COL351: Assignment-3

Abhinav Jain
2019CS10322
Himanshu Gaurav Singh
2019CS10358

November 29, 2021

Problem 1

Design an $O(nlogn)$ time Divide-and-Conquer algorithm to compute the convex hull of a set P of n input points $\{(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)\}$.
Solution. Before providing the exact algorithm and proof of correctness, the algorithm can be broadly summarised as follows:

We will present a divide and conquer algorithm. First we will sort the points in P by there x co-ordinate. Now we perform the divide step i.e split P at the median of x co-ordinate into roughly 2 equal parts P_1 and P_2 and recursively compute the convex hull of sets P_1 and P_2. Now we perform the merge step i.e combine the convex hull of P_1 and P_2. We perform this step by computing the upper and lower tangent of P_1 andP_1 and then removing the points lying between the tangents. The algorithm decribed returns the vertices lying on convex polygon (i.e the convex hull) in clockwise order. The above algorithm assumes that no two points lie on the same vertical line and no three points are collinear. These cases can be covered separately by slightly modifying the points.
Note that the procedure $Clockwise(a, b, c)$ returns true if points a,b,c are in clockwise orientation.

Algorithm 1 $ConvexHull(P)$

```
1: if $|P| \leq 3$ then
2:     Sort points of $P$ in clockwise order{Choose a point other than those in $P$ with $x$ and $y$ coordinate less than all the vertices. For each vertex $(x_i, y_i)$ sort the points in $P$ in decreasing order of $(y - y_i)/(x - x_i)$}
3:     Return $P$
4: else
5:     $x_m$ = Median of $\{x_1, x_2, ..., x_n\}$
6:     $P_1 = \{x_1, x_2, ..x_n/2\}$
7:     $P_2 = \{x_{n/2+1}, ..x_n\}$
8:     $H_1$,$H_2 = ConvexHull(P_1), ConvexHull(P_2)$
9:     $ab$, $cd = UpperTangent(H_1), LowerTangent(H_2)$
10:    $H'_1 = H_1 - \{set\ of\ points\ in\ H_1\ lying\ to\ the\ right\ of\ line\ ac\}$
11:    $H'_2 = H_2 - \{Set\ of\ points\ lying\ to\ the\ left\ of\ line\ bd\}$
12:    return $H'_1 \cup H'_2$
13: end if
```

Proof of correctness We will assume that no three points are collinear and no two points lie on a vertical line. (Otherwise we can add slight noise to the points and get the convex hull). First we will proof the correctness of the **UpperTangent** procedure.
The following claims will help us in the proof.
Claim-1 : The finite line segment ab doesn't not intersect the interior of the polygon H_1 and H_2, holds after each iteration of inner **while** loop from **line 6-8**.
Proof : We will proceed by induction.
Base Case : Observe that the base case holds as initially a is the rightmost point of H_1 and b is the leftmost point of H_2.
Induction Step : Suppose that the claim holds after some iteration i.e the line segment ab doesn't intersect the

Algorithm 2 UpperTangent(H_1,H_2)

```
1:  $a$ = Rightmost Point of $H_1$(with largest x coordinate)
2:  $a'$ = Predecessor of $a$ in clockwise direction{cyclically previous element in list $H_1$}
3:  $b$ = Leftmost Point of $H_2$(with smallest x coordinate)
4:  $b'$ = Successor of $b$ in clockwise direction{cyclically next element in list $H_2$}
5:  while Clockwise($a', b, a$) or Clockwise($b, a, b'$) do
6:      while Clockwise($a', b, a$) do
7:          $a, a'$ = Predecessor of $a$ i cw direction, Predecessor of $a'$ in cw direction
8:      end while
9:      while Clockwise($b, a, b'$) do
10:         $b,\ b'$ = Successor of $b$ in cw direction, Successor of $b'$ in cw direction
11:     end while
12: end while
13: return $ab$
```

Algorithm 3 LowerTangent(H_1,H_2)

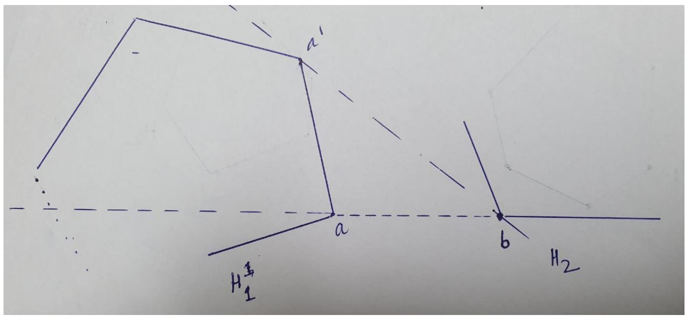
```
1:  $a$ = Rightmost Point of $H_1$(with largest x coordinate)
2:  $a'$ = Successor of $a$ in clockwise direction{cyclically next element in list $H_1$}
3:  $b$ = Leftmost Point of $H_2$(with smallest x coordinate)
4:  $b'$ = Predecessor of $b$ in clockwise direction{cyclically previous element in list $H_2$}
5:  while Clockwise($a, b, a'$) or Clockwise($b', a, b$) do
6:      while Clockwise($a, b, a'$) do
7:          $a, a'$ = Successor of $a$ in cw direction, Successor of $a'$ in cw direction
8:      end while$a$ = Predecessor of $a'$
9:      while Clockwise($b', a, b$) do
10:         $b, b'$ = Predecessor of $b$ in cw direction, Predecessor of $b'$ in cw direction
11:     end while
12: end while
13: return $ab$
```

