



# **ESCUELA POLITÉCNICA NACIONAL**

## **FACULTAD DE CIENCIAS**

### **APLICACIONES DE ESTRUCTURAS DE DATOS A PROBLEMAS MATEMÁTICOS**

#### **IMPLEMENTACIÓN DEL PROBLEMA DE SELECCIONAR DE UN CONJUNTO DE PUNTOS DADO, EL PUNTO MÁS CERCANO A OTRO PUNTO DADO.**

**TRABAJO DE INTEGRACIÓN CURRICULAR PRESENTADO COMO  
REQUISITO PARA LA OBTENCIÓN DEL TÍTULO DE INGENIERO  
MATEMÁTICO**

**BRYAN ROBERTO TIPÁN GARZÓN**

[bryan.tipan@epn.edu.ec](mailto:bryan.tipan@epn.edu.ec)

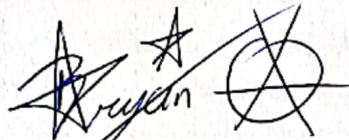
**DIRECTOR: MARÍA FERNANDA SALAZAR MONTENEGRO**

[fernanda.salazar@epn.edu.ec](mailto:fernanda.salazar@epn.edu.ec)

**DMQ, 13 DE SEPTIEMBRE DE 2022**

## **CERTIFICACIONES**

Yo, BRYAN ROBERTO TIPÁN GARZÓN, declaro que el trabajo de integración curricular aquí descrito es de mi autoría; que no ha sido previamente presentado para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.

A handwritten signature in blue ink, appearing to read 'Bryan Tipán', with a star symbol above the 'n' and a circled 'X' to the right.

---

Bryan Roberto Tipán Garzón

Certifico que el presente trabajo de integración curricular fue desarrollado por Bryan Roberto Tipán Garzón, bajo mi supervisión.

A handwritten signature in blue ink, appearing to read 'María Fernanda Salazar', written in a cursive style.

---

María Fernanda Salazar Montenegro  
**DIRECTOR**

## **DECLARACIÓN DE AUTORÍA**

A través de la presente declaración, afirmamos que el trabajo de integración curricular aquí descrito, así como el(los) producto(s) resultante(s) del mismo, es(son) público(s) y estará(n) a disposición de la comunidad a través del repositorio institucional de la Escuela Politécnica Nacional; sin embargo, la titularidad de los derechos patrimoniales nos corresponde a los autores que hemos contribuido en el desarrollo del presente trabajo; observando para el efecto las disposiciones establecidas por el órgano competente en propiedad intelectual, la normativa interna y demás normas.

Bryan Roberto Tipán Garzón

María Fernanda Salazar Montenegro

## RESUMEN

En el presente trabajo de integración curricular se propone resolver el problema del vecino más cercano, el cual consiste en encontrar de un conjunto de  $n$  puntos en el plano, aquel cuya distancia sea mínima con un punto de consulta  $q$  externo al conjunto. Este problema se resuelve mediante la implementación de dos algoritmos, el primero hace uso del paradigma de fuerza bruta, en el cual se realizan todas las comparaciones entre los puntos del conjunto y el punto  $q$ . El segundo algoritmo propuesto por Jon L. Bentley, organiza los puntos en una estructura llamada  $2d$ -Árbol que facilita la resolución del problema de manera eficiente.

Los algoritmos son implementados en el lenguaje de programación C++ y para ponerlos en ejecución, se construyeron las instancias simulando coordenadas de puntos en el plano.

Finalmente se ejecutaron distintas instancias poniendo a prueba los algoritmos y comparándolos en términos de eficiencia.

**Palabras clave:**  $2d$ -Árbol, algoritmo, eficiencia computacional, fuerza bruta, vecino más cercano.

## **ABSTRACT**

In the present work it is proposed to solve the nearest neighbor problem, which consists of finding from a set of  $n$  points in the plane, the one whose distance is minimum with a query point  $q$  external to the set. This problem is solved by implementing two algorithms. The first uses the brute force paradigm, in which all comparisons are made between the points of the set and the point  $q$ . The second algorithm proposed by Jon L. Bentley, organizes the points in a structure called  $2d$ -Tree that facilitates the resolution of the problem efficiently.

Algorithms are implemented in the programming language C++. To put them into execution, the instances were built simulating coordinates of points in the plane. Finally, different instances were executed, testing the algorithms and comparing them in terms of efficiency.

**Keywords:**  $2d$ -Tree, algorithm, computational efficiency, brute force, nearest neighbor.

---

# Índice general

---

<b>1. Descripción del componente desarrollado</b>	<b>1</b>
1.1. Objetivo general . . . . .	2
1.2. Objetivos específicos . . . . .	2
1.3. Alcance . . . . .	3
1.4. Marco teórico . . . . .	3
1.4.1. Algoritmos . . . . .	3
1.4.2. Notación Big $\mathcal{O}$ , Big $\Omega$ y Big $\Theta$ . . . . .	4
1.4.3. Complejidad de Algoritmos . . . . .	5
1.4.4. Algoritmo de fuerza bruta . . . . .	7
1.4.5. Problema del vecino más cercano . . . . .	7
1.4.6. Enfoques a la solución del NN . . . . .	8
<b>2. Metodología</b>	<b>12</b>
2.1. Introducción . . . . .	12
2.2. Algoritmo de fuerza Bruta . . . . .	12
2.2.1. Algoritmo 1: Solución por fuerza bruta . . . . .	13
2.2.2. Análisis de complejidad . . . . .	13
2.3. Algoritmo 2d-árbol . . . . .	13
2.3.1. 2d-árbol . . . . .	13
2.3.2. Algoritmo 2: Construcción del 2d-árbol . . . . .	14

2.3.3. Análisis de complejidad . . . . .	14
2.3.4. Algoritmo 3: Solución usando 2d-Árbol . . . . .	15
2.3.5. Análisis de complejidad . . . . .	15
2.4. Implementación . . . . .	16
<b>3. Resultados, conclusiones y recomendaciones</b>	<b>18</b>
3.1. Generación de instancias . . . . .	18
3.2. Resultados . . . . .	18
3.2.1. Instancia de 100 puntos aleatorios . . . . .	19
3.2.2. Instancia de 1000 puntos aleatorios . . . . .	20
3.2.3. Instancia de 10000 puntos aleatorios . . . . .	22
3.2.4. Instancia de 25000 puntos aleatorios . . . . .	24
3.2.5. Instancia de 50000 puntos aleatorios . . . . .	25
3.2.6. Instancia de 75000 puntos aleatorios . . . . .	26
3.2.7. Instancia de 100000 puntos aleatorios . . . . .	27
3.3. Casos particulares . . . . .	29
3.3.1. Conjunto de puntos $P$ todos equidistantes al punto de búsqueda $q$ . . . . .	29
3.4. Conclusiones y recomendaciones . . . . .	31
<b>Bibliografía</b>	<b>32</b>

---

## Índice de figuras

---

1.1. Problema del vecino más cercano (izquierda) y su solución (derecha). . . . .	1
1.2. Construcción de un 2d-árbol con un conjunto de ocho puntos	11
3.1. Conjunto de 100 puntos en el plano con el punto de consulta $q$	19
3.2. Solución al problema NN con 100 puntos . . . . .	20
3.3. Conjunto de 1000 puntos en el plano y el punto de consulta $q$	21
3.4. Solución al problema NN con un conjunto de 1000 puntos .	22
3.5. Problema NN con 10000 puntos . . . . .	23
3.6. Conjunto de 25000 puntos en el plano y el punto $q$ . . . . .	24
3.7. Solución al problema NN con un conjunto de 25000 puntos	25
3.8. Conjunto de 50000 puntos en el plano y el punto $q$ . . . . .	26
3.9. Solución al problema NN con un conjunto de 50000 puntos	26
3.10. Conjunto de 75000 puntos en el plano y el punto $q$ . . . . .	27
3.11. Solución al problema NN con un conjunto de 75000 puntos	27
3.12. Problema NN con 100000 puntos . . . . .	28
3.13. Problema NN en una circunferencia . . . . .	30



# Capítulo 1

---

## Descripción del componente desarrollado

---

El problema del vecino más cercano (NNP por sus siglas en inglés) consiste en que dados un punto de búsqueda  $q \in \mathbb{R}^2$ , y un conjunto  $P$  de  $n$  puntos en el plano, se desea encontrar el punto  $p$  contenido en  $P$  cuya distancia al punto  $q$  sea la menor.

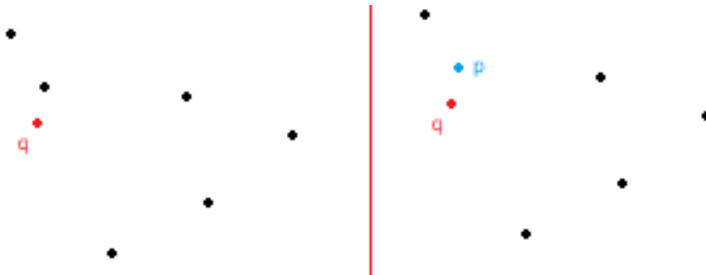


Figura 1.1: Problema del vecino más cercano (izquierda) y su solución (derecha).

La figura 1.1 muestra un ejemplo del problema del vecino más cercano en el espacio bidimensional y un conjunto de siete puntos.

El problema del vecino más cercano surge como una aproximación al problema del agente viajero TSP [10] y eventualmente ha escalado en importancia hasta entrar en campos como la inteligencia artificial o la detección de puntos atípicos (Rosenkrantz et al. 2010). Debido a la gran variedad de campos en los cuales el problema del vecino más cercano interviene directa o indirectamente, se han desarrollado varios algoritmos que permiten resolver el problema.

En este trabajo se propone resolver el problema del vecino más cercano en el espacio  $\mathbb{R}^2$  mediante la aplicación de dos algoritmos. El primero hace uso del paradigma del algoritmo de fuerza bruta, buscando la solución por comparación exhaustiva de todos los puntos; mientras que el segundo utiliza la estructura  $kd$ -árbol, la cual es un tipo particular de árbol binario, que divide el espacio de búsqueda en cada nivel del árbol permitiendo así encontrar la solución al problema en un número considerablemente menor de comparaciones.

Se presenta la descripción de ambos algoritmos, su implementación realizada en lenguaje C++ y pruebas computacionales comparando en cada instancia el número de búsquedas que se realizan hasta encontrar la solución correcta. Finalmente, se comprueba el mejor desempeño del segundo algoritmo mediante el análisis de complejidad.

