Foundations of Data Mining - Problem Set 3

Aviv Rosenberg - [redacted]

January 7, 2018

1 Question 1

Let A be a matrix with n rows and d columns, assume $n \geq d$. Let $U\Sigma V^T$ be the Singular Value Decomposition of A as defined in class, and let $U_k\Sigma_k V_k^T$ be its truncation to rank k as defined in class.

Denote by $u_1, \ldots, u_r$ the columns of U and by $v_1, \ldots, v_r$ the columns of V. As we have seen in class:

$$A = \sum_{i=1}^{r} \sigma_i(A) u_i v_i^T$$

$$A_k = \sum_{i=1}^{k} \sigma_i(A) u_i v_i^T$$

Proposition 1. $||A - A_k||_2 = \sigma_{k+1}(A)$

Proof. First notice that:

$$A - A_k = \sum_{i=1}^{r} \sigma_i(A) u_i v_i^T - \sum_{i=1}^{k} \sigma_i(A) u_i v_i^T = \sum_{i=k+1}^{r} \sigma_i(A) u_i v_i^T$$

Therefore the singular values of $A - A_k$ are $\sigma_{k+1}(A), \ldots, \sigma_r(A)$ (because the SVD without order is unique). Now by the definition of the $||\cdot||_2$ matrix norm and of singular values we have:

$$||A - A_k||_2 = \max_{||x||_2=1} ||(A - A_k)x||_2 = \sigma_1(A - A_k) = \sigma_{k+1}(A)$$

□

Proposition 2. $\min_{B:rank(B)\leq k} ||A - B||_2 = \sigma_{k+1}(A)$

Proof. Let B be a matrix with $rank(B) = k$. Therefore $\dim(\ker(B)) = d - k$. Because of the dimensions we know that:

$$\ker(B) \cap span\{v_1, \ldots, v_{k+1}\} \neq \{0\}$$

So we can take a unit vector $w \neq 0$ from the intersection, and write it in the orthonormal basis $v_1, \ldots, v_{k+1}$:

$$w = \sum_{i=1}^{k+1} \alpha_i v_i$$

And if we multiply by A we get:

$$Aw = \sum_{i=1}^{k+1} \alpha_i A v_i = \sum_{i=1}^{k+1} \alpha_i \sigma_i(A) u_i$$

Finally we get:

$$||A - B||_2 = \max_{||x||_2=1} ||(A-B)x||_2 \geq ||Aw - Bw||_2 \underbrace{=}_{w \in \ker(B)}$$

$$||Aw||_2 = ||\sum_{i=1}^{k+1} \alpha_i \sigma_i(A) u_i||_2 = \sqrt{\sum_{i=1}^{k+1} \alpha_i^2 \sigma_i^2(A)} \geq$$

$$\sigma_{k+1}(A) \sqrt{\sum_{i=1}^{k+1} \alpha_i^2} = \sigma_{k+1}(A) ||w||_2 = \sigma_{k+1}(A)$$

□

Corollary 3. $\min_{B:rank(B) \leq k} ||A - B||_2 = ||A - A_k||_2 = \sigma_{k+1}(A)$

2 Question 2

Let A be a matrix with n rows and d columns, assume $n \geq d$. Let $U\Sigma V^T$ be the Singular Value Decomposition of A as defined in class.

2.1 Section a

We complete $u_1, \ldots, u_r$ to an orthonormal basis $u_{r+1}, \ldots, u_n$ and instead of our matrix U consider the $n \times n$ orthogonal matrix $U = (u_1, \ldots, u_n)$. We do the same for $v_1, \ldots, v_r$ and consider the $d \times d$ orthogonal matrix $V = (v_1, \ldots, v_d)$. Now We change Σ to be a $n \times d$ matrix with 0 every where but $\Sigma_{ii} = \sigma_i$ for $i \in \{1, \ldots, r\}$. Notice that we still have $A = U\Sigma V^T$ because of the zeros in Σ.