Algorithm 4 *Clockwise*(p_1, p_2, p_3)

```
1: if $(p_2.y - p_1.y) * (p_3.x - p_2.x) - (p_3.y - p_2.y) * (p_2.x - p_1.x) > 0)$ then
2:     return True
3: else
4:     return False
5: end if
```

polygon H_1 and H_2. Consider the next iteration, since Clockwise(a', b, a) is true and H_1 lies strictly to the **left** of point b the line segment $a'b$ will not intersect the interior of H_1 and H_2 as can be observed from the figure below. Also, from the above claim we can conclude that this while loop must terminate, because for a given b (lying outside H_1) there are only a finite number of points a on H_1 for which line segment ab doesn't lie in interior of $H1$ and we are iterating over them in a strictly anticlockwise order.

Claim-2 : After every iteration of inner **while** loop (from line 6 - 8) the number of points in H_1 lying below or on the line ab strictly increases.
Proof : Observe that if Clockwise(a', b, a) is true then a is assigned a' which is its cyclic predecessor in H_1. Since H_1 is a convex polygon and it lies strictly to the **left** of point b, the number of points in H_1 below or on the line a'b will be strictly more that number of points below or on the line ab.

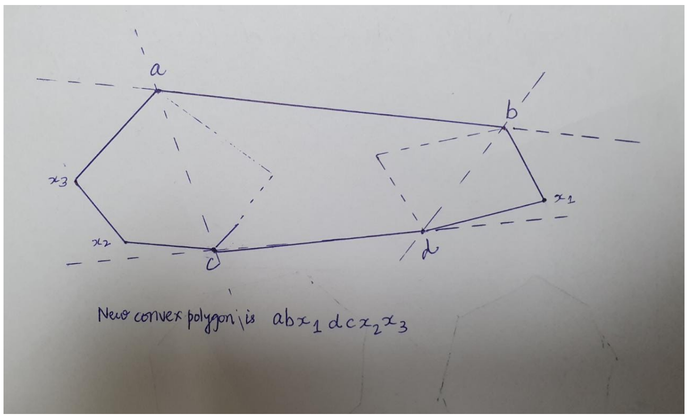


Similarly, we can prove analogous claims for inner **while** loop from line 9-11.
By **Claim-2** we conclude that after each iteration of outer **while** loop from line 5-12 the number of points in H_1 and H_2 lying below or on the line ab strictly increases.Since there are a finite number of points in H_1 and H_2 we conclude that the outer while loop must terminate.

Lemma 0.1. *The line segment ab returned by $UpperTangent(H_1, H_2)$ is the upper tangent of H_1 and H_2.*

Proof. For the sake of contradiction suppose that ab is not the upper tangent of H_1 and H_2. Then there must lie a point say p on H_1 or H_2 which is above the line ab. WLOG assume that p lies on H_1. Therefor Clockwise(p, b, a) is true, and since b lies to the right of H_1, Clockwise(a', b, a) must be true where a' is the cyclic predecessor of a. Hence we achieve a contraction as the termination condition of while loop is not satisfied. □

Using similar arguments the correctness of the procedure $LowerTangent(H_1, H_2)$ holds.



Now we are ready to prove the correctness of the algorithm $ConvexHull$.

Lemma 0.2. *The polygon returned by the procedure ConvexHull(P) is the convex hull of the points of P.*

Proof. We will proceed by induction.
Base Case : When $|P| \leq 3$ the claim holds trivially as P itself is the convex hull in this case.
Induction Step : Assume that H_1 and H_2 are the convex hull of P_1 and P_2.
First we will show that all points of P indeed lie inside the polygon formed by $H_1' \cup H_2'$. Let ab be the upper tangent of H_1 and H_2 and let cd be the lower tangent of H_1 and H_2. By definition of the tangents, all the points of P lie in between the upper and lower tangent. Hence, all the points to the right of ac in $P1$ lie inside $H_1' \cup H_2'$ and all the points to the left of bd in $P2$ lie inside $H_1' \cup H_2'$. Since H_1 is the convex hull of $P1$ and H_1' contains the points of H_1 to the left of ac, therefor all points of $P1$ to the right of ac are contained in $H_1' \cup H_2'$. Similarly all points to the left of bd are contained in $H_1' \cup H_2'$. Since $P = P_1 \cup P_2$ all points of P are contained in $H_1' \cup H_2'$.
Now we will show that $H_1' \cup H_2'$ is also convex. For this, we will show that all the angles of the polygon are less than 180 degree. Let $H_1' = \{c, p_1, p_2, ...p_k, a\}$ and $H_2' = \{b, q_1, q_2, ...q_j, d\}$. Since, from the induction hypothesis, H_1 is a convex polygon, the angle formed by the vertices p_i, p_{i+1}, p_{i+2} is less than 180 for all i between 1 and $k-2$. The angle p_{k-1}, p_k, a must be less than 180 as well because a also lies on H_1. Similarly, since H_2 is convex polygon the angle formed by the vertices q_i, q_{i+1}, q_{i+2} is less than 180 for all i between 1 and $j-2$.The angle q_{j-1}, qj_k, d must be less than 180 as well because a also lies on H_2. The angle p_k, a, b must be less than 180 as the condition Clockwise(p_k, a, b) must be true as ab is upper tangent. Similarly the angles formed by a, b, q_1 and q_k, d, c and d, c, p_1 are less than 180. Hence all angles of the polygon $H_1' \cup H_2'$ are less than 180 implying that it is convex. Hence $H_1' \cup H_2'$ is the convex hull of the set P.

