CS345: Algorithms - II | Due: 17/08/19

Assignment #1

Name: **Ayush Hitesh Soneria**
Roll-No: **170192**
Email: ayushhs@iitk.ac.in OR ayushhs@cse.iitk.ac.in

Instructor: Prof. Nitin Saxena
TA: Ashish Dwivedi
TA's Email: ashish@cse.iitk.ac.in

Question 1

[3 points]

Trivial case: S can have zero or one element (which lies inside or on the unit square) in it. Now let's suppose there are two points in the unit square. We fix point A's position, now point B cannot be within 0.71 distance from A. Meaning what this indirectly could tell us is that point B will lie outside the circle (but inside the unit square) constructed by taking A as center and radius 0.71. Here the only constraint is that we have to fix position of A such that B can lie inside the unit square. (E.g. A can't be fixed as centre of square, B will have to lie outside square because the circle will enclose all of unit square)
See Figure 1, we can see when A is fixed at one of the corners (top right) of the square, only quarter circle lies in square, so B has maximum area in this case where it can lie in unit square.
Figure 2 shows two fixed points at two corners (top and bottom right). So addition of points possible only towards left side, outside the arcs.

Maximizing Strategy: In order to increase size of S, we need to place (fix) points such that any incoming point will have maximum area to place itself, and that point will place itself such that the area for next point if maximum, and so on until no space (area) is left for any additional points to be placed.
Figure 3 shows 4 fixed points at all 4 corners of the square. The circular arc have radii 0.707 (unit square diagonal-length/2) in this figure, hence an additional point can be placed at the centre of the square, if and only if the allowable distance between any 2 points was 0.707. But given that allowable distance is 0.71 (> 0.707), so figure 3 gives us an upper bound on size of S which is 4 by showing to us that no area remains (4 circular arcs of radii=0.71 span the entire square) for a fifth element even when maximizing strategy is applied.
Hence $0 <= |S| <= 4$

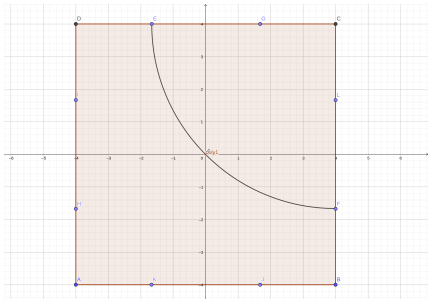


Figure 1: Top right corner point fixed

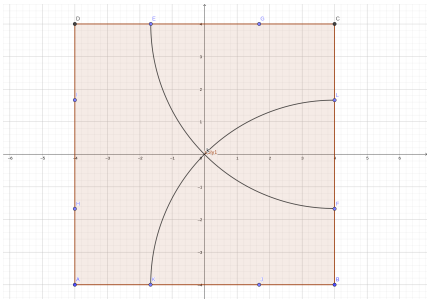


Figure 2: Top and bottom right corner points fixed

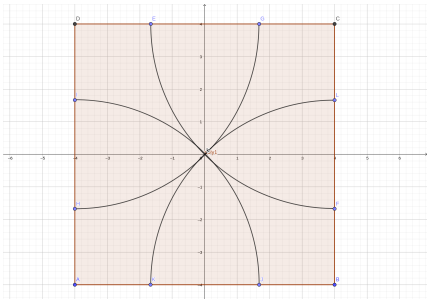
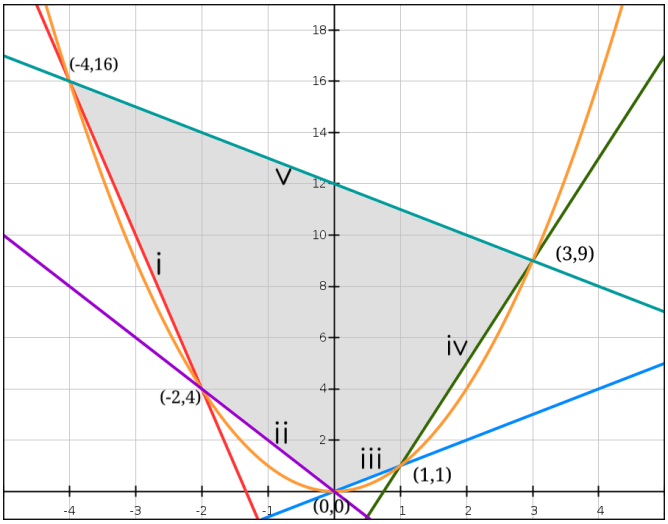