## 1.1. Objetivo general

Resolver el problema del vecino más cercano en el espacio euclídeo mediante la implementación de dos algoritmos, uno de búsqueda lineal y otro usando un  $2d$ -árbol y comparar los resultados obtenidos para diferentes instancias.

## 1.2. Objetivos específicos

1. Implementar un algoritmo de búsqueda lineal para encontrar la distancia más corta entre un punto de interés  $q$  y los  $n$  puntos del conjunto  $P$ .
2. Implementar un algoritmo utilizando un  $2d$ -árbol y usarlo para encontrar la distancia más corta entre un punto de interés  $q$  y los  $n$  puntos del conjunto  $P$ .
3. Analizar el orden de complejidad de los algoritmos implementados y comparar sus resultados.

## 1.3. Alcance

La búsqueda por fuerza bruta es sencilla de implementar y, siempre que exista, encuentra una solución para el problema de estudio; sin embargo, su costo de ejecución es proporcional al número de soluciones candidatas. Por este motivo, se implementará un algoritmo inteligente que permita resolver el problema de una forma más eficiente.

Para construir tal algoritmo se considerará la estructura  $kd$ -árbol, un tipo particular de árbol binario de búsqueda diseñado para lidiar con datos multidimensionales. Dicha herramienta permite organizar los puntos en el espacio de forma que la búsqueda del vecino cercano sea más eficiente.

Finalmente se estudiará la eficiencia del algoritmo inteligente, es decir, se determinará el orden de complejidad del algoritmo, y se comparará con el algoritmo de fuerza bruta.

## 1.4. Marco teórico

### 1.4.1. Algoritmos

Un algoritmo es una secuencia finita de instrucciones exactas para realizar un cálculo o resolver un problema [14].

Los algoritmos se pueden expresar de muchas maneras diferentes. Se pueden construir algoritmos muy simples usando oraciones regulares en cualquier lenguaje humano, los algoritmos más complejos se pueden representar como diagramas mediante el uso de diagramas de flujo, los lenguajes de programación y el pseudocódigo se pueden usar para expresar algoritmos muy complejos [9].

El pseudocódigo es el paso intermedio entre describir los pasos de un procedimiento en lenguaje humano y especificar el procedimiento en el lenguaje de programación real [14], lo que facilita la comprensión del funcionamiento general de los algoritmos, no solo para los no programadores, sino también para los programadores que no están familiarizados con un lenguaje de programación específico [15].

En general, un algoritmo tiene las siguientes características[17]:

- Condiciones iniciales explícitas, completas y precisas.
- Una serie completa e ininterrumpida, pero no necesariamente lineal, de pasos a seguir para llegar al resultado deseado.
- Condiciones terminales (detención) explícitas, completas y precisas.
- Todo lo anterior expresado en términos simbólicos, de modo que el algoritmo pueda aplicarse a cualquier instancia específica de una clase general de problemas.

### 1.4.2. Notación Big $\mathcal{O}$ , Big $\Omega$ y Big $\Theta$

Cuando se realiza un cálculo, es deseable realizarlo lo más rápido posible, o por lo menos bastante rápido. Para entender lo que eso significa computacionalmente, es necesario introducir la notación Big  $\mathcal{O}$ .

Rubinstein-Salzedo, S. (2018) [16] definen la notación Big  $\mathcal{O}$  como:

**Definición 1.4.1.** Sean  $f(x)$  y  $g(x)$  dos funciones, se dice que  $f(x)$  es  $\mathcal{O}(g(x))$  si existen constantes  $C$  y  $k$  tales que:

$$|f(x)| \leq C|g(x)| \quad \forall x > k$$

La notación Big- $\mathcal{O}$  se usa ampliamente para estimar la cantidad de operaciones que usa un algoritmo para resolver un problema, permitiendo así el poder comparar dos algoritmos en términos de dicha cantidad[14].

**Definición 1.4.2.** Sean  $f(x)$  y  $g(x)$  dos funciones, se dice que  $f(x)$  es  $\Omega(g(x))$  si existen constantes  $C > 0$  y  $k$  tales que:

$$|f(x)| \geq C|g(x)| \quad \forall x > k$$

**Definición 1.4.3.** Sean  $f(x)$  y  $g(x)$  dos funciones, se dice que  $f(x)$  es  $\Theta(g(x))$  si  $f(x)$  es  $\mathcal{O}(g(x))$  y  $f(x)$  es  $\Omega(g(x))$ .

### 1.4.3. Complejidad de Algoritmos

La eficiencia del algoritmo, se refiere a la cantidad de recursos computacionales que este requiere; es decir, en cuánto tiempo se ejecuta y cuánta memoria usa.

La idea es encontrar los algoritmos más eficientes posibles; es decir, cuando dos algoritmos se encuentran con el mismo problema, uno será más eficiente que el otro si requiere menos recursos para resolverlo. Así, podemos ver que el concepto de eficiencia nos permite comparar diferentes algoritmos entre sí. [14]

La complejidad temporal de un algoritmo es una medida del tiempo que tarda una computadora en resolver un problema usando ese algoritmo [14]. La unidad de tiempo a la que se refiere esta medida no es una unidad de tiempo específica, ya que no existe una computadora estándar a la que se puedan referir todas las medidas. Por lo tanto,  $T(n)$  se define como el tiempo de ejecución del algoritmo para una instancia de tamaño  $n$ . [11]

Esta complejidad se puede expresar como el número de operaciones básicas realizadas por el algoritmo, entendidas como las operaciones realizadas por la computadora en un período de tiempo acotado por una constante. Por lo tanto, las operaciones aritméticas básicas, las comparaciones lógicas, la asignación de variables y el acceso a estructuras como vectores y matrices se consideran operaciones básicas [11].

Se debe notar que el comportamiento del algoritmo puede variar significativamente para diferentes casos. En realidad, para muchos algoritmos, el tiempo de ejecución depende de la entrada específica y no solo del tamaño de la entrada. Entonces, para un mismo algoritmo, generalmente se investigan tres casos:

1. Peor de los casos
2. Mejor de los casos
3. Caso promedio

A continuación se describe la complejidad del peor caso.

## Complejidad del peor caso

La complejidad del peor caso de un algoritmo, corresponde al mayor número de operaciones requeridas para resolver el problema dado usando ese algoritmo en una instancia de tamaño específico. El análisis del peor caso indica cuántas operaciones requiere el algoritmo para garantizar una solución [14].

### TABLA DE COMPLEJIDADES

Complejidad	Orden
$\mathcal{O}(1)$	Constante
$\mathcal{O}(n)$	Lineal
$\mathcal{O}(n^2)$	Cuadrada
$\mathcal{O}(n^c)$	Polinomial
$\mathcal{O}(\log(n))$	Logarítmica
$\mathcal{O}(n \log(n))$	Loglineal

Cuadro 1.1: Tabla de complejidades

$\mathcal{O}(1)$  representa la complejidad de una función que siempre es la misma sin importar el tamaño de la entrada.

$\mathcal{O}(n)$  representa la complejidad de una función que crece en proporción directa al tamaño de la entrada  $n$ .

$\mathcal{O}(n^2)$  representa una función cuya complejidad es directamente proporcional al cuadrado del tamaño de la entrada. Agregar iteraciones anidadas a través de la entrada aumentará la complejidad de esta pudiendo llegar a  $\mathcal{O}(n^3)$ ,  $\mathcal{O}(n^4)$ , etc.

$\mathcal{O}(\log n)$  representa una función cuya complejidad aumenta logarítmicamente a medida que aumenta el tamaño de la entrada. Esto hace que las funciones  $\mathcal{O}(\log n)$  trabajen bien con instancias grandes sin afectar el rendimiento.

Como se observa en el cuadro 1.2, a medida que aumenta la complejidad de la función, la cantidad de cómputo o el tiempo requerido para completar una tarea puede aumentar significativamente. Por lo tanto, se requiere que este aumento sea lo más bajo posible a medida que aumenta el tamaño de la instancia.

$n$	$\mathcal{O}(1)$	$\mathcal{O}(\log n)$	$\mathcal{O}(n)$	$\mathcal{O}(n \log n)$	$\mathcal{O}(n^2)$
1	1	1	1	1	1
2	1	1	2	2	4
4	1	2	4	8	16
8	1	3	8	24	64
16	1	4	16	64	256
1024	1	10	1024	10240	1048576

Cuadro 1.2: Tabla comparativa

#### 1.4.4. Algoritmo de fuerza bruta

El paradigma de un algoritmo se refiere al enfoque general basado en un concepto particular que se puede utilizar para construir algoritmos que permitan resolver una variedad de problemas[14].

Un algoritmo de fuerza bruta usa un paradigma en el cual se prueban todas las posibilidades, sin aprovechar ninguna estructura especial, hasta encontrar una solución satisfactoria al problema[16].

#### 1.4.5. Problema del vecino más cercano

El problema del vecino más cercano (NN) es también conocido en la literatura como el problema de la oficina de correos[18]. Para formular el problema del vecino más cercano, se requiere definir los siguientes conceptos [19]:

**Definición 1.4.4** (Distancia euclidiana). Dados dos puntos  $p$  y  $q$  en  $\mathbb{R}^n$ , la distancia euclidiana  $d_{pq}$  entre  $p$  y  $q$  está dada por:

$$d_{pq} = \|p - q\|$$

donde

$$\|x\| = \left( \sum_{i=1}^n |x_i|^s \right)^{1/s}$$

**Definición 1.4.5** (Distancia entre un punto y una recta). Dada una recta  $H$  de ecuación  $ax + by + c = 0$ , donde  $a$ ,  $b$  y  $c$  son coeficientes reales tales

que  $a \neq 0$  y  $b \neq 0$ , la distancia entre  $H$  a un punto  $p = (p_x, p_y)$  está dada por:

$$\text{dist}(H, (p_x, p_y)) = \frac{|ap_x + bp_y + c|}{\sqrt{a^2 + b^2}}$$

Así, el problema NN se define formalmente como:

**Definición 1.4.6** (Problema NN). Dado un conjunto finito de puntos  $P \subset \mathbb{R}^n$  de datos y un punto de consulta  $q \in \mathbb{R}^n$ , encontrar el punto  $p \in P$  más cercano al punto  $q$ .