□

Time complexity. Assume that the time complexity for a polygon P is $T(n)$. Observe that the **if** clause takes O(1) time. Median computation takes O(1) time as we have sorted the points based on there x co-ordinate before calling the procedure ConvexHull.In the **else** clause, P_1,P_2 have size $n/2$, where n is the size of P. The lists P_1 and P_2 can be computed in O(1) time by linearly passing over list P. The **line 8** takes 2*T(n/2) time. The computation of $H_1^{'}$ and $H_2^{'}$ from H_1, H_2 also takes $O(n)$ time. Now we will show that the procedure $UpperTangent$ takes O(n) time. Observe that in each iteration of the inner while loops (from line 6-8 and line 9-11) either the point a moves in counter-clockwise direction or point b moves in clockwise direction. Since points a and b can take atmost n distinct values, $UpperTangent$ take O(n) steps. Since points a and b can take atmost $n/2$ distinct values, hence $UpperTangent$ takes $O(n)$ steps. Similarly we can show that the procedure $UpperTangent$ takes $O(n)$ time.
Thus, we get the recurrence $T(n) = 2 * T(n/2) + O(n)$. This is a famous recurrence and it's solution is $O(nlog(n))$. Since sorting the points initially can be done in $O(nlog(n))$, hence the final time complexity is $O(nlog(n))$.

Problem 2 - Particle Interaction

Solution. We will solve this problem using fast multiplication of polynomials. For each j we will compute

$$E_j = \sum_{i<j} \frac{Cq_i}{(j-i)^2} - \sum_{i>j} \frac{Cq_i}{(j-i)^2}$$

, then F_j can be computed as $E_j q_j$.

In order to compute E_j we break it down to E_{j_1} and E_{j_2} such that

$$E_{j_1} = \sum_{i<j} \frac{Cq_i}{(j-i)^2} \quad and \quad E_{j_2} = \sum_{i>j} \frac{Cq_i}{(j-i)^2}$$

, then $E_j = E_{j_1} - E_{j_2}$.
First we will compute E_{j_1}. Consider the polynomials

$$Q_1(x) = \sum_{i=1}^{n} q_i x^i \quad and \quad P_1(x) = \sum_{i=1}^{n} \frac{x^i}{i^2}$$

Let $R_1(x) = Q_1(x)P_1(x)$. Let r_{j_1} be the coefficient of x^j in $R_1(x)$. Observe that for $1 \le j \le n$ we have $r_{j_1} = \sum_{i=1}^{j-1} \frac{Cq_i}{(j-i)^2}$. Hence $r_{j_1} = E_{j_1}$. Using the fast multiplication algorithm we can compute the $R_1(x)$ in $O(nlogn)$, therefore we can compute E_{j_1} in $O(nlogn)$.

Now we will compute E_{j_2}. Consider the polynomials

$$Q_2(x) = \sum_{i=1}^{n} q_i x^i \quad and \quad P_2(x) = \sum_{i=1}^{n} \frac{x^{n-i}}{i^2}$$

Let $R_2(x) = Q_2(x)P_2(x)$. Let r_{j2} be the coefficient of x^{n+j} in $R_2(x)$. Observe that for $1 \le j \le n$ we have $r_{j_2} = \sum_{i=j+1}^{n} \frac{Cq_i}{(j-i)^2}$. Hence $r_{j_2} = E_{j_2}$. Using the fast multiplication algorithm we can compute the $R_2(x)$ in $O(nlogn)$, therefore we can compute E_{j_2} in $O(nlogn)$.

Now for each j we can compute $F_j = (E_{j_1} - E_{j_2})q_j$. This step will take $O(n)$ operations.
Hence we can compute F in $O(nlogn)$ operations. The following algorithm summarises the above discussion(proof of correctness and time complexity). The algorithm takes input a pair of lists $\{q_1, q_2, ..q_n\}$ and $\{x_1, x_2, ..x_n\}$ and outputs a list $F_1, F_2, .., F_n$. $FAST-MULTIPLY(p_1, p_2)$ is efficient $O(nlog(n))$ implementation of polynomial multiplication using Fourier Transform.

