# Laboratorio 2
# Par Más Cercano

David Salgado Cortés

**Abstract**

Desde un principio se nos ha comentado la importancia de la complejidad del tiempo de diferentes algoritmos al estar relacionado con los conditional checks lo cual vendría siendo el objetivo de este laboratorio para un algoritmo creado por nosotros para encontrar el par de puntos más cercanos en almenos 3 o más puntos dentro de un plano cartesiano. Este algoritmo plantea cierta cantidad de puntos y les crea unas coordenadas en x aleatorias las cuales posteriormente serán organizadas de menor a mayor. Al ya estar organizados se utiliza la estratégia Divide and Conquer con lo cual nos permite crear grupos más pequeños y ahí utilizar el algoritmo de Brute Force otorgado por el profesor para así poder encontrar el par más cercano. En este caso solo se creará una división entre las coordenadas para así poder crear 2 grupos. Este proceso se realiza 200 veces con cada set de coordenadas para así sacar el tiempo promedio y posteriormente poder sacar una conclusión que se mostrará más adelante en el informe.

## I. Introducción

"La complejidad algorítmica es una métrica teórica que nos ayuda a describir el comportamiento de un algoritmo en términos de tiempo de ejecución (tiempo que tarda un algoritmo en resolver un problema)" [1] Esto nos demuestra la gran importancia que tiene la complejidad algorítmica hoy en día a nivel empresarial.Muchas veces nosotros a este nivel estudiantil no caemos en cuenta de que afuera en el mundo real hay algoritmos tan complejos y extensos que la simplicidad y eficiencia es algo que se debe buscar constantemente para el mejor rendimiento de los programas y poder así seguir avanzando en cuestiones de tecnología. Al determinar que tanto puede durar un algoritmo en resolver un problema podemos seguir buscando diferentes soluciones que logren una mejor productividad en los problemas de hoy en día.

## II. Definición del Problema

En clase ya se nos fue presentado el algoritmo de fuerza bruta de manera recursivo el cual nos permite encontrar el par más cercano siendo este el algoritmo escencial dentro de nuestro problema. Se espera que obtengamos un tiempo de complejidad de O(n), ya que la forma como voy a resolver el problema se basa obtener un set de coordenadas aleatorias, organizarlas y posteriormente dividirla en 2 subsets y aplicar el algoritmo de fuerza bruta dentro de cada subset y comparar sus distancias mínimas encontradas dentro de cada subset para encontrar la distancia mínima dentro de todo el set. Además, dentro del algoritmo se pondrar variables para encontrar las comparaciones y el tiempo de ejecución promedio variando la cantidad de coordenadas aleatorias. Estos resultados serán graficados utilizando Python y serán analizados para concluir si obtuve los resultados esperados. A continuación se presentara el pseudocódigo del algoritmo de Fuerza bruta.

**Algorithm 1** Fuerza Bruta

```
dmin ← INF
for i = [1, N − 1] do
  for j = [i+1, N - 1] do
    d ← distance(coords, i, j)
    if d < d_min then
      first ← i
      second ← j
      dmin ← d
    end if
  end for
end for
return(first, second, dmin)
```

Algoritmo 1 Algoritmo de Fuerza Bruta otorgado por el profesor para encontrar el par más cercano.

## III. Metodología

En primera instancia, al iniciar el algoritmo se crea un set con tamaño n de coordenadas completamente aleatorias, este set después es ordenado de manera ascendente para poder dividir de manera correcta el set completo en 2 subsets. Cabe recalcar que para que haga la división de 2 subsets tienen que presentarse más de 3 coordenadas por que si esto no es así, con el algoritmo de fuerza bruta ya será posible encontrar el par más cercano. No obstante, si se crean los 2 subsets, cada uno se le aplicará el algoritmo de fuerza bruta para poder así encontrar el par más cercano dentro de cada subset. Más adelante, el algoritmo buscará unos candidatos para determinar si coordenadas de diferentes subsets contienen una distancia más pequeña que la encontrada por el algoritmo de fuerza bruta.

Por otro lado, dentro del algoritmo se encuentran variables que nos permiten conocer el número de comparaciones y el tiempo promedio por ejecución ya que estos son los verdaderos resultados que me permiten a mí graficar y observar la complejidad del algorítmo.Cada proceso con la misma cantidad de coordenadas se repite 200 veces, sin embargo, la cantidad de coordenadas va aumentando en un factor de 3/2 comenzando en 500 hasta 1000000. Todos los resultados son plasmados en TXT que contiene 3 diferentes columnas (Cantidad de coordenadas, número de comparaciones promedio y tiempo de ejecución promedio). Con lo cual más adelante se grafico los resultados con la ayuda de Python y me permitió sacar mis conclusiones.