We use the fact that multiplying a vector by an orthogonal matrix does not change its norm and the fact that $U^T U = I$, to conclude:

$$||Ax - b||_2 = ||U^T(Ax - b)||_2 = ||U^T(U\Sigma V^T x - b)||_2 = ||\Sigma V^T x - U^T b||_2$$

Now we denote $z = V^T x$ and continue:

$$||Ax - b||_2 = ||\Sigma z - U^T b||_2 = || \begin{pmatrix} \sigma_1 z_1 \\ \vdots \\ \sigma_r z_r \\ 0 \\ \vdots \\ 0 \end{pmatrix} - \begin{pmatrix} u_1^T b \\ \vdots \\ u_r^T b \\ u_{r+1}^T b \\ \vdots \\ u_n^T b \end{pmatrix} ||_2 = || \begin{pmatrix} \sigma_1 z_1 - u_1^T b \\ \vdots \\ \sigma_r z_r - u_r^T b \\ -u_{r+1}^T b \\ \vdots \\ -u_n^T b \end{pmatrix} ||_2 =$$

$$\sqrt{\sum_{i=1}^{r}(\sigma_i z_i - u_i^T b)^2 + \sum_{i=r+1}^{n}(u_i^T b)^2}$$

Now (because of the squaring) it is clear that a minimizing z^* will have:

$$z_i^* = \begin{cases} \frac{u_i^T b}{\sigma_i} & 1 \leq i \leq r \\ 0 & r+1 \leq i \leq n \end{cases}$$

So (since $VV^T = I$) a minimizing x^* will be $x^* = Vz^*$.

2.2 Section b

We can use the previous section to get:

$$\min_{x \in \mathbb{R}} ||Ax - b||_2 = \sqrt{\sum_{i=1}^{r}(\sigma_i z_i^* - u_i^T b)^2 + \sum_{i=r+1}^{n}(u_i^T b)^2} =$$

$$\sqrt{\sum_{i=1}^{r}(\sigma_i \frac{u_i^T b}{\sigma_i} - u_i^T b)^2 + \sum_{i=r+1}^{n}(u_i^T b)^2} = \sqrt{\sum_{i=1}^{r} 0 + \sum_{i=r+1}^{n}(u_i^T b)^2} = \sqrt{\sum_{i=r+1}^{n}(u_i^T b)^2}$$

3 Question 3

We have a long file of names (each name may appear multiple times) and we want to privately publish the name with maximum frequency. We do it by adding to the frequency n_i of each name i, a random variable N_i drawn from a $Lap(1/\epsilon)$ distribution and publishing the name j with the largest noisy frequency $n_j + N_j$.

Remark 4. I don't know if I can assume that all the names come from a finite set, so I have written two proofs. The first is with this assumption and it is simpler. The second is without the assumption.

3.1 Proof with Assumption

Assume that all the names come from the set $D = \{t_1, \ldots, t_d\}$ of size d. We define the function $f : D^n \to \mathbb{R}^d$ as follows:

$$f(X = (x_1, \ldots, x_n)) = \begin{pmatrix} \#\{i \in \{1, \ldots n\} : x_i = t_1\} \\ \vdots \\ \#\{i \in \{1, \ldots n\} : x_i = t_d\} \end{pmatrix}$$

From the claim that follows $GS_f = 2$, so as we have seen in class the function $A : D^n \to \mathbb{R}^d$ defined by $A(X) = f(X) + Lap^d(\frac{1}{\epsilon})$ is $(2\epsilon, 0)-$differentially private (We saw that adding $Lap^d(\frac{2}{\epsilon})$ gives $(\epsilon, 0)-$differential privacy, but the proof is the same). Now we define the function $B : D^n \to D$ as follows:

$$B(X) = t_{\arg\max_{1 \le i \le d} A(X)_i}$$

As we have seen in class post-processing preserves differential privacy and therefore B is $(2\epsilon, 0)-$differentially private. Notice that B is exactly our mechanism for publishing the name with maximum frequency and therefore it is $(2\epsilon, 0)-$differentially private.