En la literatura, se evidencia que este problema es de gran importancia para varias áreas de la informática. Por ejemplo, las descritas en [5][18][19][10]:

- **Reconocimiento de voz** - Al encontrar patrones de voz similares en una base de datos de audios.
- **Genética** - Al encontrar secuencias de ADN o proteínas similares en una muestra genética.
- **Detección de copia** - Al encontrar oraciones similares en textos dentro de una base de datos de documentos.
- **Minado de datos** - Al encontrar coincidencias aproximadas en series temporales (por ejemplo temperaturas a lo largo de un año, década, etc.)
- **Solución aproximada al problema del agente viajero (TSP)** - La idea principal de esta aproximación consiste en siempre visitar la ciudad más cercana.

#### 1.4.6. Enfoques a la solución del NN

Existen algoritmos eficientes conocidos para resolver el problema del vecino más cercano en un espacio dimensional relativamente pequeño. Sin embargo, a medida que aumenta el tamaño, la complejidad de resolver el problema del vecino más cercano, tanto en términos de tiempo como de recursos informáticos, parece aumentar drásticamente.[2].



Dado que el proceso para encontrar el vecino más cercano de un punto dado demanda tiempo además de recursos computacionales los cuales aumentan cuando  $n$  o la dimensionalidad de los datos crece, se han desarrollado una serie de estructuras de datos y algoritmos de búsqueda para encontrar una solución que minimice dichas demandas[3],[4],[5].

### **Búsqueda lineal**

Una forma intuitiva y sencilla para resolver el problema NN es calcular las distancias entre cada punto del conjunto y el punto de consulta  $q$  y luego buscar el punto  $p$  con la distancia mínima [3]. La cantidad de trabajo involucrada en este método es obviamente proporcional a  $n$ , el tamaño del conjunto. Si solo se plantean unas pocas consultas sobre un conjunto de datos dado, este método es probablemente el mejor.

### **LHS**

El hashing sensible a la localidad (LSH, por sus siglas en inglés) [1] se basa en la idea de que si dos puntos son similares, después de ser evaluados por alguna función, seguirán compartiendo la misma noción de similitud. Una función de hashing sensible a la localidad  $H(x)$  garantiza, con probabilidad constante, que elementos similares producirán el mismo hash. El algoritmo LSH utiliza tales funciones hash para agrupar datos similares. Para lograrlo, primero genera un código hash para cada punto del conjunto de datos. Luego, los puntos se agrupan por su valor hash. Cuando se necesita consultar un punto para encontrar sus vecinos más cercanos, el algoritmo convierte el punto de consulta en un hash utilizando la misma función hash y luego se buscan puntos similares en los grupos.[7]

El LSH es una técnica que permite realizar consultas sobre un conjunto de datos en tiempo sublineal, aunque el método óptimo de explotar esta técnica todavía constituye un problema abierto [6].

### **kd-Árbol**

El  $kd$ -árbol [4] es una estructura de datos desarrollada por Jon Bentley en 1975. El árbol- $kd$  y sus variantes están entre las estructuras de datos más populares utilizadas para buscar en espacios multidimensionales.

El  $kd$ -árbol crea un conjunto con  $n$  puntos en un espacio  $k$ -dimensional.

En esta estructura, en cada paso el espacio de existencia se divide atendiendo a las dimensiones de los puntos. Esta división se continúa recursivamente hasta que en cada zona solo queda un punto.

Friedman, Bentley, y Finkel prueban en [8] que el tiempo de búsqueda esperado para los  $k$  vecinos más cercanos de un punto de consulta  $q$  en el espacio  $k$ -dimensional es proporcional a  $\log(n)$ , donde  $n$  es el número de puntos del conjunto  $P$ .

### **Diagramas de Voronoi**

Shamos y Hoey [19] proponen un enfoque utilizando diagramas de Voronoi. Un diagrama de Voronoi es un tipo especial de descomposición de un espacio métrico determinado por las distancias a un conjunto discreto de puntos en el espacio [13]. Sean  $p_1, \dots, p_n$ , un número finito de puntos distintos en el espacio cartesiano bidimensional  $\mathbb{R}^2$ ,  $q$  un punto arbitrario en  $\mathbb{R}^2$ , y  $d(p, p_i)$  la distancia euclidiana entre la ubicación  $q$  y el punto  $p_i$  en  $P$ . Se define la región,  $V(p_i)$ , por el conjunto de posiciones que satisfacen la condición:

$$V(p_i) = \{p | d(q, p_i) \leq d(q, p_j), j \neq i, j = 1, \dots, n\}$$

Dado que la distancia desde la ubicación  $q$  al punto  $p_i$  es menor o igual que la distancia a cualquier otro punto en  $P$ , significa que el punto  $p_i$  es el punto más cercano en  $P$  desde la ubicación  $q$  [12].

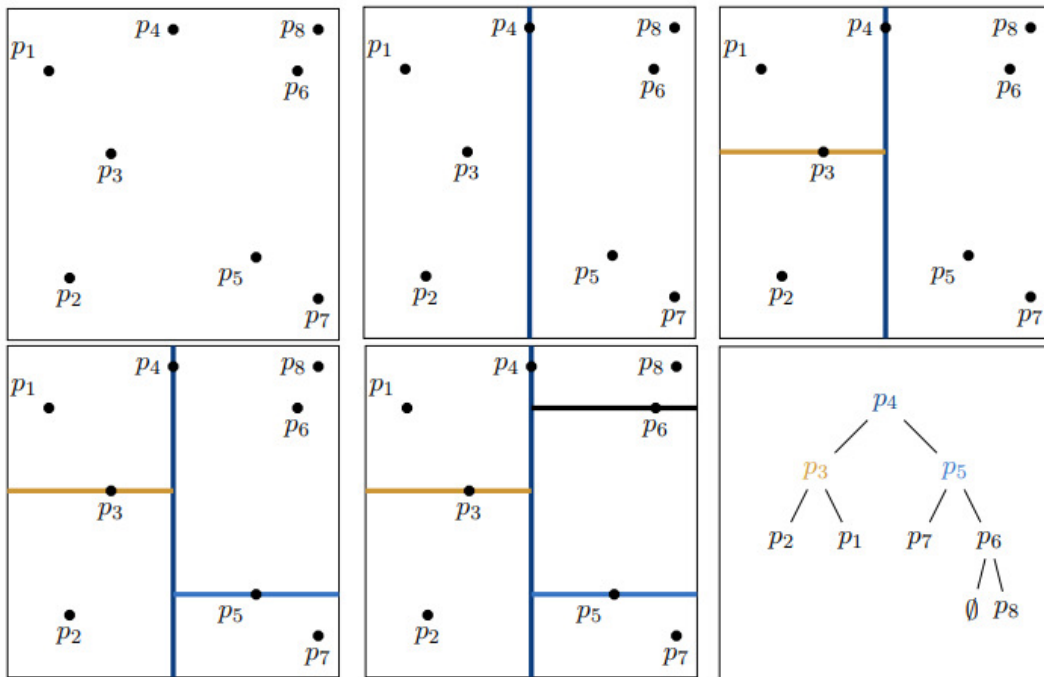


Figura 1.2: Construcción de un 2d-árbol con un conjunto de ocho puntos

# Capítulo 2

---

## Metodología

---

### 2.1. Introducción

En el presente trabajo se aborda el problema del vecino más cercano en el espacio  $\mathbb{R}^2$  utilizando la distancia euclidiana. Se resuelve el problema para varias instancias utilizando dos algoritmos, uno de fuerza bruta y otro utilizando un *kd*-árbol ambos implementados en C++. En cada uno de los casos se presenta una breve idea del funcionamiento de cada algoritmo, posteriormente se presenta su pseudocódigo y por último se realiza el análisis de complejidad.

### 2.2. Algoritmo de fuerza Bruta

El algoritmo de fuerza bruta presentado a continuación puede encontrar el punto  $p \in P$  más cercano al punto de búsqueda  $q$  calculando las distancias entre todos los  $n$  puntos del conjunto  $P$  con el punto  $q$  y determinando la distancia más pequeña.

## 2.2.1. Algoritmo 1: Solución por fuerza bruta

---

**Algoritmo 1:** Fuerza bruta para el problema NN

---

**Entrada:** Conjunto  $P = \{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$ , punto

$$q = (x_q, y_q)$$

**Salida** : Punto  $p = (x_p, y_p) \in P$  cuya distancia a  $q$  es la menor

```
1  $min = \infty$ 
2  $p = (x_1, y_1)$ 
3 for  $i = 1$  to  $n$  do
4   if  $(x_q - x_i)^2 + (y_q - y_i)^2 \leq min^2$  then
5      $min := \sqrt{(x_q - x_i)^2 + (y_q - y_i)^2}$ 
6      $p := (x_i, y_i)$ 
```

---

## 2.2.2. Análisis de complejidad

Para estimar el número de operaciones utilizadas por el Algoritmo 1, primero se nota que hay  $n$  puntos  $(x_i, y_i)$  con  $i = 1, 2, \dots, n$ , que recorrer. Para cada punto  $i$  se calcula  $(x_q - x_i)^2 + (y_q - y_i)^2$ , se compara con el valor actual de  $min$  y, si es menor que  $min$ , el valor actual de  $min$  es reemplazado por este nuevo valor.

De ello se deduce que este algoritmo utiliza un orden de complejidad  $\mathcal{O}(n)$  para llegar a una respuesta.

## 2.3. Algoritmo 2d-árbol

### 2.3.1. 2d-árbol

Dado un conjunto  $P = \{p_1, \dots, p_n\}$ , donde  $p_j = (p_{jx}, p_{jy}) \in \mathbb{R}^2$  para  $j = 1, \dots, n$ , el 2d-Árbol de  $P$  se construye de manera recursiva. En el primer paso, los puntos de  $P$  se ordenan de acuerdo a su primera coordenada  $x$ . Segundo, se encuentra la mediana  $m$  de los valores ordenados en el primer paso. Es decir, se calcula el valor de  $m$ , de modo que al menos el 50% de los puntos tengan su primera coordenada mayor a  $m$ , y al menos el 50% de los puntos tengan su primera coordenada menor o igual a  $m$ . El punto se almacena como el nodo raíz y el conjunto  $P$  se divide en  $P_{izq}$

y  $P_{der}$ , donde  $P_{izq}$  contiene solo los puntos con su primera coordenada menor o igual que  $m$ , y  $P_{der}$  contiene solo los puntos con su primera coordenada mayor que  $m$ . Luego, el proceso se repite recursivamente tanto en  $P_{izq}$  como en  $P_{der}$ , reemplazando la segunda coordenada de los puntos de  $P$ . Cuando la mediana de las segundas coordenadas es tomada, en el siguiente paso de recursividad, la mediana será calculada con las primeras coordenadas.