## IV. Resultados

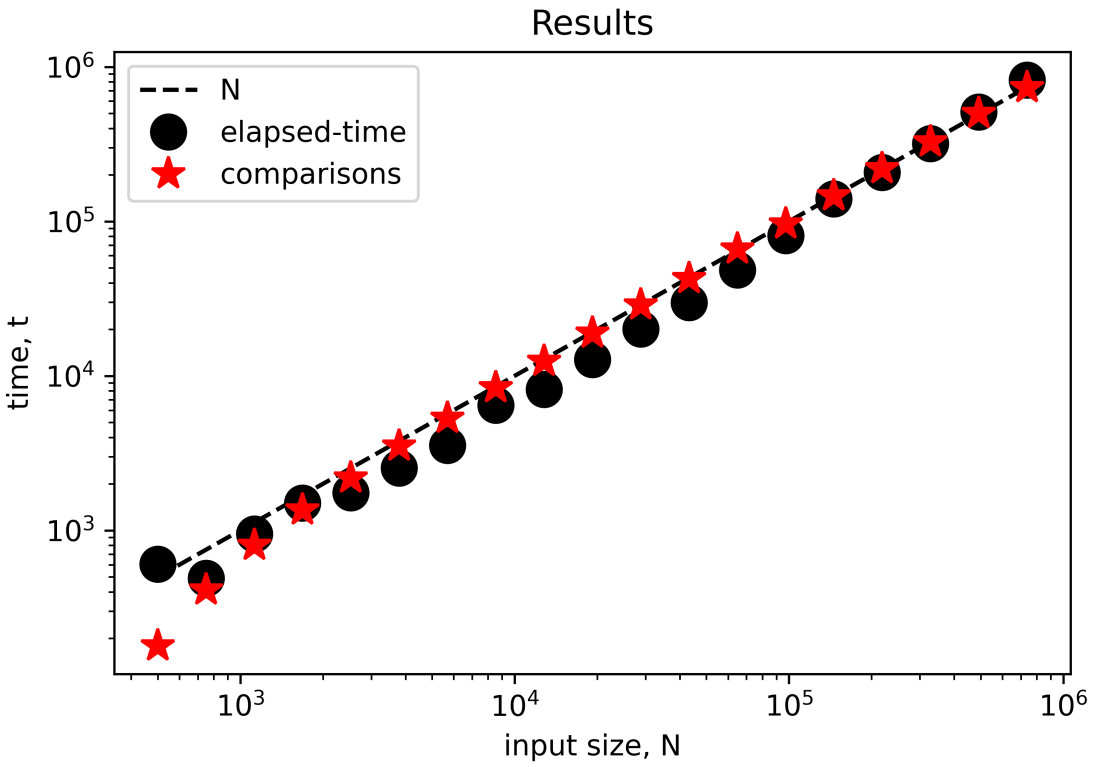
Como fue mencionado anteriormente, los resultados son plasmados dentro de un TXT que contiene 3 columnas. Estas 3 diferentes columnas son: Cantidad de coordenadas por ejecución, número de comparaciones promedio dentro del algoritmo por ejecución y tiempo de ejecución promedio por ejecución respectivamente. Cuando me refiero a ejecución hago referencia a las 200 repeticiones realizadas con la misma cantidad de coordenadas aleatorias dentro del set. Los resultados del TXT son plasmados a continuación:

——————————————————————————N————Comparaciones——-Tiempo——————————————————————————

| | | |
|---|---|---|
| 500 | 1572 | 126065 |
| 750 | 3608 | 102180 |
| 1125 | 7015 | 198001 |
| 1687 | 11913 | 313625 |
| 2530 | 19184 | 365987 |
| 3795 | 30844 | 529395 |
| 5692 | 46649 | 742072 |
| 8538 | 73471 | 1349086 |
| 12807 | 109162 | 1708670 |
| 19210 | 165911 | 2658783 |
| 28815 | 252176 | 4205513 |
| 43222 | 374190 | 6230468 |
| 64833 | 580806 | 10148619 |
| 97249 | 845079 | 16846087 |
| 145873 | 1286634 | 29207892 |
| 218809 | 1921676 | 43388437 |
| 328213 | 2863266 | 66377295 |
| 492319 | 4391100 | 106475577 |
| 738478 | 6429431 | 171103922 |