Algorithm 5 $ParticleInteraction(q)$

```
1: Construct the Polynomials
2: $Q_1 \leftarrow \{0, q_1, q_2, ..q_n\}$
3: $P_1 \leftarrow \{0, 1/1^2, 1/2^2, ..1/n^2\}$
4: $R \leftarrow FAST-MULTIPLY(P_1, Q_1)$
5: $E_{left} = \{R_1, R_2, ..R_n\}$
6: $Q_2 \leftarrow \{0, q_1, q_2, ..q_n\}$
7: $P_2 \leftarrow \{1/n^2, 1/(n-1)^2, 1/(n-2)^2, ..1/1^2, 0\}$
8: $R \leftarrow FAST-MULTIPLY(Q_2, P_2)$
9: $E_{right} = \{R_1, R_2, ..R_n\}$
10: $Initialise\ list\ F\ of\ size\ n$
11: for $i \in \{1, 2, ...n\}$ do $F_i \leftarrow (E_{left}(i) - E_{right}(i)) * q_i$
12: end for
13: return $F$
```

Problem 3

Solution.

0.1 (a)

Observe that the adjacency matrix of an unweighted graph completely and uniquely characterises it. Denote the adjacency matrix of G by A_G. We claim that $A_H = sgn(A_G^2 + A_G)$, where $sgn(A)_{ij} = 1\ if\ A_{ij} > 0\ and\ 0\ otherwise$.

Proof. We will prove that $(A_G^2)_{ij} > 0$ iff there exists k such that $(i,k),(k,j) \in G$.
Consider an arbitrary element $(A_G^2)_{ij}$, $(A_G^2)_{ij}) = \sum_{k=1}^{k=n} (A_G)_{ik} * (A_G)_{kj}$, $(A_G^2)_{ij} > 0$ implies that one of the summands in the expression above is non-zero. Hence, there exists k such that $(A_G)_{ik} > 0, (A_G)_{kj} > 0$. To prove the converse, observe that if there exists some such k_0, then the term $(A_G)_{ik_0} * (A_G)_{k_0 j}$ will be non-zero. Since, all terms in the summation will be non-negative, $(A_G^2)_{ij} > 0$.
Thus, from the above and the fact that A_G and A_G^2 both have all entries non-negative, $(A_G^2 + A_G)_{ij} > 0$ if and only if either $(i,j) \in G$(from the definition of A_G) or there exists k such that $(i,k),(k,j) \in G$. Hence, $sgn(A_G^2 + A_G)$ is the adjacency matrix of H. □

Observe that (A_G^2) can be computed in $O(n^\omega)$ time using matrix multiplication. Addition with A_G and applying sgn function both takes $O(n^2)$ time. Since $\omega \geq 2$. Total time complexity is $O(n^\omega)$.

0.2 (b)

Observe that if there is a path between two vertices u, v in G, then there is a path between them in H, since any edge in G is in H as well. Also, if there is a path from u to v in H, then there is also a path from u to v in G since for every edge in H, either that edge is in G or there are a pair of edges that connect the endpoints of that edge in G. Thus, connectivity in G is equivalent to connectivity in H.
First we will deal with the trivial case when x and y are disconnected in G. In this case $(D_G)_{x,y} = \infty$ and $(D_H)_{x,y} = \infty$ since disconnectivity in G implies disconnectivity in H. Thus, $D_H(x,y) = \lceil D_G(x,y)/2 \rceil$ in this case.

From now on, we will assume that x, y are connected.
For any arbitrary $x, y \in V$, consider the shortest path $P = \{x_0(= x), x_1, x_2, ..y\}$ of length $D_G(x,y)$ in G. Observe that for all valid i, $\{(x_i, x_{i+1}), (x_{i+1}, x_{i+2})\} \in G.E$, thus $\{(x_i, x_{i+1}), (x_{i+1}, x_{i+2}), (x_i, x_{i+2})\} \in H$. Thus, there exists a path $P' = \{x_0, x_2, ..y\}$ in H. If $D_G(x,y)$ is even, for every two edges in P, there is one edge in P'. Thus, length of P' is $D_G(x,y)/2$. Hence, $D_H(x,y) \leq D_G(x,y)/2$, if $D_G(x,y)$ is even, since $D_H(x,y)$ is the length of the shortest path from x to y in H. In the case P has odd length, we have $P' = \{x_0, x_1, x_3, ..y\}$ which has length $(D_G(x,y)+1)/2$. Thus,$D_H(x,y) \leq (D_G(x,y)+1)/2$, if $D_G(x,y)$ is odd. From both the even and odd case, we can infer that $D_H(x,y) \leq \lceil D_G(x,y)/2 \rceil$ for all x, y. **−1**

Now, consider the shortest path from x to y, $P = \{x_0 = (x), x_1, x_2, ..y\}$ in H of length $D_H(x,y)$. For any $(x_i, x_{i+1}) \in P$, either $(x_i, x_{i+1}) \in G$ or there exists z_i such that $\{(x_i, z_i), (z_i, x_{i+1})\} \in G$. We can create a path $P' = \{....x_i, x_{i+1}, ..x_j, z_j, x_{j+1}, ..\}$ in G by replacing each edge in P with the corresponding edge or pair of edges present in G. Observe that the length of P' is at most twice the length of P. Thus, $|P'| \leq 2 * D_H(x,y)$. Since $D_G(x,y)$ is the shortest path in G, $D_G(x,y) \leq |P'| \leq 2 * D_H(x,y)$, or $D_H(x,y) \geq D_G(x,y)/2 \geq \lceil D_G(x,y)/2 \rceil$(since $\lceil x \rceil$ is the greatest integer greater than or equal to x). **−2**
From **1** and **2**, we can conclude that $D_H(x,y) = \lceil D_G(x,y)/2 \rceil$ for all x, y.