Cuando el conjunto de puntos en un nodo tiene tamaño 1, la recursividad se detiene.

### 2.3.2. Algoritmo 2: Construcción del 2d-árbol

---

**Algoritmo 2:** Construcción 2d-Árbol.

---

**Entrada:** Conjunto  $P = \{(p_{1x}, p_{1y}), (p_{2x}, p_{2y}), \dots, (p_{nx}, p_{ny})\}$

**Salida** : nodo raíz

```

1  $\delta = 0$  Profundidad actual del árbol
2  $d \leftarrow$  Primera coordenada de los puntos en  $P$ ,  $x$ 
3 if  $|P| \leq 1$  then
4   | Return Nodo conteniendo el único punto de  $P$ 
5 else
6   |  $m \leftarrow$  Mediana de  $P$  considerando sólo la coordenada  $d$ 
7   |  $H_L \leftarrow$  Recta que divide al plano según  $d$ 
8   |  $P_{der} = P \cap \{p_j \in P \mid p_{jd} > m\}$ 
9   |  $P_{izq} = P \cap \{p_j \in P \mid p_{jd} \leq m\}$ 
10  |  $\delta \leftarrow \delta + 1$ 
11  |  $d \leftarrow$  Cambiar de coordenada en los puntos de  $P$ ,  $y$ 
12  |  $nodo_{der} \leftarrow$  Construcción 2d-árbol con  $P_{der}$ 
13  |  $nodo_{izq} \leftarrow$  Construcción 2d-árbol con  $P_{izq}$ 
14  | Return Nodo conteniendo  $m$  con  $nodo_{der}$  como nodo hijo
    |   derecho y  $nodo_{izq}$  como nodo hijo izquierdo

```

---

### 2.3.3. Análisis de complejidad

En el algoritmo 2, en cada paso de recurrencia, la parte que consume más tiempo es encontrar la mediana. Esto se puede hacer en  $\mathcal{O}(n)$  [17]. Por lo tanto, el tiempo de construcción  $T(n)$  de un 2d-árbol satisface

la siguiente igualdad:

$$T(n) = \begin{cases} \mathcal{O}(1) & \text{si } n = 1 \\ \mathcal{O}(n) + 2 \cdot T(\lceil n/2 \rceil) & \text{si } n > 1 \end{cases}$$

La cual se resuelve en  $T(n) = \mathcal{O}(n \log(n))$ [17]

### 2.3.4. Algoritmo 3: Solución usando 2d-Árbol

El siguiente algoritmo trabaja de manera recursiva. Empieza en la raíz y luego compara los nodos del 2d-árbol encontrando las regiones del plano en las cuales buscar primero y, al mismo tiempo, omite las regiones del plano donde la posibilidad de obtener un punto  $p'$  más cercano que el punto  $p$  encontrado hasta ahora sea menor.

Inicialmente el valor de la distancia mínima es infinita y el punto  $p$  es nulo. Se empieza en el nodo raíz  $r$ , con la dimensión de corte siendo  $x$  entonces:

- Al llegar a un nodo nulo regresar el valor de  $p$ .
- Si entre la región del plano que contiene a la raíz actual y el punto de consulta  $q$  existe una distancia mayor que la distancia entre  $p$  actual y  $q$  implica que, esta región no tiene posibilidad de tener un punto más cercano que la distancia mínima actual, entonces no realizar comparaciones en ese subárbol.
- Si la raíz actual está más cerca de  $q$  que la distancia mínima, se extrae el punto en la raíz, se lo guarda como el punto  $p$  y se actualiza la distancia mínima.
- Si el valor de la dimensión de corte de  $q$  es más pequeño que el de la raíz, entonces comparar primero a la izquierda, segundo a la derecha.

### 2.3.5. Análisis de complejidad

Se consideran los siguientes casos en el análisis de complejidad para el Algoritmo 3

---

**Algoritmo 3:** SoluciónNN( $q, r$ )

---

**Entrada:** Punto  $q = (q_x, q_y)$ , nodo raíz del 2d-árbol:  $r$

**Salida :** Punto  $p = (p_x, p_y)$  cuya distancia a  $q$  es la menor

```
1  $min = \infty$ 
2  $p \leftarrow null$ 
3 if  $r == null$  then
4   | return  $p$ 
5 else
6   |  $p \leftarrow$  Punto contenido en el nodo  $r$ 
7   |  $min = ||q - p||$ 
8   | if  $r$  no tiene subárboles then
9     | return  $p$ 
10  | else
11  | SoluciónNN( $q, nodo_{der}$  hijo de  $r$ )
12  | if  $dist(H_{\mathcal{L}}, q) < min$  then
13  |   | SoluciónNN( $q, nodo_{der}$  hijo de  $r$ )
14  |   | else if  $||p - q|| > dist(H_{\mathcal{L}}, q)$  then
15  |   |   | SoluciónNN( $q, nodo_{izq}$  hijo de  $r$ )
16  |   | return  $p$ 
```

---

- Si el 2d-árbol está balanceado el Algoritmo 3 visita por lo menos  $\mathcal{O}(\log(n))$  nodos del árbol descendiendo por todos los niveles del árbol hasta presentar la respuesta correcta.
- Si el 2d-árbol no está balanceado, en el peor de los casos el Algoritmo 3 visitará todos los nodos del árbol hasta llegar a la respuesta correcta, es decir su complejidad en este caso es  $\mathcal{O}(n)$

## 2.4. Implementación

Para el correcto funcionamiento de la implementación del Algoritmo 1 y el Algoritmo 3, estos hacen uso de archivos tipo header .h los cuales son cargados desde el fichero fuente main.cpp. El fichero funciones\_1.h, usado en ambas implementaciones sirve para lectura de los archivos .txt que contienen diferentes instancias, los archivos que contienen dichas instancias son: base100.txt, base1000.txt, base10000.txt, base25000.txt, base50000.txt, base100000.txt, base\_circulo.txt.

Ambas implementaciones contienen un archivo .h llamado funcio-



nes\_operacionales.h para el caso del Algoritmo 1 dentro de este archivo se encuentran las funciones implementadas para:

- Imprimir los datos leídos desde el archivo .txt correspondiente
- Ingresar las coordenadas del punto  $q$ .
- Calcular la distancia entre el punto  $q$  y cada punto del conjunto  $P$ .
- Determinar el índice del punto  $p$  cuya distancia a  $q$  es mínima.
- Presentar los resultados en pantalla.

El archivo funciones\_operacionales.h en el Algoritmo 3 contiene las funciones implementadas para:

- Imprimir los datos leídos.
- Ordenar los datos.
- Imprimir los datos ordenados.
- Insertar los datos en los nodos del árbol.
- Imprimir el árbol.
- Ingresar las coordenadas del punto de consulta  $q$  y calcular la distancia mínima.
- Imprimir los resultados.

# Capítulo 3

---

## Resultados, conclusiones y recomendaciones

---

### 3.1. Generación de instancias

En el presente trabajo se realizan las pruebas computacionales con instancias generadas aleatoriamente en Excel usando la función ALEATORIO.ENTRE( ) para obtener números aleatorios entre -10000 y 10000 obteniendo así instancias de 100, 1000, 10000, 100000 datos en  $\mathbb{R}^2$  y de igual manera se consideran casos especiales en la distribución de los datos.

En cada una de las instancias se considera como punto de consulta  $q$  el origen de coordenadas, es decir se tiene que  $q = (0, 0)$

### 3.2. Resultados

Todos los resultados computacionales fueron obtenidos utilizando un computador con procesador Intel Core i5-2450M CPU 2.50GHz con 4 GB de memoria RAM bajo el sistema operativo Windows 10 Pro 64-bit. Los algoritmos fueron implementados usando el entorno de desarrollo integrado de código abierto Code::Blocks versión 20.03.

### 3.2.1. Instancia de 100 puntos aleatorios

En la figura 3.1 se muestra la distribución de los 100 puntos aleatorios, así como al punto de consulta  $q$ . El punto  $p$  más cercano a  $q$  es el punto  $(-141, -35)$  como se aprecia en la figura 3.2 siendo la distancia entre estos dos de 145.279. La distancia mínima y el punto  $p$  coinciden usando ambos algoritmos con la principal diferencia que el Algoritmo 1 realiza 100 comparaciones mientras que usando el Algoritmo 3 se realizan solo 19 comparaciones.