Proposition 5. *The global sensitivity of the function f defined above is 2.*

Proof. Let $X = (x_1, \ldots, x_n)$ and $X' = (x'_1, \ldots, x'_n)$ be neighboring datasets, and assume they differ in the ith entry where $x_i = t_j$ and $x'_i = t_k$. Thus $f(X)$ and $f(X')$ differ only in the jth and the kth entry where $f(X)_j = f(X')_j + 1$ and $f(X)_k = f(X')_k - 1$. Therefore $||f(X) - f(X')||_1 = 2$. So $GS_f = 2$ since no neighboring data can have bigger distance, and these two datasets reach distance 2. □

3.2 Proof without Assumption

Assume that all the names come from the set D. We define a function $f : D^n \to \mathbb{R}^n$ as follows. For $X = (x_1, \ldots, x_n) \in D^n$ assume that the distinct names are $d_1, \ldots, d_\ell$ and that they are ordered by their frequency (i.e. d_1 has the biggest frequency and d_ℓ the smallest). Denote by $n_i(X)$ the frequency of d_i in X, so f returns $(n_1(X), \ldots, n_\ell(X), 0, \ldots, 0) \in \mathbb{R}^n$.

From the claim that follows $GS_f = 2$, so as we have seen in class the function $A : D^n \to \mathbb{R}^n$ defined by $A(X) = f(X) + Lap^n(\frac{1}{\epsilon})$ is $(2\epsilon, 0)-$differentially private (We saw that adding $Lap^n(\frac{2}{\epsilon})$ gives $(\epsilon, 0)-$differential privacy, but the proof is the same). Now we define the function $B : D^n \to D$ to return the name d_i that has the biggest entry in $A(X)$. As we have seen in class post-processing preserves differential privacy and therefore B is $(2\epsilon, 0)-$differentially private. Notice that B is exactly our mechanism for publishing the name with maximum frequency and therefore it is $(2\epsilon, 0)-$differentially private.

Proposition 6. *The global sensitivity of the function f defined above is 2.*

Proof. Let $X = (x_1, \ldots, x_n)$ and $X' = (x'_1, \ldots, x'_n)$ be neighboring datasets, and assume they differ in the ith entry where $x_i = t_j$ and $x'_i = t_k$. The proof of this claim from the previous section actually works here as well. The only problem that can arise here is that because of the change, the order of the names in $f(X')$ is different then the order of the names in $f(X)$. But a change in order can only happen if the changed names had frequencies equal to some other names. In that case we could have ordered $f(X)$ such that a change in order would not have happened. So we can assume without loss of generality that a change in order did not occur, and therefore the proof of this claim from the previous section will work here as well. □

4 Question 4

Given a database $S = s_1, \ldots, s_n$, all values integer and distinct between 1 and T, n odd. we seek to give an $(\epsilon, 0)-$differential privacy preserving approximation to the median element in the database (The true median is $s_{\lfloor \frac{n}{2} \rfloor}$).

The utility of an "approximate median" r for S is defined as follows: Let Q be the set of all $R = r_1, \ldots, r_n$, all values integer and distinct between 1 and T, such that the median of R is equal to r ($r_{\lfloor \frac{n}{2} \rfloor} = r$). For all $R \in Q$, let $\Delta(R)$ be the size of the symmetric difference between S and R. Let $\Delta = \min_{R \in Q} \Delta(R)$. The utility of an approximate median r, $u_S(r) = -\Delta$.