0.3 (c)

The following lemmas will help in proving the final result.

Lemma 0.3. *For any undirected, unweighted graph G, for all $(u,v) \in G.E$, $|D_G(u,x) - D_G(v,x)| \leq 1$ for any $x \in G.V$.*

Proof. We proceed by contradiction. Assume that there exists some $(u,v) \in G.E$ and $x \in G.V$ such that $|D_G(u,x) - D_G(v,x)| > 1$. WLoG, assume that $D_G(u,x) - D_G(v,x) > 1$,consider the shortest path P from x to v of length $D_G(v,x)$, then $P \cup (u,v)$ is a walk from x to u(it will or will not be a simple path if u lies on P or not). In any case, this implies that the shortest path from x to u has length at most that of $P \cup (u,v)$. Thus $D_G(u,x) \leq D_G(v,x) + 1$, which contradicts the assumption. □

Lemma 0.4. *For any $v \in N_G(u)$(in other words, $(u,v) \in G.E$) and for any $x \in G.V$, if $D_H(v,x) > D_H(u,x)$ then $D_H(w,x) \geq D_H(u,x)$ for all $w \in N_G(u)$. Similarily, if for some x, $D_H(v,x) < D_H(u,x)$ then $D_H(w,x) \leq D_H(u,x)$ for all $w \in N_G(u)$.*

Proof. Using the lemma above and the assumption $D_H(v,x) > D_H(u,x)$, we can infer that $D_H(v,x) = D_H(u,x) + 1$. If $N_G(u) = \{v\}$, then the claim follows directly from the hypothesis. Otherwise, consider any $w \in N_G(u)$ other than v. Observe that $D_H(w,x)$ cannot be less than $D_H(u,x) - 1$. This follows from the lemma proved above and that $(w,u) \in G.E$.
From the definition of H, $(w,v) \in H.E$.Again using the above lemma, $|D_H(w,x) - D_H(v,x)| \leq 1$. If $D_H(w,x) = D_H(u,x) - 1$, then $|D_H(w,x) - D_H(v,x)|$ will equal 2, violating the above inequality.
Thus, we can conclude that $D_H(w,x) \geq D_H(u,x)$ for all $w \in N_G(u)$ if $D_H(v,x) > D_H(u,x)$ for some v.

Similarly, $D_H(v,x) < D_H(u,x)$ implies that $D_H(v,x) = D_H(u,x) + 1$.
If $N_G(u) = \{v\}$, then the claim follows directly from the hypothesis. Otherwise, consider any $w \in N_G(u)$ other than v. Observe that $D_H(w,x)$ cannot be greater than $D_H(u,x) + 1$. This follows from the lemma proved above and that $(w,u) \in G.E$.
From the definition of H, $(w,v) \in H.E$.Again using the above lemma, $|D_H(w,x) - D_H(v,x)| \leq 1$. If $D_H(w,x) = D_H(u,x) + 1$, then $|D_H(w,x) - D_H(v,x)|$ will equal 2, violating the above inequality.
Thus, we can conclude that $D_H(w,x) \leq D_H(u,x)$ for all $w \in N_G(u)$ if $D_H(v,x) < D_H(u,x)$ for some v. ☐

Consider an arbitrary element $M_(i,j)$ of M, $M_(i,j) = \sum_{k=1}^{k=n} (D_H)_{ik} * (A_G)_{kj}$. Since $(A_G)_{ij} = 1$ iff $(i,j) \in G.E$ and 0 otherwise, this expression reduces to:

$$M_{ij} = \sum_{k \in N_G(j)} (D_H)_{ik}$$

If x and y are disconnected in G, then $D_G(x,y) = D_H(x,y) = \infty$. Thus,$D_G(x,y) = 2 * D_H(x,y) = 2 * D_H(x,y) - 1$ in this case.

Now we examine the non-trivial case.

From the above problem, we know the fact:

$D_H(x,y) = \lceil D_G(x,y)/2 \rceil$. Thus, for any $x, y \in G.V$, $D_G(x,y)$ is either $2 * D_H(x,y)$ or $2 * D_H(x,y) - 1$. –**1**

We will proceed case by case:

Case 1: $M_{xy} < degree_G(y) * (D_H)_{xy}$.
Since, $\sum_{z \in N_G(y)} (D_H)_{xz} < degree_G(y) * (D_H)_{xy}$, there must exist atleast one z_0 in $N_G(y)$ for which $(D_H)_{xz_0} < (D_H)_{xy}$. Otherwise, $\sum_{z \in N_G(y)} (D_H)_{xz} \geq \sum_{z \in N_G(y)} (D_H)_{xy} = degree_G(y) * (D_H)_{xy}$, contradicting the hypotheses.