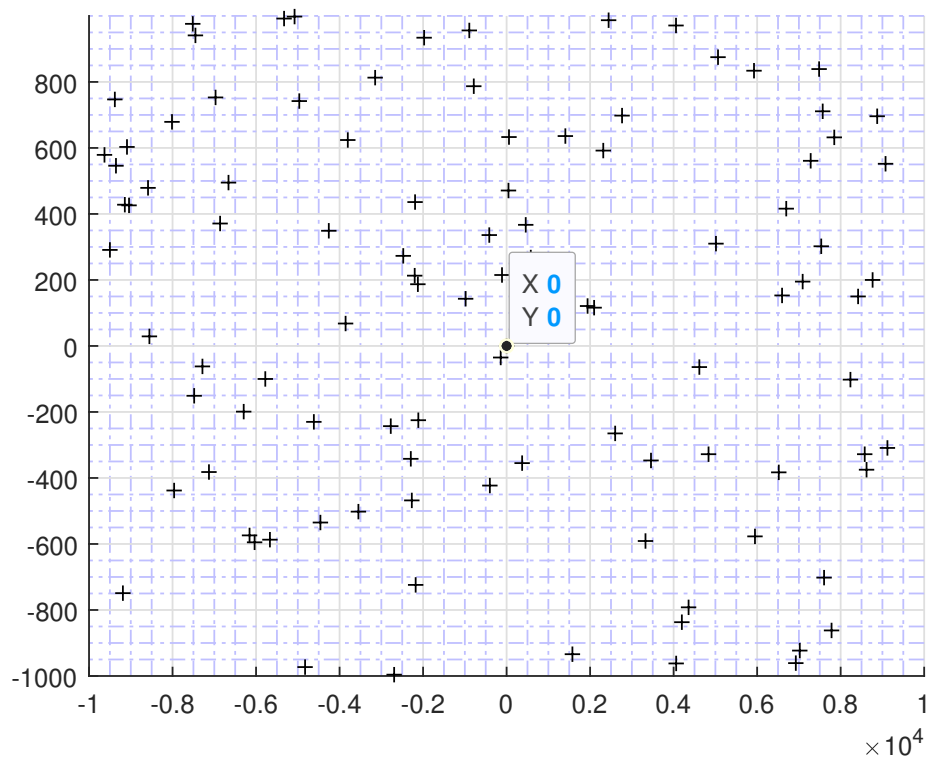


Figura 3.1: Conjunto de 100 puntos en el plano con el punto de consulta  $q$

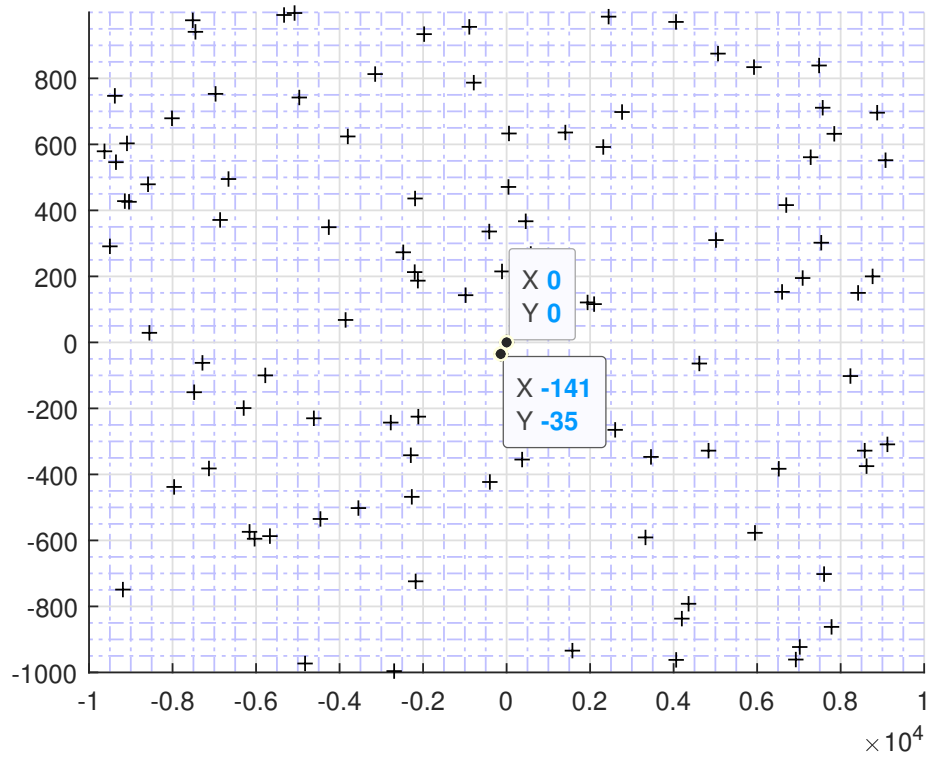


Figura 3.2: Solución al problema NN con 100 puntos

### 3.2.2. Instancia de 1000 puntos aleatorios

En la figura 3.3 se muestra la distribución de 1000 puntos aleatorios, así como al punto de consulta  $q$ .

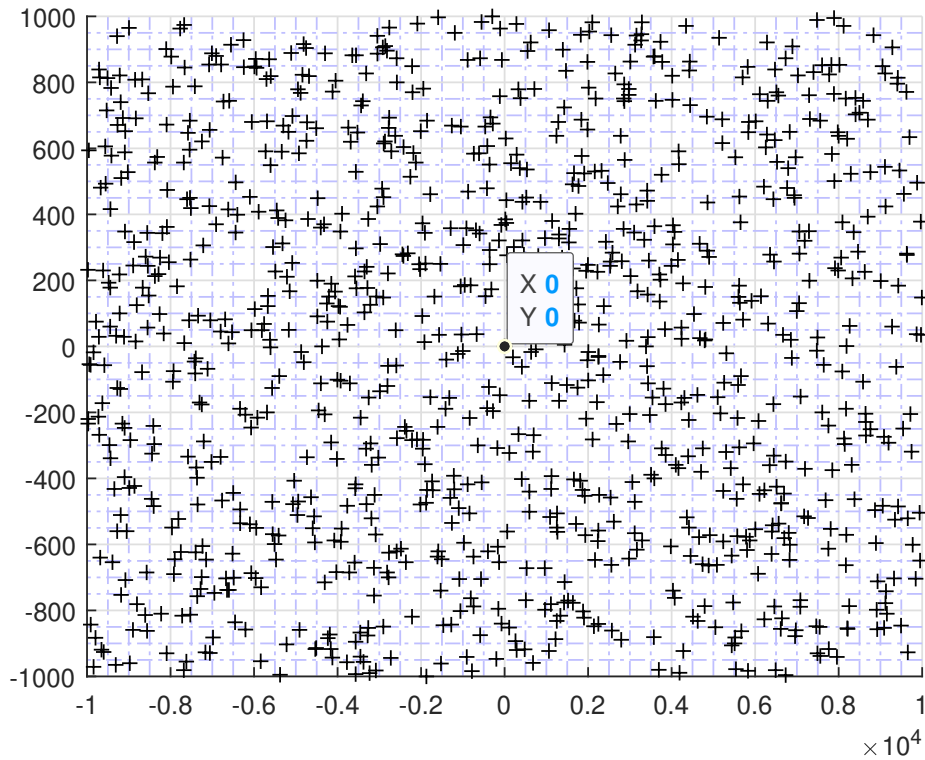


Figura 3.3: Conjunto de 1000 puntos en el plano y el punto de consulta  $q$

El punto  $p$  más cercano a  $q$  es el punto  $(-115, -148)$  como se aprecia en la figura 3.4 siendo la distancia entre estos dos de 187.427. Tanto la distancia mínima como el punto  $p$  reportados coinciden usando los dos algoritmos siendo el número de búsquedas realizadas por el Algoritmo 1 un total de 1000 mientras que con el Algoritmo 3 un total de 25 búsquedas.

Gráficamente se observa que, aparentemente, existen puntos más próximos a  $q$  que el punto  $p$  obtenido con los algoritmos sin embargo esta es una percepción visual originada por la escala presentada en el gráfico.

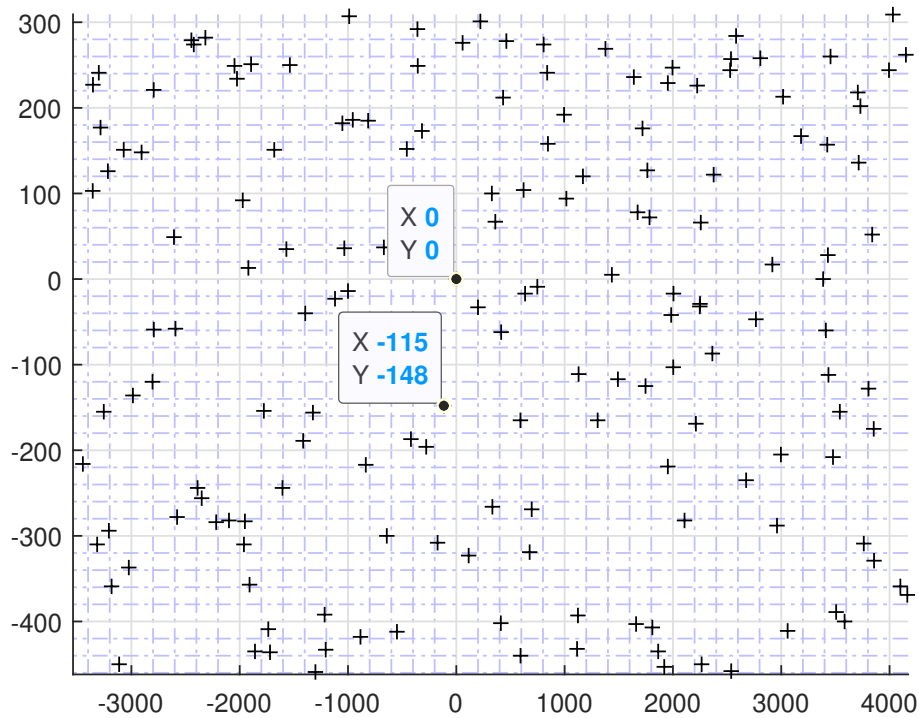
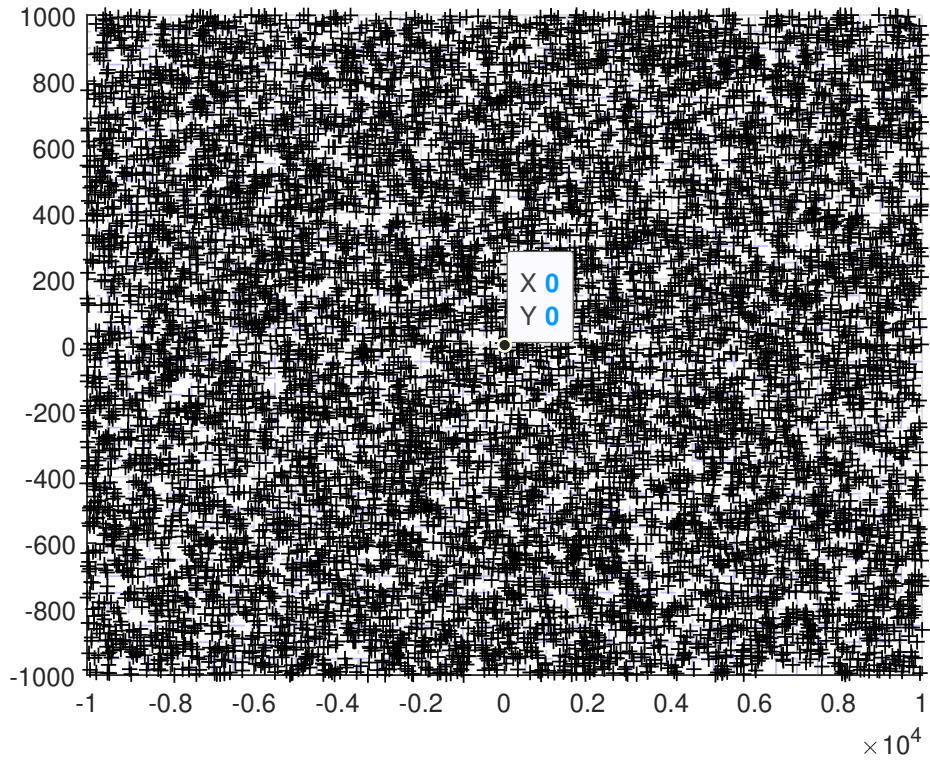