4.1 Section a

The sensitivity of this utility is 2. Let X and X' be neighboring datasets (thus $|X \Delta X'| \leq 2$), we want to show that $|u_X(r) - u_{X'}(r)| \leq 2$ for every $r \in \{1, \ldots, T\}$. Denote $s = u_X(r), s' = u_{X'}(r)$ and let B be the dataset such that $|X \Delta B| = -s$ and the median of B is r. By the claim that follows:

$$|X' \Delta B| \leq |X' \Delta X| + |X \Delta B| \leq 2 - s$$

Therefore, by the definition of the utility, $s' = u_{X'}(r) \geq s - 2$. We can perform the same steps symmetrically and get $s \geq s' - 2$. Now we can conclude the proof as follows:

$$|u_X(r) - u_{X'}(r)| = |s - s'| \leq 2$$

Proposition 7. *Let A, B, C be sets, then $B \Delta C \subseteq (A \Delta B) \cup (A \Delta C)$. Therefore $|B \Delta C| \leq |A \Delta B| + |A \Delta C|$.*

Proof. If $B \Delta C = \emptyset$ the claim is trivial. Otherwise, let $x \in B \Delta C$ then $x \in B \setminus C$ or $x \in C \setminus B$. Assume without loss of generality that $x \in B \setminus C$ (the other case is symmetric). If $x \in A$ then $x \in A \setminus C$, so $x \in A \Delta C$. If $x \notin A$ then $x \in B \setminus A$, so $x \in A \Delta B$. Any way $x \in (A \Delta B) \cup (A \Delta C)$. □

Remark 8. It is not clear what is the definition of neighboring datasets. If for neighboring datasets X and X' we have $|X \Delta X'| \leq 1$, then the sensitivity is 1.

4.2 Section b

The exponential mechanism outputs an integer $r \in \{1, \ldots, T\}$ with probability proportional to $e^{\frac{\epsilon}{2} u_S(r)}$. Consider the following example:

We take $n = 7$, $S = \{1, 2, 3, 4, T-2, T-1, T\}$, and now for every $i \in \{5, \ldots, T-3\}$ we define:

$$R_i = \{1, 2, 3, i, T-2, T-1, T\}$$

Notice that $median(S) = 4$ and $median(R_i) = i$. Also notice that $u_S(i) = -2$ because $|S \Delta R_i| = 2$, so every $i \in \{5, \ldots, T-3\}$ has the same probability $\propto e^{\frac{\epsilon}{2}(-2)} = e^{-\epsilon}$ to be the output. 4 has probability $\propto 1$ to be the output.

Therefore the expected difference between the median and the approximate median produced by the exponential mechanism is at least:

$$|4-\frac{4}{1+(T-2)e^{-\epsilon}}-\sum_{i=5}^{T-3}\frac{e^{-\epsilon}i}{1+(T-2)e^{-\epsilon}}| = |4-\frac{1}{1+(T-2)e^{-\epsilon}}(4+e^{-\epsilon}\frac{5+T-3}{2}(T-2))|$$

This can be very large, and tends to ∞ as $T \to \infty$.

4.3 Section c

Consider the mechanism A that returns the ith largest item from S where $i = \left\lfloor \frac{n}{2} + Lap(\frac{1}{\epsilon}) \right\rfloor$ (Round up to 1 if below 1, round down to n if above n). The following example will show that there is no $\epsilon > 0$ for which this mechanism is $(\epsilon, 0)-$differentially private.

Take the neighboring datasets $X = \{1,2,3,4,5\}$ and $X' = \{1,2,3,4,6\}$ that both have median 3. Take the approximate median $r = 5$. On the one hand $r \in X$ and therefore $\Pr[A(X) = 5] > 0$, but on the other hand $r \notin X'$ and therefore $\Pr[A(X') = 5] = 0$.

Assume towards contradiction that A is $(\epsilon, 0)-$differentially private for some $\epsilon > 0$, then by the differential privacy definition:

$$0 < \Pr[A(X) = 5] \leq e^{\epsilon} \Pr[A(X') = 5] = 0$$

In contradiction.