This fact along with Lemma 0.4 gives us that $(D_H)_{z_0 x} = (D_H)_{xy} - 1$. From **1**, we can say that:

$$\lceil (D_G)_{z_0 x}/2 \rceil = \lceil (D_G)_{xy}/2 \rceil - 1 \text{ –}\mathbf{2}$$

Also, since $(z_0, y) \in G.E$,

$$|(D_G)_{z_0 x} - (D_G)_{xy}| \leq 1 \text{ –}\mathbf{3}$$

Assume $(D_G)_{xy}$ is even, $2 * k$ for some whole number k, then from **2**, $\lceil (D_G)_{z_0 x}/2 \rceil = k - 1$, which implies, $(D_G)_{z_0 x}$ is either $2 * k - 2$ or $2 * k - 3$ giving $|(D_G)_{z_0 x} - (D_G)_{xy}|$ equal to 2 or 3, both of which contradict **3**. Thus, $(D_G)_{xy}$ is odd. From **1**, $(D_G)_{xy} = 2 * D_H(x,y) - 1$.

Case 2: $M_{xy} \geq degree_G(y) * (D_H)_{xy}$.

In this case, our goal is to prove that the stated condition implies that $(D_G)_{xy} = 2 * D_H(x,y)$.
We will prove the contra-positive of the above, which is that if $(D_G)_{xy}$ is not equal to $2 * D_H(x,y)$, then $M_{xy} < degree_G(y) * (D_H)_{xy}$. This will be equivalent to our goal.

Assume that, $(D_G)_{xy}$ is not equal to $2 * D_H(x,y)$, then from **1**, $(D_G)_{xy} = 2 * D_H(x,y) - 1$. Consider the shortest path $P = (x_0(= x), x_1, .., z, y)$ in G from x to y. Here z is the vertex in P which is adjacent to y. Since z lies on the shortest path from x to y and is adjacent to y, $D_{Gxz} = D_{Gxy} - 1 = 2 * (D_H)_{xy} - 2$. From **1**, we get:

$$D_{Hxz} = \lceil (D_G)_{xz}/2 \rceil = (D_H)_{xy} - 1. \; \mathbf{-4}$$

Using **4** and Lemma 0.5, we can infer that for all $w \in N_G(y)$, $(D_H)_{xw} \le (D_H)_{xy}$. Thus, $M_{xy} = \sum_{w \in N_G(y)} (D_H)_{wx} \le \sum_{w \in N_G(y)} (D_H)_{yx} = degree_G(y) * (D_H)_{xy}$. Since for z(one of the possible values of w), $D_{Hxz} = (D_H)_{xy} - 1$ is strictly less than $(D_H)_{xy}$, the above inequality for M_{xy} becomes strict as well. Hence, $M_{xy} < degree_G(y) * (D_H)_{xy}$ as desired. Since, $\sum_{z \in N_G(y)} (D_H)_{xz} \ge degree_G(y) * (D_H)_{xy}$, there must exist atleast one z_0 in $N_G(y)$ for which $(D_H)_{xz_0} < (D_H)_{xy}$. Otherwise, $\sum_{z \in N_G(y)} (D_H)_{xz} \ge \sum_{z \in N_G(y)} (D_H)_{xy} = degree_G(y) * (D_H)_{xy}$, contradicting the hypotheses.

0.4 (d)

The following algorithm can be used to compute D_G from D_H. The proof of correctness of the algorithm follows

Algorithm 6 $G - From - H(D_H, A_G)$

```
1: $M = MULTIPLY(D_H, A_G)$
2: $Initialise\ (D_G)_{ij} = \infty\ for\ all\ i, j$
3: for $y = 1, 2, ..., n$ do
4:     $degree_y = \sum_{z=1}^{z=n} (A_G)_{zy}$
5:     for $x = 1, 2, ..., n$ do
6:         if $M_{xy} \neq \infty$ then
7:             if $M_{xy} \ge (D_H)_{xy} * degree_y$ then
8:                 $(D_G)_{xy} \leftarrow 2 * (D_H)_{xy}$
9:             else
10:                $(D_G)_{xy} \leftarrow 2 * (D_H)_{xy} - 1$
11:            end if
12:        end if
13:    end for
14: end for
15: $return\ D_G$
```

straightforward from the proof in 3(c).
For the time complexity, observe that the computation of M takes $O(n^\omega)$ time. The inner **for** loop takes $O(n^2)$ time. The calculation of $degree_y$ in l**line 4** takes $O(n^2)$ time. Since, $\omega \ge 2$, the total time complexity is $O(n^\omega)$

0.5 (e)

The following algorithm computes D_G.

Algorithm 7 $ALL - PAIR - SHORTEST - PATHS(A_G, i)$

```
1: if $i \ge log(\lceil |G.V| \rceil)$ then
2:     $Initialise\ D_{ij} \leftarrow 1\ if\ (A_G)_{ij} = 1; \leftarrow 0\ if\ i = j; \leftarrow \infty\ otherwise$
3:     $return\ D_G$
4: else
5:     $A_H \leftarrow sgn(A_G^2 + A_G)$
6:     $D_H \leftarrow ALL - PAIR - SHORTEST - PATHS(A_H, i + 1)$
7:     $D_G \leftarrow G - From - H(D_H, A_G)$
8: end if
Call $ALL - PAIRS - SHORTEST - PATHS(A_G, 0)$
```

Proof of correctness

Lemma 0.5. *For any undirected graph G, consider the sequence, $G_0(= G), G_1, G_2, ..$ where $G_i = H(G_{i-1})$ is an undirected graph. Then $G_i.V = G.V$ and $(u, v) \in G_i.E$ iff $D_G(u, v) \le 2^i$. The following lemma will help us in the proof.*
$G_{\lceil log(n) \rceil}$ is the transitive closure of G.