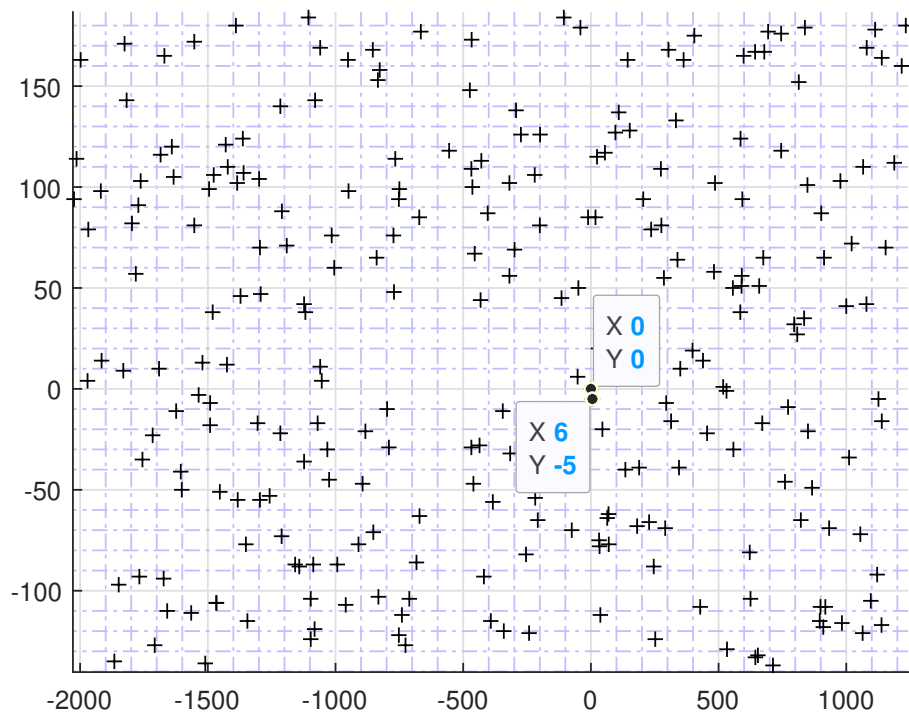
Figura 3.4: Solución al problema NN con un conjunto de 1000 puntos

### 3.2.3. Instancia de 10000 puntos aleatorios

En la figura 3.5a se muestra la distribución de 10000 puntos aleatorios, así como al punto de consulta  $q$ . El punto  $p$  más cercano a  $q$  es el punto  $(6, -5)$  como se aprecia en la figura 3.5b siendo la distancia entre estos dos de 7.8102. Tanto la distancia mínima como el punto  $p$  reportados coinciden usando los dos algoritmos siendo el número de búsquedas realizadas por el Algoritmo 1 un total de 10000 mientras que con el Algoritmo 3 un total de 27 búsquedas.



(a) Conjunto de 10000 puntos



(b) Solución al problema NN con un conjunto de 1000 puntos

Figura 3.5: Problema NN con 10000 puntos

### 3.2.4. Instancia de 25000 puntos aleatorios

En la figura 3.6 se muestra la distribución de 25000 puntos aleatorios, así como al punto de consulta  $q$ . El punto  $p$  más cercano a  $q$  es el punto  $(-16, 22)$  como se aprecia en la figura 3.7 siendo la distancia entre estos dos de 27.2029. Tanto la distancia mínima como el punto  $p$  reportados coinciden usando los dos algoritmos siendo el número de búsquedas realizadas por el Algoritmo 1 un total de 25000 mientras que con el Algoritmo 3 un total de 30 búsquedas.

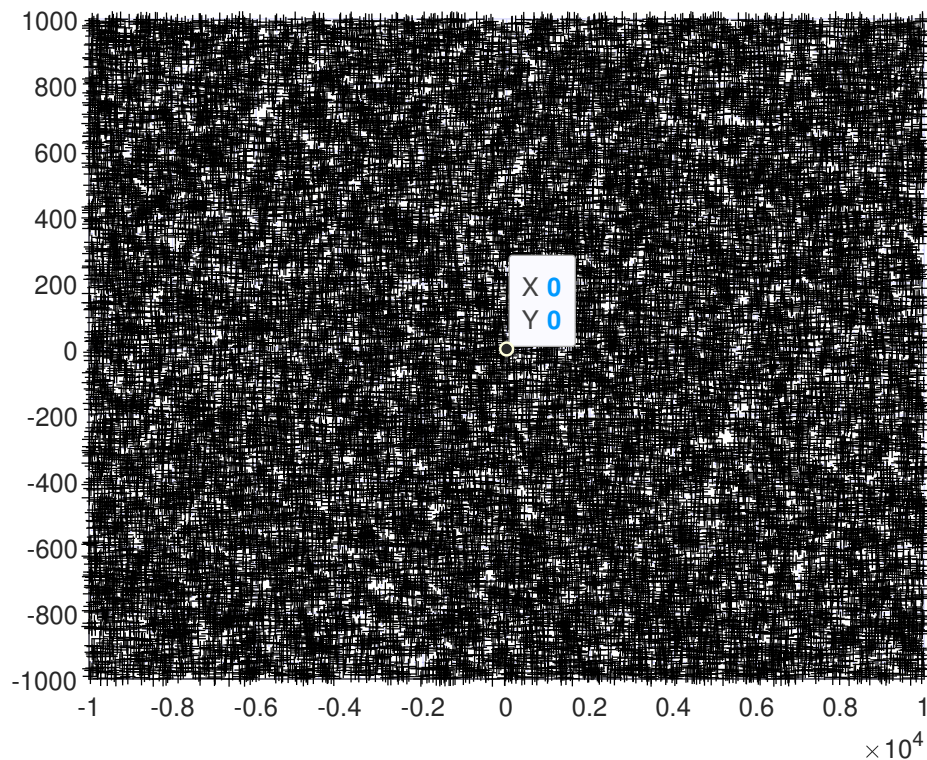


Figura 3.6: Conjunto de 25000 puntos en el plano y el punto  $q$



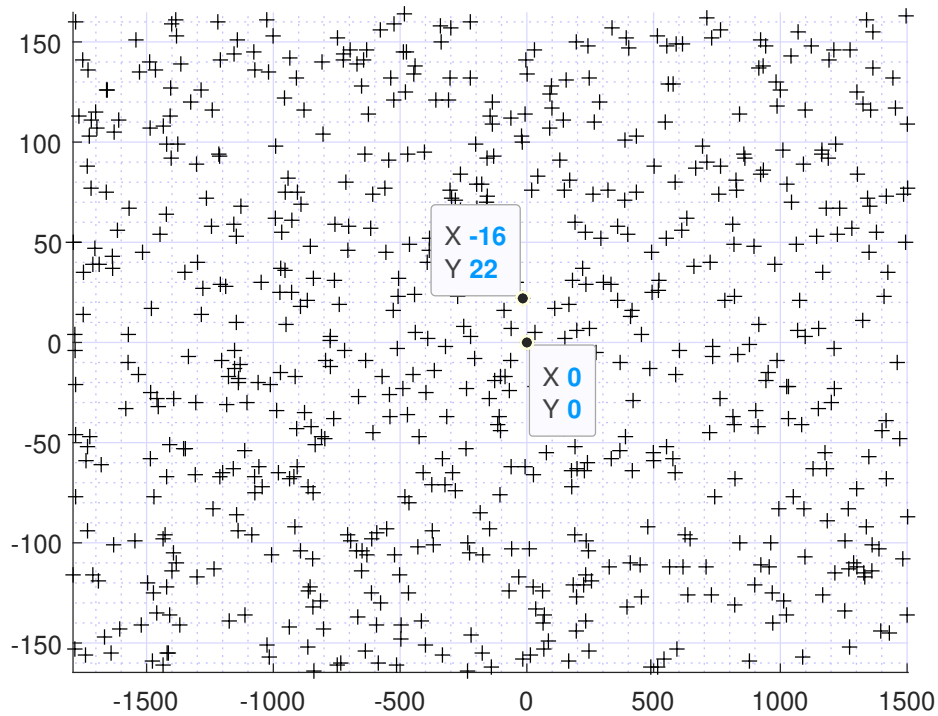


Figura 3.7: Solución al problema NN con un conjunto de 25000 puntos

### 3.2.5. Instancia de 50000 puntos aleatorios

En la figura 3.8 se muestra la distribución de 50000 puntos aleatorios, así como al punto de consulta  $q$ . El punto  $p$  más cercano a  $q$  es el punto  $(11, -4)$  como se aprecia en la figura 3.9 siendo la distancia entre estos dos de 11.7046. Tanto la distancia mínima como el punto  $p$  reportados coinciden usando los dos algoritmos siendo el número de búsquedas realizadas por el Algoritmo 1 un total de 50000 mientras que con el Algoritmo 3 un total de 36 búsquedas.

