{d}{20}) = Pr(Y \leq (1 - \frac{1}{10}) \cdot \frac{d}{2}) \leq (\frac{e^{-\frac{1}{10}}}{(1-\frac{1}{10})^{1-\frac{1}{10}}})^{\frac{d}{2}} \approx 0.99742^d \tag{3}$$

Based on (2) and (3), equation (1) turns into:

$$Pr(\frac{|Y-(d-Y)|}{d} \geq \frac{1}{10}) \leq Pr(Y \geq \frac{d}{2} + \frac{d}{20}) + Pr(Y \leq \frac{d}{2} - \frac{d}{20}) \approx 0.9976^d + 0.99742^d$$

b)

We pick randomly k d-dimentional vectors. We define the event $E_{i,j}$ to note the event that the i-th vector is not $\frac{1}{10}$ close to the j-th vector. We note the bound of the probability p that among all k vectors, exist at least two vectors that are not $\frac{1}{10}$ close, as the union bound of:

$$Pr(\bigcup_{i\in[k-1], j\in[k], i<j} E_{i,j}) \leq Pr(E_{1,2}) + ... + Pr(E_{m,n}) = \frac{k \cdot (k-1)}{2} \cdot Pr(E_{1,2})$$

Based on result of subquestion (a):

$$Pr(\bigcup_{i\in[k-1], j\in[k], i<j} E_{i,j}) \leq Pr(E_{1,2}) + ... + Pr(E_{m,n}) = \frac{k \cdot (k-1)}{2} \cdot Pr(E_{1,2}) \Longrightarrow$$

$$= \frac{k \cdot (k-1)}{2} \cdot 0.9976^d + 0.99742^d \leq \frac{k^2}{2} \cdot 0.9976^d + 0.99742^d$$

Making sure that the choice of k vectors to be $\frac{1}{10}$ close with a constant probability of $1-\epsilon$, where $0 < \epsilon < 1$, we equalize the upper bound of the previous inequality to ϵ in order to calculate the larger value of k. Hence:

$$\epsilon = \frac{k_{max}^2}{2} \cdot 0.9976^d + 0.99742^d \Longrightarrow k_{max} = \sqrt{\frac{2 \cdot \epsilon}{0.9976^d + 0.99742^d}}$$

2 Using Chernoff/Hoeffding bounds

a) Amplification of the success probability

Using the hint of exercise we run the algorithm $t = \Theta(log(\frac{1}{\delta}))$ times and we define the random variable X_i the output each time. Also, we note the indicator random variable I_{X_i} if X_i is in the range that we think acceptable (we note the range as [a,b]). So:

$$I_{X_i} = \begin{cases} 1 & X_i \text{ not inside the range}[a,b] \\ 0 & X_i \text{ inside the range}[a,b] \end{cases}$$

Therefore, if we want to solve the algorithm will probability $\frac{2}{3}$ then $Pr(I_{X_i} = 1) \leq \frac{1}{3}$. Also, we define the random variable Y as $Y = \sum_{i=1}^{m} I_{X_i}$, which describes the total number of tries outside the range $[a,b]$. The expected value of Y is given as:

$$E[Y] = E[\sum_{i=1}^{m} I_{X_i}] = \sum_{i=1}^{m} E[I_{X_i}] \leq \frac{m}{3}$$

Since we output the median number of the total m tries, we need to know when the algorithm produces a wrong answer. The algorithm outputs a wrong answer if half of m tries gives a result either smaller than a or bigger than b. So the probability that upper bounds the wrong answer is $Pr(Y \leq \frac{m}{2})$. Applying the Chernoff inequality to the complimentary probability $Pr(Y \geq \frac{m}{2}) = 1 - Pr(Y \leq \frac{m}{2})$:

$$Pr(Y \geq \frac{m}{2}) = Pr(Y \geq (1+\frac{1}{2}) \cdot \frac{m}{3}) \leq (e^{-\frac{1}{12}})^{\frac{m}{3}} = \Theta(\delta)$$

Hence the algorithm gives a good approximation within the range $[a,b]$ with probability $\geq 1-\delta$ in time $O(T(n) \cdot log\frac{1}{\delta})$.

b) Noisy Binary Search

b).1

The algorithm is similar to the original binary search but with the twist of asking k queries about the position of the input x value instead of only one. After k votes, we have an intuition based on our counter c where the input x is located. If c is positive is going to be at upper half,otherwise it is going to be at lower half. By doing this, we increase the probability of finding the correct answer per step. The probability can be specified based on the number of k queries we ask per step. As we are going to present at subquestion (b) the lower bound of k to achieve a 90% success probability is $\frac{ln(10 \cdot log(n))}{2 \cdot \epsilon^2}$. At each step, we get the correct answer from the query with probability $\frac{1}{2} + \epsilon$ and a wrong one with probability $\frac{1}{2} - \epsilon$. The algorithm is presented below as "Algorithm 1: Noisy-Binary-Search".