Proof. We will proceed by induction.
Base Case. $i = 1$ By the definition of H, $H(G).E$ has (u, v) iff they have a common neighbour or they are adjacent themselves that occurs iff $D_G(u, v) \le 2$. Thus, the base case is true. **Induction Step.** From the above

problem we know that $D_{G_(i+1)}(u, v) = \lceil D_{G_(i)}(u, v)/2\rceil$. From the induction hypothesis, we know that $(u, v) \in G_i.E$ iff $D_G(u, v) \leq 2^i$. Consider any two adjacent vertices (u, v) in G_{i+1}.Then, we must have one of the 2 cases :
1.) Vertices u and v are adjacent in G_i. By induction hypothesis we have $D_G(u, v) \leq 2^i$, hence $D_G(u, v) \leq 2^{i+1}$ in this case.
2.) There exists a vertex w such that u,w are adjacent in G_i and w,v are adjacent in G_i . By induction hypothesis we have $D_G(u, w) \leq 2^i$ and $D_G(w, v) \leq 2^i$, hence by triangle inequality $D_G(u, v) \leq D_G(u, w) + D_G(u, v) \leq 2^{i+1}$. Hence, the induction is complete.

From the definition, $G_{\lceil log(n)\rceil} = \{(u, v) iff D_G(u, v) \leq 2^{\lceil log(n)\rceil}\}$. Observe that $2^{\lceil log(n)\rceil} \geq n$, since the shortest path between any two vertex will not be greater than n, $G.E$ contains (u, v) iff u, v are reachable from each other. Hence, $G_{\lceil log(n)\rceil}$ is the transitive closure of G. □

For proving that $ALL-PAIR-SHORTEST-PATHS(A_G)$ correctly computes D_G, we will induct on the sequence $G_0(= G), G_1, G_2, ...$ as defined above.

Base case. $i = \lceil log(n)\rceil$
Consider any two vertices u, v such that u, v are reachable from each other in $G_{\lceil log(n)\rceil}$. Since $G_{\lceil log(n)\rceil}$ is the transitive closure of G, the pair of vertices for each edge in the path from u to v are reachable from another in G and hence u is reachable from v in G. Thus, $(u, v) \in G_{\lceil log(n)\rceil}.E$. Hence, any two vertices u, v that are reachable from one another have an edge between them and hence $D_{G_{\lceil log(n)\rceil}}(u, v) = 1$. Otherwise, if u, v are not reachable from one another, $D_{G_{\lceil log(n)\rceil}}(u, v) = \infty$. Thus, the base case is true.
Induction Step Assume that $D_{G_i} = ALL - PAIRS - SHORTEST - PATHS(A_{G_i})$.
Consider $ALL - PAIRS - SHORTEST - PATHS(A_{G_{i-1}}$, from the above lemma, $H_{G_{i-1}}) = G_i$. Thus, form the induction hypothesis, we can conclude that D_H in **line 6** is equal to $D_{H(G_{i-1})}$. We then compute $D_{G_{i-1}}$ from it using the algorithm from the problem above. Thus, the algorithm is correct for G_{i-1}. Hence, the induction is complete.

Time complexity Assume the total complexity is $T(G)$. The **if** clause takes $O(n^2)$ time. In the **else** clause, the **line 7** and **line 5** take $O(n^\omega)$ time as proved in the above problems. Thus, $T(G, i) = O(n^\omega) + T(H, i + 1)$. Since $T(G, log(n)) = n^2$, We can write $T(G, 0) \leq c(n^\omega) + T(H, 1) = 2 * c(n^\omega) + T(H_1, 2)....$
$=\leq c\lceil logn\rceil * (n^\omega) + n^2$. Assuming $\omega > 2$, the time complexity is $O(n^\omega * log(n))$.

1 Problem-4

1.1 (a)

For all $i \in 0, 1, 2, ..n-1$, define random variables $X_i = |\{x : x \in S \ and \ H(x) = i\}|$, that is the number of values in S with hash-value i. Then, the problem reduces to proving that $P(X > log_2(n)) \leq 1/n$ where $X = max(X_1, X_2, ..X_n)$. Observe that for arbitray positive numbers c, t:

$$P(X_i \geq c) = P(tX_i \geq tc) = P(2^{tX_i} \geq 2^{ct}) \leq E[2^{tX_i}]/2^{ct} \ \mathbf{-1} \text{ (by Markov's inequality)}$$

Now, we can write $X_i = \sum_{s \in S}\{s = i\}$ where $x = i$ is an indicator random variable which is 1, if $x = i$ and 0 otherwise. Observe that $E[s = i] = P(s = i) = 1/n$ for all i since the s are chosen uniformly, randomly.
Thus,

$$E[X_i] = \textstyle\sum_{s \in S} E[\{s = i\}] = (1/n) * n = 1 \text{ (by linearity of expectation)}$$