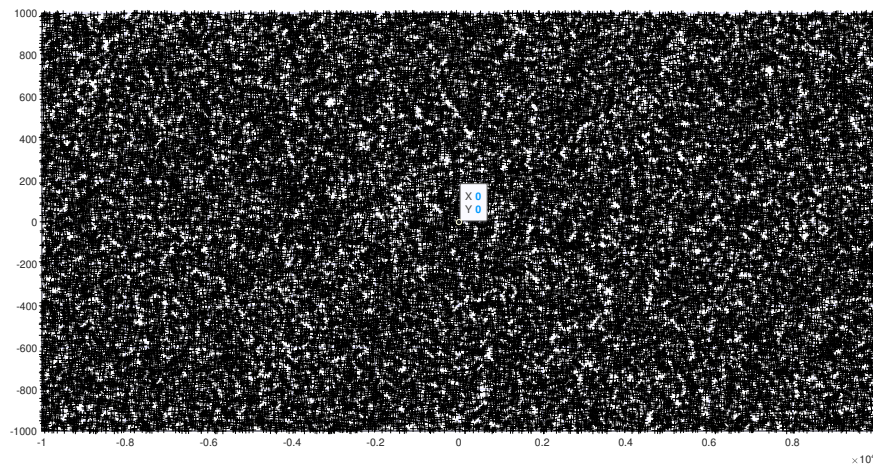


Figura 3.8: Conjunto de 50000 puntos en el plano y el punto  $q$

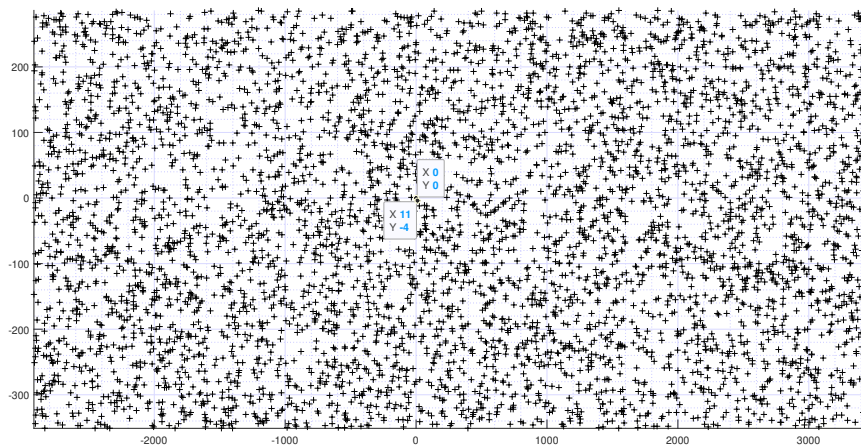


Figura 3.9: Solución al problema NN con un conjunto de 50000 puntos

### 3.2.6. Instancia de 75000 puntos aleatorios

En la figura 3.10 se muestra la distribución de 75000 puntos aleatorios, así como al punto de consulta  $q$ . El punto  $p$  más cercano a  $q$  es el punto  $(-1, 2)$  como se aprecia en la figura 3.11 siendo la distancia entre estos dos de 2.2360. Tanto la distancia mínima como el punto  $p$  reportados coinciden usando los dos algoritmos siendo el número de búsquedas realizadas por el Algoritmo 1 un total de 75000 mientras que con el Algoritmo 3 un total de 35 búsquedas.

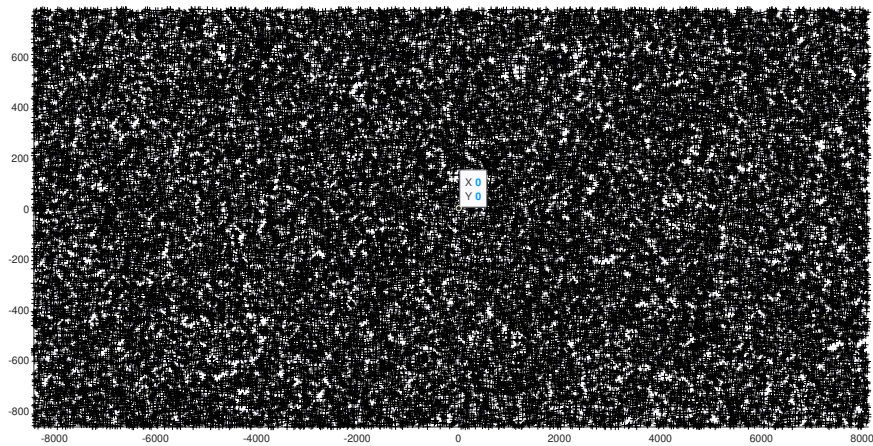


Figura 3.10: Conjunto de 75000 puntos en el plano y el punto  $q$

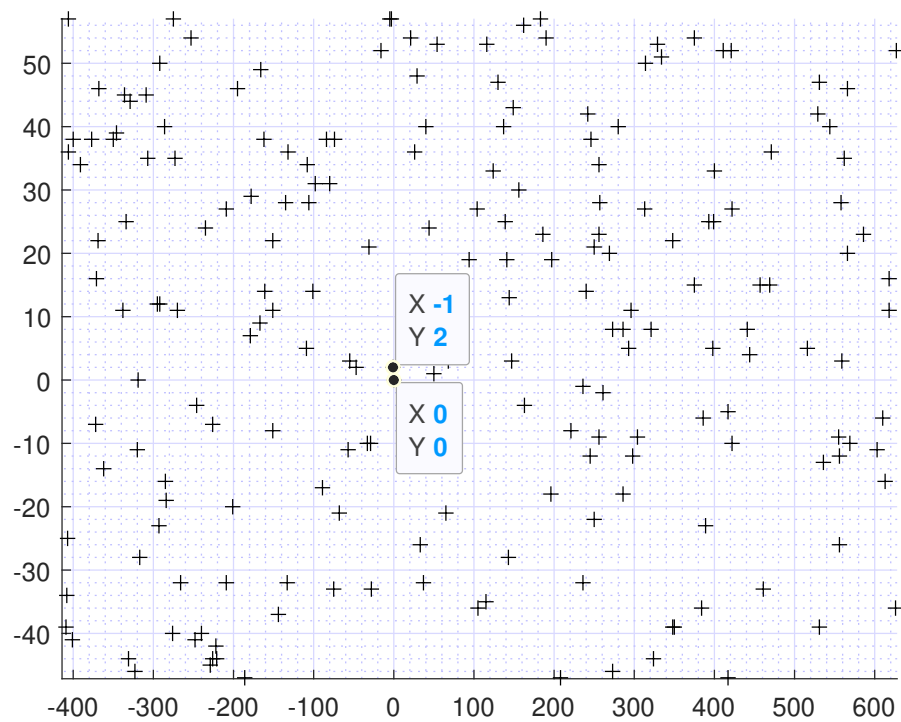
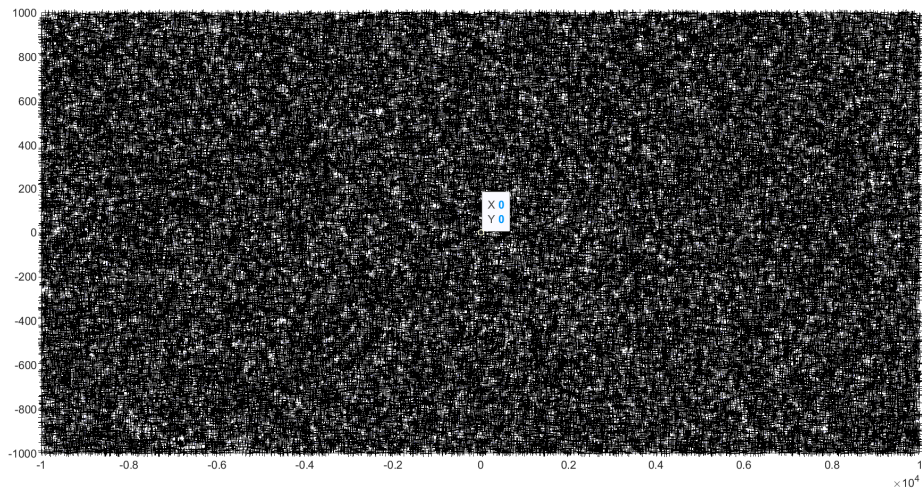


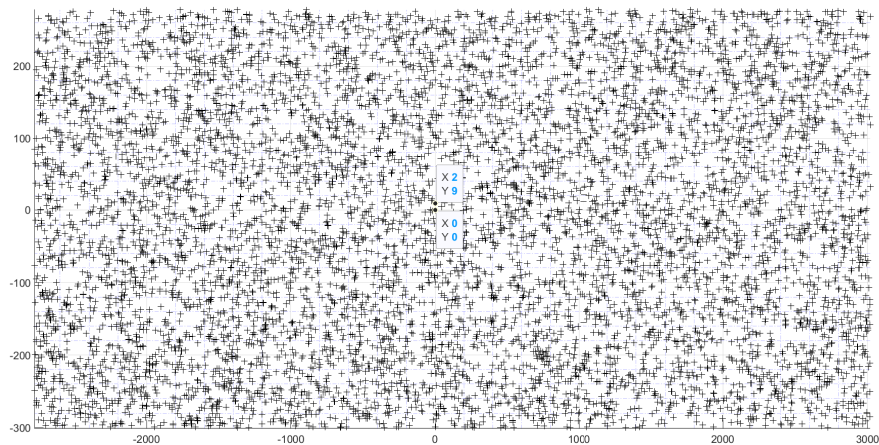
Figura 3.11: Solución al problema NN con un conjunto de 75000 puntos

### 3.2.7. Instancia de 100000 puntos aleatorios

En la figura 3.12a se muestra la distribución de 100000 puntos aleatorios, así como al punto de consulta  $q$ . El punto  $p$  más cercano a  $q$  es el



(a) Conjunto de 100000 puntos



(b) Solución al conjunto de 100000 puntos

Figura 3.12: Problema NN con 100000 puntos

punto  $(2, 9)$  como se aprecia en la figura 3.12b siendo la distancia entre estos dos de 7.8102. Tanto la distancia mínima como el punto  $p$  reportados coinciden usando los dos algoritmos siendo el número de búsquedas realizadas por el Algoritmo 1 un total de 100000 mientras que con el Algoritmo 3 un total de 39 búsquedas.

Se presenta a continuación un resumen con el número de búsquedas realizadas por cada algoritmo para cada instancia, así también el tiempo de cómputo utilizado por cada algoritmo.