4.4 Section d

Algorithm. The server side protocol:

1. Initialize $middle = \frac{T}{2}$ and $diff = \frac{T}{4}$
2. while $diff > \frac{1}{2}$ do
   (a) broadcast the predicate $P_{middle} =$ "$are\ you\ <\ middle$" and get $answers$ from clients
   (b) if $Count(answers, P_{middle}) < \frac{n}{2}$ then
      i. $middle = middle + diff$
   (c) else
      i. $middle = middle - diff$
   (d) $diff = \frac{diff}{2}$
3. if $middle \in \mathbb{Z}$ then
   (a) output $middle$
4. else

(a) if $Count(answers, P_{\lceil middle \rceil}) < \frac{n}{2}$ then

i. output $\lceil middle \rceil$

(b) else

i. output $\lfloor middle \rfloor$

The function $Count(answers, P_{middle})$ (which intends to approximate the number of clients with inputs $< middle$) returns:

$$\frac{1}{2\alpha}\left(n(\alpha - \frac{1}{2}) + \sum_{i=1}^{n} answers[i]\right)$$

Algorithm. The client side protocol (with input x and predicate P_{middle}):

1. if $x < middle$ then
   (a) return 1 with probability $\frac{e^\alpha}{1+e^\alpha}$ and 0 with probability $\frac{1}{1+e^\alpha}$
2. else
   (a) return 0 with probability $\frac{e^\alpha}{1+e^\alpha}$ and 1 with probability $\frac{1}{1+e^\alpha}$

Proposition 9. *For $\alpha = \frac{\epsilon}{2\log T}$ the mechanism is $\epsilon-$local differentially private.*

Proof. The protocol has less then $2\log T$ rounds (since every round we divide $diff$ (which is initially $\frac{T}{4}$) by 2 until it is $< \frac{1}{2}$). So we use each input at most $2\log T$ times, and every time through a $\frac{\epsilon}{2\log T}-$randomizer (we saw in class that this algorithm is indeed a randomizer). In conclusion we have $\epsilon-$local differential privacy. □

Proposition 10. $\mathbb{E}\left[Count(answers, P_{middle})\right]$ *is* the number of clients with inputs that are $< middle$.

Proof. Denote by $c = \#\{i \in \{1, \ldots, n\} : x_i < middle\}$. Thus:

$$\mathbb{E}\left[Count(answers, P_{middle})\right] = \mathbb{E}\left[\frac{1}{2\alpha}\left(n(\alpha - \frac{1}{2}) + \sum_{i=1}^{n} answers[i]\right)\right] =$$

$$\frac{1}{2\alpha}\left(n(\alpha - \frac{1}{2}) + \mathbb{E}\left[\sum_{i=1}^{n} answers[i]\right]\right) = \frac{1}{2\alpha}\left(n(\alpha - \frac{1}{2}) + \sum_{i=1}^{n}\mathbb{E}\left[answers[i]\right]\right) =$$

$$\frac{1}{2\alpha}\left(n(\alpha - \frac{1}{2}) + \sum_{i:x_i < middle}\mathbb{E}\left[answers[i]\right] + \sum_{i:x_i \geq middle}\mathbb{E}\left[answers[i]\right]\right) =$$

$$\frac{1}{2\alpha}\left(n(\alpha-\frac{1}{2})+\sum_{i:x_i<middle}(\frac{1}{2}+\alpha)+\sum_{i:x_i\geq middle}(\frac{1}{2}-\alpha)\right)=$$

$$\frac{1}{2\alpha}\left(n(\alpha-\frac{1}{2})+c(\frac{1}{2}+\alpha)+(n-c)(\frac{1}{2}-\alpha)\right)=$$

$$\frac{1}{2\alpha}\left(n(\alpha-\frac{1}{2})+\frac{1}{2}c+\alpha c+n(\frac{1}{2}-\alpha)-\frac{1}{2}c+\alpha c\right)=\frac{1}{2\alpha}2\alpha c=c$$