Algorithm 1 Noise Binary Search

```
Require: int X, int[ ] A, int k, int n
  low = 0
  high = n
  while low ≤ high do
      c = 0
      mid = (low+high)/2
      for i = 0 to k do
          if A[x] == mid then return mid
          if A[x] > mid then c = c + 1
          if A[x] < mid then c = c − 1
      end for
      if c ≥ 0 then
          low = mid + 1
      else
          high = mid − 1
      end if
  end while
  return − 1
```

b).2

We define as X_i the indicator random variable that describes the event of query i returns a correct answer. Also, we define the random variable $X = \sum_{i=1}^{k} X_i$ as the event of the summation of correct answer from all k queries we asked. The expected value of X_i regardless of the direction is $E[X_i] = \mu = Pr(X_i) = \frac{1}{2} + \epsilon$, as it is given from the last two bullets. If we want to show that the algorithm is at least 90% accurate that means that algorithm points out the new direction correctly in all k iterations with probability at least 0.9 or in other words that algorithm points out the new direction wrongly in all k iterations with probability at most 0.1. In general, we give a wrong answer if we answer wrongly at the majority of the queries. In particular, if we increase or decrease our counter wrongly for over $\frac{k}{2}$ queries, we end up arrange wrongly the limits of the new iteration. So applying Hoeffding bound we have:

$$Pr(X \leq \frac{k}{2}) = Pr(X \leq \frac{k}{2} + k \cdot \epsilon - k \cdot \epsilon) = Pr(X \leq k \cdot (\frac{1}{2} + \epsilon) - k \cdot \epsilon) = Pr(X \leq k \cdot \mu - k \cdot \epsilon) \leq e^{-2 \cdot k\epsilon^2}$$

Since we want the error probability for the whole algorithm and not for one iteration of it, we should take into consideration that the query "voting" will be executed at most $log(n)$ times as in each iteration the mid variable is divided by 2. So, if we want the error probability for the whole algorithm to be at most 0.1

we imply that:

$$\begin{aligned} log(n) \cdot e^{-2 \cdot k\epsilon^2} &\leq \frac{1}{10} \Longrightarrow \\ e^{-2 \cdot k\epsilon^2} &\leq \frac{1}{10 \cdot log(n)} \Longrightarrow \\ -2 \cdot k \cdot \epsilon^2 &\leq ln(\frac{1}{10 \cdot log(n)}) \Longrightarrow \\ 2 \cdot k \cdot \epsilon^2 &\geq ln(10 \cdot log(n)) \Longrightarrow \\ k &\geq \frac{ln(10 \cdot log(n))}{2 \cdot \epsilon^2} \end{aligned}$$

b).3

As we see at sub question (a) the algorithm divides the midterm variable at each iteration of the while loop by 2, so at most the while loop will run for $log(n)$ times. As a result, we will use $log(n)$ times the k queries we ask at each iteration. Given that we have a lower bound of k from sub question (b) we have:

$$\begin{aligned} \text{Total query iterations} = log(n) \cdot k &= log(n) \cdot \frac{ln(10 \cdot log(n))}{2 \cdot \epsilon^2} \xrightarrow{ln(x)=2.303 \cdot log(x)} \\ &= log(n) \cdot \frac{2.303 \cdot log(10 \cdot log(n))}{2 \cdot \epsilon^2} \Longrightarrow \\ &= O(log(n) \cdot \frac{log(10 \cdot log(n))}{\epsilon^2}) \end{aligned}$$

In conclusion, the total query iterations are polylogarithmic in terms of n and polynomial in terms of $\frac{1}{\epsilon}$.

3 Randomized Routing on the Hypercube

a)

We define two source nodes $x \in R^n$ and $y \in R^n$ having paths p_x and p_y respectively. Assume that source nodes meet at i^{th} step, which means that the bit fixing algorithm has processed i bits or in other words their destination addresses are the same for the first i steps and their source addresses are the same for the last $n - i$ bits. Since algorithm fixes one bit per round, at its next iteration, $i + 1$ step, the nodes will be separated, which means that the $i+1$ bit will be 1 for one path and 0 for the other. After that the algorithm will continue with $i + 2$ bit and never work on the $i + 1$ bit till the end of it, so $i + 1$ bit is fixed. If we assume that they are going to meet again, that means there is a future step at bit k such that $k > i$. At this step, all the first k bits should be the same but this is not possible since i^{th} bit is different.