Número de búsquedas		
Instancia	Fuerza Bruta	$kd$ -Árbol
100	100	19
1000	1000	25
10000	10000	27
25000	25000	30
50000	50000	36
75000	75000	35
100000	100000	39

Cuadro 3.1: Número de búsquedas realizadas

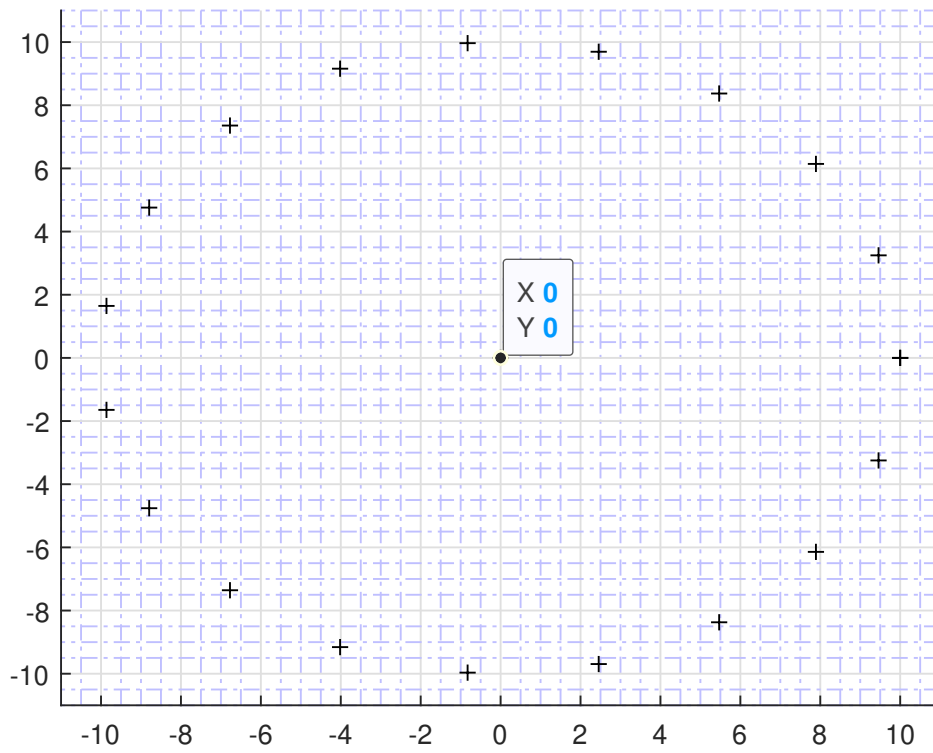
Tiempo de búsqueda [segundos]			
Instancia	Fuerza Bruta	$kd$ -Árbol	Mejoría
100	1.819	2.440	-34.139%
1000	1.963	1.704	13.194%
10000	2.231	1.839	17.570%
25000	2.311	1.951	15.577%
50000	2.586	2.190	15.313%
75000	3.619	2.909	19.618%
100000	4.739	3.902	17.661%

Cuadro 3.2: Tiempo en realizar la búsqueda

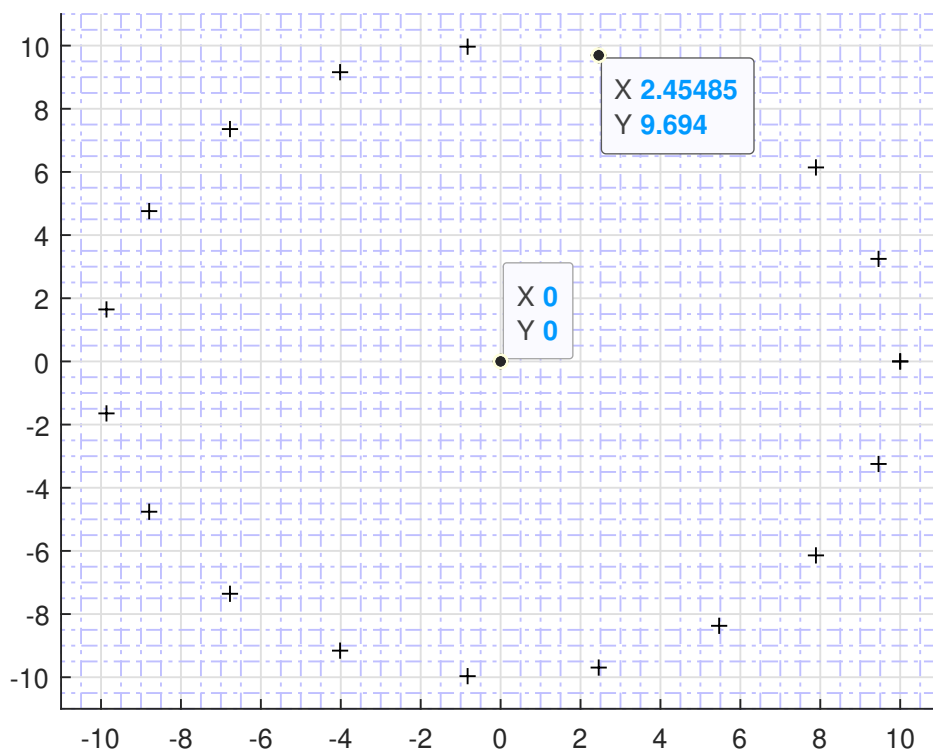
### 3.3. Casos particulares

#### 3.3.1. Conjunto de puntos $P$ todos equidistantes al punto de búsqueda $q$

El conjunto  $P$  está formado por 20 puntos ubicados en una circunferencia de radio 10 como lo muestra la figura 3.13a. El punto  $p$  más cercano a  $q$  es el punto (2,45485, 9,694) como se aprecia en la figura 3.13b siendo la distancia entre estos dos de 10. Tanto la distancia mínima como el punto  $p$  reportados coinciden usando los dos algoritmos siendo el número de búsquedas realizadas por el Algoritmo 1 un total de 20 mientras que con el Algoritmo 3 un total de 8 búsquedas.



(a) Conjunto de 20 puntos



(b) Solución al conjunto de 20 puntos

Figura 3.13: Problema NN en una circunferencia

### 3.4. Conclusiones y recomendaciones

- Al solucionar el problema del vecino más cercano, en instancias de distinto tamaño se observa en el Cuadro 3.1 que al utilizar el 2d-árbol como lo hace el Algoritmo 3 se reduce el número de búsquedas realizadas hasta encontrar la solución correcta, lo cual es consecuente con el análisis de complejidad del Algoritmo 1 en comparación al Algoritmo 3.
- La reducción en el número de búsquedas más marcada se obtiene en la instancia de 100 000 datos, en la cual el Algoritmo 3 busca 39 datos que representan el 0,039 % (omitiendo buscar 99 961 datos) en contraste con el Algoritmo 1 el cual realiza búsquedas en los 100000 datos que representan el 100 % de los datos.
- En el Cuadro 3.2 se observa que, salvo para la instancia con 100 datos, en todas las pruebas el Algoritmo 3 tiene un tiempo de ejecución menor al Algoritmo 1, llegando a una disminución del 19.618 % en la instancia con 75000 datos.
- En el Cuadro 3.2 se observa que para la instancia 1, de 100 datos, al Algoritmo 1 tarda 0.621 segundos menos en llegar a la solución en comparación con el Algoritmo 3, este aumento del 34.359 % en el tiempo se debe al trabajo extra que debe realizar el Algoritmo 3 crear la estructura 2d-árbol.

Dada la importancia práctica a la solución del problema del vecino más cercano en diversos campos, se recomienda estudiar más algoritmos y estructuras de datos que permitan encontrar una solución eficiente al problema cuando la dimensión de los datos aumenta. Actualmente, es un problema abierto y no se conoce una heurística apropiada para datos de dimensión mayor o igual a 20 por lo que el uso del algoritmo de fuerza bruta sigue siendo la mejor forma de encontrar una solución al problema.

---

## Referencias bibliográficas

---

- [1] Alexandr Andoni and Piotr Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06)*, pages 459–468, 2006.
- [2] Sunil Arya. *Nearest Neighbor Searching and Applications*. PhD thesis, USA, 1996. UMI Order No. GAX95-39606.
- [3] Jon Louis Bentley. A Survey of Techniques for Fixed Radius Near Neighbor Searching. 8 1975.
- [4] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, sep 1975.
- [5] W. A. Burkhard and R. M. Keller. Some approaches to best-match file searching. *Commun. ACM*, 16(4):230–236, apr 1973.
- [6] Carlos Eiras-Franco, Bertha Guijarro-Berdiñas, Amparo Alonso-Betanzos, and Antonio Bahamonde. Scalable feature selection using relief aided by locality-sensitive hashing. *International Journal of Intelligent Systems*, 36(11):6161–6179, 2021.
- [7] Carlos Eiras-Franco, Leslie Kanthan, Amparo Alonso-Betanzos, and David Martínez-Rego. Scalable approximate k-nn graph construction based on locality sensitive hashing. In *ESANN*, 2017.
- [8] Jerome H. Friedman, Jon Louis Bentley, and Raphael Ari Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Trans. Math. Softw.*, 3(3):209–226, sep 1977.



- [9] Anthi Karatrantou and Christos Panagiotakopoulos. Algorithm, pseudo-code and lego mindstorms programming. 02 2008.
- [10] Rajesh Matai, Surya Singh, and Murari Lal Mittal. Traveling salesman problem: an overview of applications, formulations, and solution approaches. In Donald Davendra, editor, *Traveling Salesman Problem*, chapter 1. IntechOpen, Rijeka, 2010.
- [11] Antonio Vallecillo Moreno and Rosa Guerequeta García. *Técnicas de diseño de algoritmos*. Servicio de Publicaciones e Intercambio Científico de la Universidad de Málaga, Málaga, 1997.
- [12] Atsuyuki Okabe, Barry Boots, and Kokichi Sugihara. Nearest neighbourhood operations with generalized voronoi diagrams: a review. *International Journal of Geographical Information Systems*, 8(1):43–71, 1994.
- [13] Atsuyuki Okabe, Barry Boots, Kokichi Sugihara, and Sung Nok Chiu. *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams*. Series in Probability and Statistics. John Wiley and Sons, Inc., 2nd ed. edition, 2000.
- [14] K. Rosen. *Discrete Mathematics and Its Applications*. McGraw-Hill Education, 2011.
- [15] Geoffrey G Roy. Designing and explaining programs with a literate pseudocode. *J. Educ. Resour. Comput.*, 6(1):1–es, mar 2006.
- [16] Simon Rubinstein-Salzedo. *Big O Notation and Algorithm Efficiency*, pages 75–83. Springer International Publishing, Cham, 2018.
- [17] Robert Sedgewick and Kevin Wayne. *Algorithms, 4th Edition*. Addison-Wesley, 2011.
- [18] Gregory Shakhnarovich, Trevor Darrell, and Piotr Indyk. *Nearest-Neighbor Methods in Learning and Vision: Theory and Practice (Neural Information Processing)*. The MIT Press, 2006.
- [19] Michael Ian Shamos and Dan Hoey. Closest-point problems. *16th Annual Symposium on Foundations of Computer Science (sfcs 1975)*, pages 151–162, 1975.