□

Proposition 11. *The protocol gives an approximation of the median.*

Proof. The protocol has $O(\log T)$ rounds, and it performs an approximate binary search for the median. Notice that if $Count(answers, P_{middle})$ gives the correct count every time (or at least close enough so we choose the right direction for the next round), then the protocol gives the exact median. As we saw in the previous claim $Count(answers, P_{middle})$ gives the correct count in expectation, so we have a reason to believe that we get somewhat of an approximation to the median. I think that I can't prove anything more with the tools we learned in class. But I will say this:

It seems that, given that we were right in the previous round, we have an exponentially small probability to be wrong in this round (because of the chernoff bound and the fact that two wrong answers can actually compensate one another). Thus we have $\log T$ exponentially small probabilities to be wrong, so our probability to be wrong is very small. □

5 Question 5

A $2-d$ tree T is a binary tree that represents a set P of points in the plane as follows. Every leaf stores a point of P. Every non-leaf node v at an even distance from the root corresponds to a horizontal line $\ell(v)$ such that $\lfloor \frac{k}{2} \rfloor$ of the k points in v's subtree are above $\ell(v)$ and are stored at the left subtree of v and $\lceil \frac{k}{2} \rceil$ of the of the k points in v's subtree are below $\ell(v)$ and are stored at the right subtree of v. Similarly, every non-leaf node v at an odd distance from the root corresponds to a vertical line $\ell(v)$ such that $\lfloor \frac{k}{2} \rfloor$ of the k points in v's subtree are to the left of $\ell(v)$ and are stored at the left subtree of v and $\lceil \frac{k}{2} \rceil$ of the of the k points in v's subtree are to the right of $\ell(v)$ and are stored at the right subtree of v.

5.1 Section a

Let v be the root of a subtree whose leaves correspond to the points $p_1 = (x_1, y_1), \ldots, p_\ell = (x_\ell, y_\ell)$. Define:

$$x_{min} = \min_{1 \le i \le \ell} x_i$$

$$x_{max} = \max_{1 \le i \le \ell} x_i$$

$$y_{min} = \min_{1 \le i \le \ell} y_i$$

$$y_{max} = \max_{1 \le i \le \ell} y_i$$

and take the rectangle $r(v) = \left\{p = (x, y) \in \mathbb{R}^2 : x_{min} \le x \le x_{max}, y_{min} \le y \le y_{max}\right\}$. From the definition of $r(v)$ it is clear that all the points $p_1, \ldots, p_\ell$ are contained in the rectangle. Now we want to show that all the other points in P are not contained in $r(v)$.

Let $p = (x, y)$ be a point whose leaf is not in the subtree rooted by v. Let u be the lowest common ancestor of v and the leaf of p, and assume without loss of generality that u is at an even distance from the root (otherwise the proof is similar). Now assume without loss of generality that v is in the left subtree of u, and the leaf of p is on the right subtree of u. By definition of the tree p is below $\ell(u)$, and $p_1, \ldots, p_\ell$ are above $\ell(u)$. By the definition of $r(v)$, it is above $\ell(u)$ and therefore p cannot be contained in $r(v)$.

5.2 Section b

We can use $2-d$ tree storing a set of points P to find all points in some query rectangle R as follows. We start at the root and for each internal node v which we visit we perform the following. If the rectangle $r(u)$ corresponding to the right child u of v is contained in R then we report all points in the subtree of u. Similarly, if the rectangle $r(w)$ corresponding to the left child w of v is contained

in R then we report all points in the subtree of w. If R intersects $r(u)$ but does not contain it then we continue the search in u. Similarly, if R intersects $r(w)$ but does not contain it then we continue the search in w. When we reach a leaf z then we report the point stored at z if it is contained in R.

Let n be the size of P and let k be the number of points that are contained in a query rectangle R. Notice that there are only two kinds of nodes we visit:

1. Nodes v such that $r(v) \subseteq R$. Since we stop the search at these nodes, the total number of these nodes we visit is $O(k)$.

