# CS345 : Algorithms II
# Semester I, 2017-18, CSE, IIT Kanpur

**Special Assignment**

Deadline : 6:00 PM, 5 October 2017

## 1 Introduction

The aim of this assignment is to provide each student a glimpse of how an elegant algorithm for a problem is designed. Designing an algorithm involves making observations for a better insight, pondering over various questions for hours, making some failed attempts, learning from failures, taking different perspectives, sometimes starting from scratch with a fresh mind. The problem that is picked is smallest enclosing circle. In this assignment we shall take the student on the journey of an elegant algorithm for this problem. The tools required to desigh this algorithm are elementary geometry, elementary probability, and some creative skills. Each student may take one of the two routes: not attempting the problem at all and just implementing the final algorithm which will be posted on the website after 3 weeks or re-inventing the algorithm on his/her own and then implemeting it. No distinction will be made in grading based on which of these two routes a student takes. Read the last section of this assignment for the tasks that a student will have to complete for grading purpose. Time allocated for completing the assignment is 45 days. Plan your time accordngly.

The rest of this document is organzed as follows. In Section 2 we define the problem, its history and motivation for designing the linear time algorithm. Section 3 presents a short journey to design a brute force algorithm for this problem. For this we shall analyse some simple properties of the smallest enclosing circle, most of which were covered in the lectures using animation. Section 4 provides hints as well as a skeleton for the linear time algorithm. Section 5 states the details of the tasks each student has to accomplish to complete the assignment. The reader will be advised to take a pause at various places in this writeup. For a better understanding of the final algorithm, he/she should follow this advice scrupulously.

## 2 The smallest enclosing circle: problem, history, and motivation

The problem is defined as follows: Given a set of $n$ points in a plane, compute the circle of least radius that encloses all of them. This is one of the most fundamental problems in computational geometry with many applications as well. Moreover, the simplicity of the problem is so appealing that it inspires a reader to instantaneously ponder over it in search for its efficient algorithm.

This problem was first introduced in 1857 - long before the discipline of algorithm was formalized. But the researchers started working on it seriously only in 1960s. There is a simple $O(n^4)$ algorithm for this problem which will take days for any moderate size input even on the ultrafast computers present today. For many decades, there was no linear time algorithm for this problem. The first linear time algorithm was designed in 1983. However, it is too complex and uses many sophisticated algorithms and tools, e.g. linear programming in finite dimensions. As a result, every researcher will naturally have the following strong feeling: *A simple problem like this deserves a simple and elegant algorithm as well. So we must continue (re)search on this problem.* Indeed, one such elegant algorithm was discovered in 1991. Its salient features are as follows.

1. It requires elementary knowledge of geometry. Its intuition is based on a simple probability fact.
2. The algorithm is self contained. Once designed, it can be coded even by an average student of ESC101.
3. This algorithm takes expected $O(n)$ time. The algorithm is extremely efficient in reality though it suffers the drawback of any other randomized algorithm like randomized quick sort.

## 3 Towards designing an algorithm

Let $P$ denote the given set of $n$ points, and $C(P)$ denote the smallest circle that encloses every point of the set $P$. In order to design an algorithm to compute $C(P)$, it is natural to first explore its properties. We begin with the following two facts whose validity can be established using high school geometry only.

**Fact 1:** For any two points, the smallest enclosing circle is the circle with the two points lying on its diameter.

**Fact 2:** For any three points forming an acute angle triangle, the smallest enclosing circle is the circumcircle of the triangle formed by these points.

Let us now try to analyse the properties of $C(P)$ for any general $n$. We begin with a very basic lemma which has a simple proof by contradiction as well.

**Lemma 3.1** *$C(P)$ must pass through at least one point from $P$.*

**Proof:** If no point from $P$ lies on $C(P)$, then shrink the circle keeping its center fixed till one point from P appears on its circumference. This is surely a circle of smaller radius and different from $C(P)$.

The proof of the following lemma is based on similar reasoning as Lemma 3.1 and is left as an exercise.

**Lemma 3.2** *$C(P)$ must pass through at least two points from $P$.*

We now introduce a definition of a diametric line.

**Definition 3.1** *A chord of a circle is called a diametric line if it passes through the center as well. In other words, length of this chord will be the diameter of the circle.*

The proof of the following lemma is also based on similar reasoning as Lemma 3.1 and is left as an exercise.

**Lemma 3.3** *Let $S$ be the set of points lying on the circumference of $C(P)$. Let $QR$ be any diametric line of $C(P)$. Each of the two semicircles of $C(P)$ defined by $QR$ must have at least one point from $P$ on its boundary.*

Use the above lemmas to establish the following theorem.