Figure 3: All corner points fixed, radii = 0.707, fifth point at centre

Question 2: Sort using Convex Hull [5 points]

We could take the following approach, find any function f, which plots the given n numbers onto the XY plane as (x,f(x)) and then when we convert the points into a convex hull and traverse it from an extreme point w.r.to the x-coordinates, we should obtain a sorted sequence. The criteria for a function f to satisfy our needs is that the graph of f(x) should be convex at all points, so that when we are connecting two consecutive points with edges we obtain an edge of the convex hull. If at any point the graph is concave, edge created at that point can be concave and hence that point might not end up as a vertex in our final convex hull and we won't get the correct sorted result. Let's use the Function f(x) = x^2 (convex function).
Let's take an example array A = [3,0,1,-4,-2] and sort this using convex hull.



The shaded area of the graph gives us the convex hull, we start traversing the hull from the left-most extreme point (-4,16). We move in anticlockwise direction along edge i, to reach (-2,4).
Doing this till we reach (-4,16) again, our path is $i \rightarrow ii \rightarrow iii \rightarrow iv \rightarrow v$.
The points we traverse are in following order $(-4, 16) \rightarrow (-2, 4) \rightarrow (0, 0) \rightarrow (1, 1) \rightarrow (3, 9)$. We can see here that the x-coordinates are in sorted order. Below is the pseudo-code for algorithm we are implementing.

PSEUDO-CODE

Data: $A \leftarrow$ array of **n** numbers
Result: $S \leftarrow$ n **sorted** numbers
for $i = 0$ *to* n **do**
  $B[i] \leftarrow A[i] * A[i]$; `/* B is the array of y-coordinates (Square of n given numbers) */`
end
CH $\leftarrow Convex\_Hull(A, B)$;
`/* CH is set of vertices representing resultant Convex Hull (Parabola) */`
$P \leftarrow$ Min x-coordinate point in CH; `/* smallest or leftmost (w.r.to x-axis) element */`
$j \leftarrow 0$;
while $(j < n)$ **do**
  `/* Traversing CH starting from P in anticlockwise direction (in increasing x-value direction because our CH will be a Parabola) */`
  $S[j] \leftarrow$ x(P); `/* S is the array storing sorted result of n given numbers */`
  $P \leftarrow$ P.next.anticlockwise;
  j++;
end

Algorithm 1: Sort using Convex_Hull in $O(nlogn)$

Question 2: Continued [5 points]

Proof of Correctness Analysis

The convex hull call we are making in line 4, has been done in class. In addition to that the only thing we are doing is traversing the convex hull in increasing order of x-values (anticlockwise direction), which should result in increasing order of numbers. Hence we get an output of sorted numbers.
Line 4, implementing the convex hull paradigm takes $O(nlogn)$ as discussed in class. All other lines take $O(n)$. The while loop from line 8-13 takes $O(n)$ because of the j-parameter incrementing from 0 to n-1. The line 6 also takes $O(n)$ as it has to look at all n points to find out the minimum. So the overall time complexity is;

$$T(n) = O(nlogn)$$

Question 3: Convex Hull [10 points]

The definition of convex hull of S is that it is a convex polygon of smallest area enclosing all the points of set S.
Let us first prove the second part of the question

Property 1: vertices of the convex hull of S are in S

Proof: Suppose there exists a vertex, P, of convex hull that is not in S. Let the edges from this vertex P be AP and PB, where A and B are in S. But is this the polygon with smallest area enclosing points in S? Suppose we delete point P, along with edges AP and PB from said convex hull and add the edge AB (join A and B) to it. Now the area of the convex hull is definitely smaller, because we have removed the area of triangle APB from it. So the polygon with vertex P (which is not in S) in it is not a convex hull. Hence for all vertices of polygon which are not is S (this polygon is not convex hull of S, because min area not achieved), they can all be removed along with respective edges and the area be made smaller and smaller until all the vertices of this polygon (now can be called convex hull of S) are necessarily in S. So in conclusion, in order to obtain a Convex Hull of S, the vertices need to be from set S. Hence proved.
Now taking help from above property and proving the first part of the question