2. Nodes v such that $r(v) \cap R \neq \emptyset$ but $r(v) \not\subseteq R$. According to the claim that follows, the total number of these nodes we visit is $O(\sqrt{n})$.

The total amount of work we perform in nodes of the first type is $O(k)$ because we just report all the points that are contained in R. For each node of the second type we perform $O(1)$ work because we just check the conditions for its children and continue the search at them. Thus the running time of the algorithm is $O(\sqrt{n}+k)$.

Proposition 12. *A rectangle R will intersect (and not be contained) with at most $O(\sqrt{n})$ rectangles corresponding to nodes in a $2-d$ Tree.*

Proof. According to the next claim, each side of R passes through at most $O(\sqrt{n})$ rectangles corresponding to nodes in a $2-d$ Tree. Thus R will intersect at most $4O(\sqrt{n}) = O(\sqrt{n})$ rectangles corresponding to nodes in a $2-d$ Tree. □

Proposition 13. *A vertical / horizontal line ℓ passes through at most $O(\sqrt{n})$ rectangles corresponding to nodes in a $2-d$ Tree.*

Proof. Assume without loss of generality that ℓ is a vertical line, and let a be a node corresponding to a vertical line $\ell(a)$. It is obvious that ℓ can intersect exactly one of a's children, denote this child by c. ℓ will intersect both of $c's$ children because c corresponds to a horizontal line. So ℓ will intersect at most 2 of a's grandchildren. According to the next claim the tree's height is $\leq \log n+1$, and therefore the number of rectangles corresponding to nodes in a $2-d$ Tree, ℓ will pass through is at most:

$$2\sum_{i=0}^{(\log n+1)/2} 2^i = 2\frac{2^{(\log n+1)/2}-1}{2-1} = 2(2^{\frac{1}{2}\log n+\frac{1}{2}}-1) \leq 3\sqrt{n} = O(\sqrt{n})$$

□

Proposition 14. *The height of a $k-d$ Tree with n leaves is $H(n) = \lceil \log n \rceil$.*

Proof. By induction on n. For $n=1$ the height is $H(1) = 0 = \lceil \log 1 \rceil$. Assume the claim is correct for all $k < n$ and prove for n. By the definition of the $k-d$ Tree:

$$H(n) = 1 + H(\left\lceil \frac{n}{2} \right\rceil)$$

Consider the two possible cases:

1. n is even. Then by the induction hypothesis we get: $H(n) = 1 + H(\frac{n}{2}) = 1 + \lceil \log \frac{n}{2} \rceil = 1 + \lceil \log n - 1 \rceil = \lceil \log n \rceil$.

2. n is odd. Then by the induction hypothesis we get: $H(n) = 1 + H(\frac{n+1}{2}) = 1 + \lceil \log \frac{n+1}{2} \rceil = 1 + \lceil \log(n+1) - 1 \rceil = \lceil \log(n+1) \rceil = \lceil \log n \rceil$. Where the last equality follows because n is odd so $\log n$ cannot be an integer (and then it is easy to check that this is correct).

□

5.3 Section c

A $k-d$ tree T is a binary tree that represents a set P of points in $\mathbb{R}^k$ as follows. Every leaf stores a point of P. Every non-leaf node v at a distance $i(\mod k)$ from the root corresponds to a hyperplane $\ell(v)$ that is parallel to the ith axis such that $\lfloor \frac{m}{2} \rfloor$ of the m points in v's subtree are above $\ell(v)$ (with respect to the ith axis) and are stored at the left subtree of v, and $\lceil \frac{m}{2} \rceil$ of the of the m points in v's subtree are below $\ell(v)$ (with respect to the ith axis) and are stored at the right subtree of v.