Also,

$$\begin{aligned} E[2^{tX_i}] = E[2^{t\sum_{s \in S}\{s=i\}}] = E[\Pi_{s \in S}2^{t\{s=i\}}] &= \Pi_{s \in S}E[2^{t\{s=i\}}] \text{ (since the } s_i\text{'s are independent)} \\ &= \Pi_{s \in S}(2^t * 1/n + 1 * (1 - 1/n)) \\ &= (2^t * 1/n + 1 * (1 - 1/n))^n \end{aligned}$$

From **1**,

$$P(X_i \geq c) \leq E[2^{tX_i}]/2^{ct} = (2^t * 1/n + 1 * (1 - 1/n))^n/2^{ct}$$

Plugging in $c = log_2(n)$ and $t = 3$ gives

$$P(X_i \geq log_2(n)) \leq (1 + 7/n)^n/n^3$$

We know that $(1 + 7/n)^n < e^7$ for all n. Therefore,

$$P(X_i \geq log_2(n)) \leq e^7/n^3$$

For sufficiently large n(here, for $n > e^7$),

$$P(X_i \geq log_2(n)) \leq e^7/n^3 \leq 1/n^2$$

Now using union bound we can claim that,

$$P(X > log_2(n)) \leq P(X \geq log_2(n)) = P(\textstyle\bigcup_{i=1}^n X_i \geq log_2(n)) \leq \sum_{i=1}^n P(X_i \geq log_2(n)) = 1/n$$

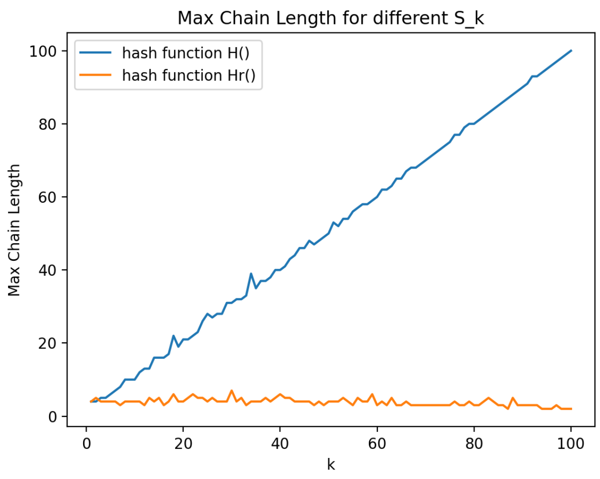
for sufficiently large n. Hence proved.

1.2 (b)

For any given r, observe that the function $H_r()$ is deterministic and has domain $\{1, 2, .., M\}$ and range $\{0, 1, .., n-1\}$. Thus, there are n possible values in the range. From the Pigeonhole principle, we can conclude that for atleast one of the values in the range, there will be atleast M/n elements in the domain that will be mapped to it. Consider any subset of size n from these M/n elements(Since $n <<< M$, we can assume that $M/n > n$). There will be $\binom{M/n}{n}$ such subsets. Also, all the elements in each of these subsets will be hashed to the same value using $H_r()$ giving maximum chain-length size as n. Thus, there are atleast $\binom{M/n}{n}$ subsets that give $\Theta(n)$ maximum chain length. Hence, proved.

1.3 (c)

Following is the plot for maximum chain length versus k. The prime number chosen is $p = 13469$.



The plots are reasonable. Since $\{0, n, 2n, 3n, ..., (k-1)n\} \in S_k$ always, the maximum chain length is atleast k for each S_k because the above k elements always get hashed to the same hash-value 0 with hash function H. This leads to an approximately linear plot for the hash function H.
Whereas, when H_r is chosen randomly from a hash family that is 2-universal(as discussed in lecture), the maximum chain length does not show any significant increase with increase in k. This is reasonable because now, the bound on the probability that a hash collision occurs does not depend upon the contents of the set S.

Python code for generating the plot.

```
import random
import matplotlib.pyplot as plt

M=10000
n=100
p=13469

def h(x) :
  return x%n

def hr(r,x) :
  return ((r*x)%p)%n

def max_chain_length(S_k,hash) :
  r = random.randint(1,p)
  chains = [[] for i in range(n)]

  for x in S_k :
    if(hash=='h') :
      chains[h(x)].append(x)
    else :
      chains[hr(r,x)].append(x)

  maxChainLength = 0

  for i in range(n) :
    maxChainLength = max(maxChainLength,len(chains[i]))

  return maxChainLength

A_h = []
A_hr = []
for k in range(1,n+1) :
  S_k = []
  for i in range(k) :
    S_k.append(i*n)
  for i in range(n-k) :
     S_k.append(random.randint(0,M))
     i=i+1
  A_h.append(max_chain_length(S_k,'h'))
  A_hr.append(max_chain_length(S_k,'hr'))
```

```python
k = [i for i in range(1,n+1)]
plt.plot(k, A_h,label='hash function H()')
plt.plot(k, A_hr,label='hash function Hr()')
plt.xlabel('k')
plt.ylabel('Max Chain Length')
plt.title('Max Chain Length for different S_k')
plt.legend()
plt.show()
```