Property 2: Convex Hull of S is unique

Proof:
Suppose we are given a convex hull of S with set of vertices V and set of edges E. Here let us try and form a different convex hull to disprove the uniqueness of convex hull of S. There are multiple cases from which a different convex hull can be formed.
Case 1: We can replace a vertex with a non-vertex point from S. We can observe here that the non-vertex point will be inside the convex hull, and when the we replace the vertex with this non-vertex point, the vertex point will go outside the convex hull. This contradicts the idea of the convex hull that it should enclose all points in S. Hence this case isn't plausible.
Case 2: If we just delete any vertex from V and its respective edges, and connect the neighbouring vertices with an edge, then that point will lie outside the convex hull. Again, Contradiction to idea of the CH(S).
Case 3: Add a non-vertex to set of vertices. The non-vertex would be inside the original convex hull and so joining it with any other vertex would yield to concavity or formation of a polygon inside another polygon. This result would not be called a convex hull.
So if the vertices of the convex hull cannot be altered then the edges certainly cannot be altered either. And so if the set of vertices or edges (which define the convex hull) can't be altered then we cannot find a different/alternative convex hull. Meaning there is a unique (only one) convex hull for a set S. Hence Proved.

Question 4: Non-Dominated Points [15 points]

n: # of points in input set
h: # of Non-Dominated points in output set

We have already seen the O(nlogn) and O(nh) algorithms in class. For $h = 1, O(nlogn) > O(nh) = O(n)$ but for $h >> n, O(nlogn) << O(nh)$. In order to obtain a more optimized algorithm we can take some ideas from the O(nh) algorithm and put them in place in the O(nlogn) algorithm.

Pseudo-code

```
   Data: S ← The Set of n points
   Result: Set of Non-Dominated (NP) Points
 1 Function NPpoints(S):                          /* algorithm as a recursive function */
 2 |  if (|S| <= 1) then                                          /* |S| = Length of S */
 3 |  |  return S;
 4 |  end
 5 |  Find x-median of all the points in S;   /* using median of medians approach, it takes O(n) */
 6 |  L ← Set of points in S whose x-coordinate is less than(/equal to) x-median;   /* Divide step */
 7 |  R ← Set of points in S whose x-coordinate is greater than x-median;           /* Divide step */
 8 |  Q ← point with max y-coordinate in R;              /* Definitely a Non-Dominated Point */
 9 |  for (point A in L) do
10 |  |  if (y(A) < y(Q)) then          /* check for lower y-coordinate points than Q in L */
11 |  |  |  L ← L.Delete(A);                    /* A is Dominated hence Pruning needed */
12 |  |  end
13 |  end
14 |  for (point B in R) do
15 |  |  if (x(B) < x(Q)) then          /* check for lower x-coordinate points than Q in R */
16 |  |  |  R ← R.Delete(B);                    /* B is Dominated hence Pruning needed */
17 |  |  end
18 |  end
19 |  NP_L ← NPpoints(L);                      /* Recursion on (remaining of) left half */
20 |  NP_R ← NPpoints(R);                     /* Recursion on (remaining of) right half */
21 |  return NP_L + NP_R;                                                  /* output */
22 End Function
```

Algorithm 2: NPpoints in $O(nlogh)$

Proof of Correctness

As discussed in class, In the O(nh) algorithm, the main idea was of

1. finding and outputting the max y-coordinate point, say P (which is a Non-Dominated point)
2. then deleting all the points whose x-coordinate is lesser then x(P) and also deleting point P
3. repeating the same process on remaining points till none remain

In the O(nlogn) algorithm, the main idea was

1. Divide the set by x-median into 2 sets, L and R
2. Recur on L and R to get NP_L and NP_R respectively
3. Find max y-coordinate point in NP_R, say Q (which is a Non-Dominated point)
4. Delete all points in NP_L whose y-coordinate is less than y(Q)
5. Return (remaining of) NP_L and NP_R.