b)

The answer is no. In particular, there could be more than one in succession repeated edges between two routes r_i and r_j. Consider the case where i is served first but its route is blocked in the next edge due to the delay of other routes, so packet i is waiting.Then j packet could be served next and come to the same node waiting in list with i packet. Hence, it is possible both packets i and j might wait in the same queue in more than one node at same time step.

c)

c).1

We note i' the last packet that blocks packet i.The delay of the packet i increases from l to $l+1$ only if another packet p appears after packet i' and this packet does not have any delay and it only uses this repeated edge. Based on subquestion (a) it is true that packet p will never be an obstacle again since once separated they do not rejoin again ,therefore the delay of packet i' will be increased by 1. Based on the fact that the last packet i' that blocks i, has an initial lag equal to $l-1$ (Otherwise, the delay of the packet i would not be equal to l), i' packet,which belongs to S, leaves p_i with lag $(l-1)+1=l$. Moreover, we should add that if $l=0$ then the i' packet that leaves p_i with lag $l+1=1$ is the initial packet i.

c).2

Taking into consideration all the possible cases, at the best case packet i has no delays. This happens if $S=\emptyset$. If there is a packet i' in the set S then the delay could increase by at most 1, if they share a common edge and packet i waits packet i'. If there is one more packet p in set $S=i',p$, there are two options. In the best case, it does not delay packet i or in the worst case the packet p is ahead of packet i' in the common edge and delays both i' and i packet so at most the delay is $|S|=2$. Generalize this idea, given that there are at most $|S|$ packets whose routs pass through the common edge of i packet, the delay could be at most $|S|$, if all packets included in S are stacked in the queue of this common edge.

d)

We will argue based on a specified example. For this example, we are summing over all edges in the hypercube for the calculation of $\sum_e Y_e$ and we note $Y_e=0$ to edges in which no packets pass through. Also, we are summing over all packets in the hypercube for the calculation of $\sum_i L_i$ and we note $L_i=0$ to packets in which destination and source are identical.

In our example, we define a hypercube with number of nodes equal to 4, a square. Therefore, we have as nodes $n_0=00, n_1=01, n_2=10, n_3=11$ and as edges $e_{01}, e_{10}, e_{02}, e_{20}, e_{13}, e_{31}, e_{23}, e_{32}$. Also, we define the packets as following: $\{p_0: n_0 \to n_0, p_1: n_3 \to n_3, p_2: n_1 \to n_2, p_3: n_2 \to n_1\}$. Based on the assumption that if source and destination is identical then $L_0=L_1=0$ and $L_2=L_3=2$, since we need two moves to go from $n_1=01$ to $n_2=10$ respectively. Hence, $\sum_{i=0}^{3} L_i=4$. Looking at edge perspective, packet p_2 will use edges e_{13},e_{32} and packet p_3 will use edges e_{20},e_{01}. All these edges will have $Y_e=1$, since they are used only one time, while edges $e_{10}, e_{02}, e_{31}, e_{32}$ will not be used so $Y_e=0$. In conclusion,we have that $\sum_{i=0}^{3} L_i=\sum_{i=0}^{8} Y_{ei}=4$

e)

We want to calculate the expected value of the number of edges in the route of all packets, in particular $\sum_{i=i}^{N} E[L_i]$. For the calculation of $E[L_i]$, we define as N the number of nodes in hypercube and the total number of bits to describe its node as $logN$. Also, we note the indicator variable I_j of the event that the j-th bit of the source and the destination are different. Based on the fact that source and destination are chosen independently and randomly, we note that $E[I_j]=Pr(I_j=1)=\frac{1}{2}$. Based on the fact that there are

$logN$ bits, the length of the path of a packet i is equal to the number of different bits. So we have:

$$E[L_i] = \sum_{j=i}^{logN} E[I_j] = \frac{logN}{2}$$

In conclusion, generalizing for N packets we have: $\sum_{i=i}^{N} E[L_i] = \frac{N \cdot logN}{2}$

f)

Based on the fact that the total number of edges is $N \cdot logN$, each has the same distribution with respect to the number of packets that pass it and from subquestion (d) we know that $\sum_i L_i = \sum_e Y_e$, using the expected value of subquestion (e) we know that:

$$\begin{aligned}
\sum_e E[Y_e] &= \sum_i E[L_i] \Longrightarrow \\
\sum_e E[Y_e] &= \frac{N \cdot logN}{2} \Longrightarrow \\
N \cdot logN \cdot E[Y_e] &= \frac{N \cdot logN}{2} \Longrightarrow \\
E[Y_e] &= \frac{1}{2}
\end{aligned}$$