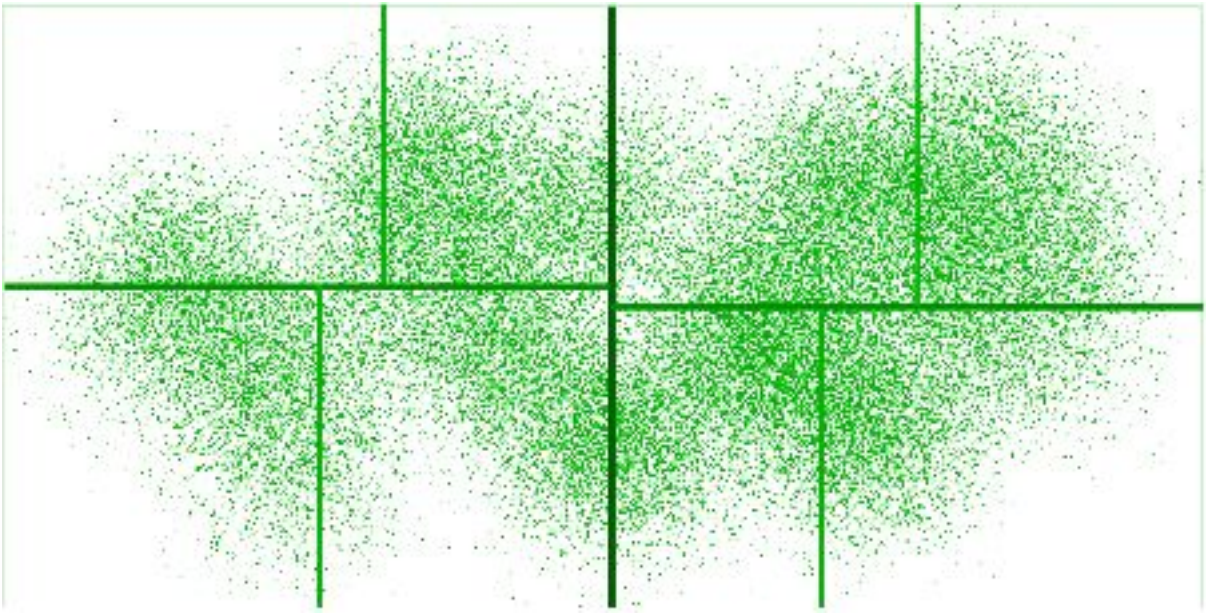
***Grupo 3***
***Integrantes: Daniel Delgado Gomis, Carlos Rodriguez Bellmunt,***
***Carlos Jiménez Moreno, Carlos Romero Gallén***

# MEMORIA ENTREGABLE 4 KD-TREES



# Objetivos

El objetivo a realizar en esta práctica es el de poner el práctica los conocimientos del algoritmo divide y vencerás. Para poner esto en práctica, el entregable 4 nos propone obtener un kd-tree dado un fichero de texto que contiene los puntos.

# Actividades realizadas

Lo primero que se debió realizar para poder resolver este entregable fue la realización de la función **read_points.** Esta función recibe el nombre de un archivo como parámetro y lo lee línea a línea.

```
def read_points(filename:str)-> List[Tuple[float, float]]:
    with open(filename, mode='r') as file:
        puntos_2D = []
        for linea in file:
            valores_punto = extract_Point(linea)
            puntos_2D.append(valores_punto)
    return puntos_2D
```

Una vez tienes las líneas, se apoya en una función auxiliar que  hemos realizado para mejor legibilidad del código. Esta función se denomina **extract_Point** y recibe como parámetro una línea. En ella se extraen los valores del punto que describen y la devuelve como una tupla con estos valores convertidos en float.

```
def extract_Point(linea):
    valores_linea = linea.split(' ')
    coordenada_x = float(valores_linea[0])
    coordenada_y = float(valores_linea[1])
    valores_punto = (coordenada_x, coordenada_y)
    return valores_punto
```

Coste temporal: O(n)
Coste espacial:  O(n)

Por otro lado para la construcción del Kdtree se ha tenido que hacer uso de la función **build_kd_tree**. Esta función recibe como parámetro la lista de tuplas que contiene los puntos (obtenida con read_points). En ella se aplica el algoritmo de divide y vencerás y se devuelve el Kdtree creado.

```
def build_kd_tree(points: List[Tuple[float, float]]) -> KDTree:


    #Casos base

    if(len(points)==0):
        return None
    elif (len(points) ==1):
        return KDTree.KDLeaf(points[0])

    #Recursividad

    else:
        cantidadDePuntos = len(points)
        medio = cantidadDePuntos // 2
        ordenacionHorizontal = sorted(points, key=lambda i: i[0])
        ordenacionVertical = sorted(points, key=lambda i: i[1])

        if(ordenacionHorizontal[-1][0]-ordenacionHorizontal[0][0] >= ordenacionVertical[-1][1]-ordenacionVertical[0][1]):
            eje = HORIZONTAL
        else:
            eje = VERTICAL

        if(eje==HORIZONTAL):
            arbol1 = build_kd_tree(ordenacionHorizontal[0:medio])
            arbol2= build_kd_tree(ordenacionHorizontal[medio:])
            mediana = hallaMediana(medio, ordenacionHorizontal, eje)
        else:
            arbol1 = build_kd_tree(ordenacionVertical[0:medio])
            arbol2= build_kd_tree(ordenacionVertical[medio:])
            mediana = hallaMediana(medio, ordenacionVertical, eje)


        nodo = KDTree.KDNode(eje, mediana, arbol1 ,arbol2 )
        return nodo
```

Coste temporal: O(n log^2(n)
Coste espacial: O(n^2)

Además usamos la llamada a una función propia llamada **hallaMediana**, que nos devuelve la mediana a partir del índice del punto medio redondeado hacia arriba (indistintamente de si es mediana o no), la lista introducida, ordenada según el eje y el eje de ordenación.

```
def hallaMediana(medio: int, points: List[Tuple[float, float]], eje: Orientacion) -> float:
    cantidadDePuntos = len(points)
    if (cantidadDePuntos % 2 == 0):
        return (points[medio-1][eje] + points[medio][eje]) / 2
    return points[medio][eje]
```

Coste temporal: O(1)
Coste espacial: O(n)

Por último tenemos el **main** que hará uso de las funciones creadas anteriormente. Primero leerá el archivo y obtendrá la lista de tuplas que representan a los puntos. Luego hace uso de build_kd_tree para obtener el kdtree deseado. Para finalizar, imprimer por pantalla haciendo uso del método proporcionado en la clase kdtree, llamado pretty.

```
if __name__ == "__main__":
    if len(sys.argv) >= 2:
        list_points = read_points(sys.argv[1])
        kd_tree_solution = build_kd_tree(list_points)
        print(kd_tree_solution.pretty())
    else:
        print('Argumentos introducidos incorrectos.')
```

# Costes Totales

| Coste Espacial | Coste Temporal |
|---|---|
| O(n^2) | $O(n \log^2(n))$ |

# Conclusión

El entregable a realizar ha sido fácil de resolver debido a que tanto en clase como en los seminarios se han explicado los fundamentos del algoritmo divide y vencerás. Teniendo claros los conocimientos previamente, aplicar una solución al problema no ha tenido gran dificultad.

En cuanto a los costes obtenidos, son apropiados para el problema a resolver.