Question 4: Continued [15 points]

As discussed in class, In O(nlogn) algorithm we see that the recursion tree has height logn and # of leaf nodes is n.
The steps 3,4 of O(nlogn) algorithm takes similar approach to O(nh) algorithm. Then we have realized that all points of NP_R obtained at any level of recursion are going to be included in the set of final Non-Dominated points. Hence pruning(deleting) done only on NP_L.
But what if the pruning step was done before calling recursion steps? What if we eliminated points using O(nh) algorithm idea and recurred on lesser # of points? Indirectly leading to lesser recursion levels, meaning height of recursion tree would be lesser.

Tweaking the O(nlogn) algorithm we get following new approach (which is implemented in the pseudo-code)

1. Divide the set by x-median into 2 sets, L and R
2. Find max y-coordinate point in R, say Q (which is a Non-Dominated point)
3. Delete all points in L whose y-coordinate is less than y(Q)
4. Also Delete all points in R whose x-coordinate is less than x(Q)
5. Recur on (remaining of) L and (remaining of) R to get NP_L and NP_R respectively
6. Return NP_L and NP_R

So as one can see this algorithm is only slightly modified from the O(nlogn) one and hence the proof of correctness follows from O(nlogn) algorithm.

Analysis of Pseudo-code

In this pseudo-code, lines 9-18 remove all the definitely Dominated (or unnecessary) points before recurring, with the help of Q, a definite Non-Dominated point. Lines 2-4, consider the corner case where L or R has only one element left and return that element as that element has yet not been dominated.
Making a simple observation, every leaf node of the recursion tree yields a Non-Dominated point. Because in order to be a leaf node, according to the algorithm, the point has go through the pruning process and survive(never pruned) until it is the only point left in that recursion path. It can survive only if it is a non-dominating point, if it is not a non-dominating point then it will come in contact with a point which dominates it and so be pruned(not survive) and hence never reach the leaf node.
So extrapolating the observation, we will have $'h'$ leaf nodes, as there are $'h'$ Non-Dominating points. The height of tree with $'h'$ leaf nodes is $'logh'$. Lines 2-18 take time complexity O(n) at each recursion level and there are $'logh'$ such recursion levels. Hence

$$T(n) = O(n) * logh = O(nlogh)$$

Question 5: Multi-set [17 points]

To solve this question the main idea we will use is that if we want to add two numbers p and q (p+q) then we can do this in the following manner, form polynomials x^p and x^q, multiply them together to obtain x^{p+q}. As one can see the power of x^{p+q} is the result we wanted (p+q).
So let us form two new polynomials both of degree 10n, S(x) and T(x) as follows,

$$S(x) = \sum_{i=0}^{n-1} x^{A_i} \qquad \text{where} \quad A_i = \ i^{th} \text{ element of set A}$$

$$T(x) = \sum_{j=0}^{n-1} x^{B_j} \qquad \text{where} \quad B_j = \ j^{th} \text{ element of set B}$$

We can see that the degree of S(x) and T(x) will be at most 10n. It takes O(n) time to form these two polynomials because size of A and B is both n.

$$U(x) = S(x) * T(x)$$

This Polynomial multiplication in done in O(nlogn) time (discussed in class, using Fast Fourier Transform by Cooley-Tukey '65). The number of non-zero terms in U(x) (equal to the size of the multi-set A+B) will be at most n^2. The form of U(x) will be:

$$U(x) = \sum_{k=0}^{2n-1} f_k x^k$$

The number of times (frequency) we get the term x^k is f_k (if $f_k > 1$ then it tells us that two or more different additions from set A and B gave same result)
Each x^k with non-zero coefficient is the result of multiplication of some x^{A_i} and some x^{B_j}. Which in turn tells us that k = $A_i + B_j$ for some i,j ($< n$)

Hence the multi-set A+B = The set of Powers of the terms (with non-zero coefficients) present in U(x).

The time complexity of the overall process is O(nlogn) because the polynomial multiplication takes O(nlogn) and everything else can be done in O(n).