Just like in the $2-d$ tree, we can show that for a $k-d$ tree the leaves of a subtree rooted by a node v correspond to the points of P that are contained in a box $r(v)$ such that:

$$r(v) = \left\{p = (x_1, \ldots, x_k) \in \mathbb{R}^k : \forall i \in \{1, \ldots, k\}. x_{i,min} \leq x_i \leq x_{i,max}\right\}$$

Where $x_{i,min}$ and $x_{i,max}$ are defined as follows. Let $\left\{p_1 = (x_1^1, \ldots, x_k^1), \ldots, p_m = (x_1^m, \ldots, x_k^m)\right\}$ be the points in v's subtree. We define:

$$x_{i,min} = \min_{1 \leq j \leq m} x_i^j$$

$$x_{i,max} = \max_{1 \leq j \leq m} x_i^j$$

We can use $k-d$ tree storing a set of points P to find all points in some query box R as follows. We start at the root and for each internal node v which we visit we perform the following. If the box $r(u)$ corresponding to the right child u of v is contained in R then we report all points in the subtree of u. Similarly, if the box $r(w)$ corresponding to the left child w of v is contained in R then we report all points in the subtree of w. If R intersects $r(u)$ but does not contain it then we continue the search in u. Similarly, if R intersects $r(w)$ but does not contain it then we continue the search in w. When we reach a leaf z then we report the point stored at z if it is contained in R.

5.4 Section d

Consider the following algorithm to find the nearest neighbor of a point $x \in \mathbb{R}^k$ using a $k-d$ Tree T. We maintain a pointer to the point that is currently closest to x. We denote it by *currBest* (initialized to *null*) and its distance from x

by $currDist$ (initialized to ∞). We traverse the tree recursively starting at the root as follows (the current node is denoted by $curr$):

1. if $curr$ is a leaf
   (a) if $curr$ is closer to x then $currBest$ update $currBest \leftarrow curr.point$, $currDist \leftarrow ||x - curr.point||$
2. if $x[curr.axis] > curr.value$ ($curr$ is splitting the space with respect to $axis$ at $value$)
   (a) if the distance between x and $r(curr.left)$ is at most $currDist$ continue the search recursively in $curr.left$
   (b) if the distance between x and $r(curr.right)$ is at most $currDist$ continue the search recursively in $curr.right$
3. else
   (a) if the distance between x and $r(curr.right)$ is at most $currDist$ continue the search recursively in $curr.right$
   (b) if the distance between x and $r(curr.left)$ is at most $currDist$ continue the search recursively in $curr.left$

At the end we return $currBest$ as the nearest neighbor of x.

The idea of the algorithm is to traverse the whole tree but with two modifications:

1. Prune subtrees once their rectangle say that they can't contain any point closer then $currBest$.
2. Search the subtrees in order that maximizes the chance for pruning, i.e. use the tree splits to traverse first nodes with greater chance to be close to x.

The correctness of the algorithm is obvious since we traverse every point unless it is contained in a rectangle that is farther than the point we already have. Unfortunately we may still have to traverse the whole tree. According to the claim that follows the size of the tree is $O(n)$, and therefore the worst case running time is $O(n)$.

Proposition 15. *The size of a $k-d$ Tree with n leaves is $S(n) = 2n - 1$.*

Proof. By induction on n. For $n = 1$ the size is $S(1) = 1 = 2 \cdot 1 - 1$. Assume the claim is correct for all $k < n$ and prove for n. By the definition of the $k-d$ Tree:

$$S(n) = 1 + S(\left\lfloor \frac{n}{2} \right\rfloor) + S(\left\lceil \frac{n}{2} \right\rceil)$$

Consider the two possible cases:

1. n is even. Then by the induction hypothesis we get: $S(n) = 1 + S(\frac{n}{2}) + S(\frac{n}{2}) = 1 + 2\frac{n}{2} - 1 + 2\frac{n}{2} - 1 = 2n - 1$.

2. n is odd. Then by the induction hypothesis we get: $S(n) = 1 + S(\frac{n-1}{2}) + S(\frac{n+1}{2}) = 1 + 2\frac{n-1}{2} - 1 + 2\frac{n+1}{2} - 1 = 2n - 1$.

□