**Theorem 3.1** *$C(P)$ either passes through two points from $P$ lying on its diameter or passes through three points that form an acute angle triangle.*

Hence there exists a pair or triplet of points from $P$ whose smallest circle is indeed the smallest enclosing circle for $P$. Ponder over this observation to design an algorithm for this problem. Do not worry about its running time at present

—— Pause for sufficient time before reading further ——

The algorithm is as follow. For each pair as well as triplet of points, find the corresponding circle passing through them and see if it encloses all the points as well. Among all such circles, report the circle of least radius. This algorithm runs in $O(n^4)$ time.

At first sight it appears that we should not hope for a faster algorithm for computing $C(P)$ because there does not seem to be any better algorithm to search for those 2 or 3 points that define the $C(P)$. But there might be an alternate way of computing $C(P)$ than just searching for those 2 or 3 points. We need to explore such an alternate way with fresh mind. For this purpose we might have to think over the problem from a different perspective.

## 4 Towards designing an efficient algorithm

Suppose we pick $n-1$ points out of $P$ and compute their smallest enclosing circle. How likely is it that it will be different from $C(P)$ ? Well, you will say that it depends upon what those $n-1$ points are. But, what if we pick them randomly uniformly ? Let us define this question formally.

**Open Question 4.1** *We pick a set $P'$ of $n-1$ points randomly uniformly from the set $P$ of $n$ points. What is the probability that $C(P') \neq C(P)$ ?*

Ponder over this question for sufficient time.

—— Pause for sufficient time before reading further ——

The answer to Question 4.1 is $\leq 3/n$. Make sincere attempts to provide the explanation for this answer. Notice that it requires elementary probability only. (By the way, how efficiently will you determine whether $C(P') \neq C(P)$ ? think over it). Use this observation to design an efficient algorithm.

—— Pause for sufficient time before reading further ——

Yes ! The algorithm is an incremental algorithm. It begins with a uniformly random permutation of $P$ Let $\langle p_1, p_2, \ldots, p_n \rangle$ denotes this permutation. Let $C(i)$ denote the smallest enclosing circle for first $i$ points in this permutation. The algorithm is described below.

**Algorithm 1:** Skeleton of the linear time algorithm for the smallest enclosing circle problem

```
1 C(2) ← smallest enclosing circle for p_1, p_2;
2 foreach i = 3 to n do
3 |   if p_i is enclosed by C(i − 1) then
4 |   |   C(i) ← C(i − 1);
5 |   else
6 |   |   .........
7 |   end
8 end
9 return C(n);
```

If the condition of the **If** statement is true, the time complexity of $i$th iteration is $O(1)$. But what if the condition fails ? How should we compute $C(i)$ in that case ? From scratch ? It seems all our efforts till now will be just a waste.

Have patience. You might have read this proverb sometime in the past : *each failure is a stepping stone to success*. This proverb is true, at least, in case of our current situation. The failure of the condition of the **If** statement indeed reveals some crucial information about $C(i)$. Try to find it out by working like a true researcher ...

—— Pause for sufficient time before reading further ——

The following lemma states the important inference when the condition of the **if** statement fails.

**Lemma 4.1** *If $P(i)$ does not lie inside $C(i-1)$, then $P(i)$ is one of the defining point for $C(i)$.*

So at least 50% or 33.3% of our work is already complete since we know one of points from the pair or triplet that defines $C(i)$. We just need to find the remaining one or two. How to do that ? Think over it and try to apply the insight we got till now. Work with passion, work with perseverance, work diligently to experience the joy of re-inventing an algorithm...

## 5 Details of the Assignment

You need to accomplish the following tasks as part of the assignment.

1. **[Not to be graded]** Make sincere attempts on your own to design the algorithm whose sketch is given above.

2. Implement the two algorithms: the brute force algorithm and the efficient randomized algorithm. Instead of wasting your time to write code for various procedures, you have to use a library LEDA (Library for Efficient Data Structures and Algorithms). This will be very helpful for your coding. A small tutorial on installing it and using it will soon be provided. Interestingly, if you use LEDA, the code for both the algorithms will be less than 100 lines.

3. Plot the running time of the algorithms for various values of $n$. Make inference about their running time carefully.

4. Find the distribution of the running time of the randomized algorithm empirically. How often does it deviate from $O(n)$ bound ?

5. **[Not to be graded]** Try to establish the $O(n)$ bound on the expected running time of the randomized algorithm formally.

The evaluation of this assignment will be based on the report you submit, the demo you present to the TA, and a short 10 minutes quiz.