Con los datos de la tabla se presentó la siguiente gráfica:

Como se puede analizar al observar la gráfica, los datos se presentan de manera casi perfectamente lineal, al iniciar se observa la dispersión de los datos pero cuando tienden al infinito se logra observar como los datos de N, comparaciones y tiempo se alinean para demostrar así el comportamiento lineal que estamos esperando como resultados. En distintas ocasiones el tiempo de ejecución o las comparaciones estan por debajo del esperado, sin embargo, esto puede suceder ya que al momento de buscar los candidatos se les ejecuta el algoritmo de fuerza bruta el cual puede variar mucho dependiendo de la distancia entre todos los puntos, por lo que si la lista de candidatos es pequeña o muy grande esta afecta el tiempo de ejecución considerablemente.




Para comprobar de manera matemática que realmente que la complejidad del algoritmo es lineal, podemos utilizar lo aprendido en clase para analizar algoritmos recursivos a través del uso de la Master Equation analizando la ecuación de complejidad de nuestro algoritmo. Esta ecuación se plantea de la siguiente forma:

$$T(N) = 2T(N/2) + (COMB + DIV)$$

Esto se plantea ya que al momento en el que nosotros hacemos una división significa que el tamaño (N) es divido en 2, siendo la razón por la cual se presenta T(N/2), y al ser 2 subsets eso significa que el algoritmo de Brute Force se tiene que ejecutar 2 veces por cada subset creado a partir de la división, presentando así el 2*T(N/2). Finalmente para saber la complejidad de COMB+DIV al analizar el código se logra ver con claridad que estos al tender al infinito tienen una complejidad constante debido ya que estos procedimientos dependen realmente en la distancia de los puntos teniendo un resultado máximo. Por ende, al final la ecuación se plantea así:

$$T(N) = 2T(N/2) + 1$$

Como fue aprendido en clase, la Master Equation [2] se plantea utilizando 3 variables (a , b y d), en mi caso a=2 , b=2 y d=1 por lo que en base al siguiente cuadro de decisión:

$$T(N) = \begin{cases} O(N^d), if a < b^d \\ O(N^d logN), if a = b^d \\ O(N^{log_b a}), if a > b^d \end{cases}$$

Por lo cual en mi caso se tiene en cuenta el tercer caso donde $a > b^d = 2 > 1$ otorgando así una complejidad de:

$$O(N^{log_b a}) = O(N^{log_2 2}) = O(N^1)$$

brindandonos la certeza de que la complejidad presentada en la gráfica si esta correcta y que la complejidad esperada tambien fue certera respecto a los resultados presentados.

## V. Conclusión

Para concluir este laboratorio podemos decir con certeza que logramos el objetivo principal ya que encontramos la complejidad del algoritmo a trabajar. Esto fue analizado en base a la cantidad de coordenadas, la cantidad de comparaciones por

ejecución y tiempo de ejecución. Estos valores a simple vista nos presentaron una complejidad lineal como se observa en la gráfica presentada ya que a medidad que aumentan la cantidad de coordenadas por set, es lógico pensar que las comparaciones y el tiempo de ejecución aumentan demostrando que las variables son directamente proporcionales entre ellas. Por otro lado, tambien logramos encontrar la complejidad del algoritmo de manera matemática con lo aprendido en clase y el uso de la Master Equation, estos resultados al igual que el esperado obtuvimos el mismo resultado de que la complejidad del algoritmo realizado para el problema planteado tiene una complejidad de O(n).

Para finalizar, puedo decir de que no se presentaron graves problemas en la realización de este laboratorio, lo más dificil fue realmente plantear la parte lógica en un codigo funcional de JAVA para que pueda proporcionar unos resultados. No obstante, gracias al trabajo realizado en clase otras funciones y subprocedimiento fueron reutilizadas de trabajos anteriores tales como la creación y modificación de TXT, graficar unos datos basado en TXT, Crear un set de numeros aleatorios, entre otros. Para futuras investigaciones puede ser interesante saber que sucede si el set no se divide unicamente en 2 sino que se dividiera en 4 por ejemplo donde las coordenadas en Y ya tendrían su importancia por lo cual sería bueno preguntarse si ¿Entre más divisiones se creen para un mismo set, la complejidad del algoritmo puede aumentar ?.

## REFERENCES

[1] J. Guillermo, "¿qué es la complejidad algorítmica y con qué se come?" 2018.
[2] J. McConnell, *Analysis of Algorithms*, ser. Reference, Information and Interdisciplinary Subjects Series. Jones & Bartlett Learning, 2008.