**Reporte de proyecto final**

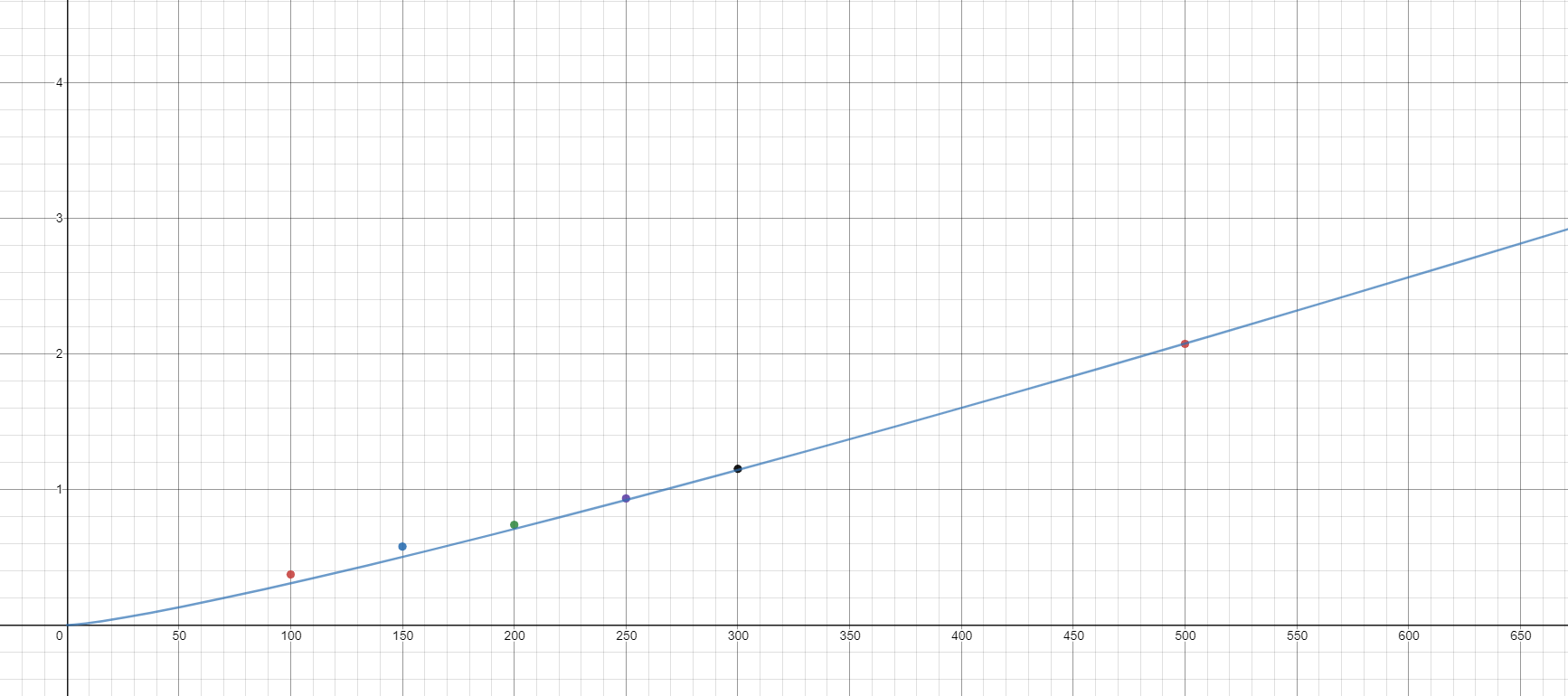
**Algoritmo utilizado para encontrar el algoritmo**

def calcularRuta(campo3D, puntoInicio:list, puntoFin:list):  
  
 puntoAnt:list = puntoInicio #O(1)  
 puntoActual:list = puntoInicio #O(1)  
  
 ruta:list = [] #O(1)  
 veces = 0  
 while puntoActual != puntoFin: #O(n)  
  
 ruta.append(puntoActual) #n\*O(1) -> #O(n)  
 puntosVecinos: list = buscarVecinos2(campo3D, puntoActual) #n\*O(27) -> #O(n)  
 #PuntosVecinos tiene 26 puntos (son los vecinos)  
 listaPuntos = list() #n\*O(1) -> #O(n)  
  
 for puntoPrueba in puntosVecinos: #n\*O(26) -> O(26n) -> #O(n)  
  
 vectores: list = vectorizar(puntoAnt, puntoActual, puntoPrueba) #n\*O(1) -> #O(n)  
 pUno: list = vectores[0] #n\*O(1) -> #O(n)  
 pDos: list = vectores[1] #n\*O(1) -> #O(n)  
 avance: bool = productoPunto(pUno, pDos) >= 0 #n\*O(1) -> #O(n)  
 disponble: bool = campo3D[puntoPrueba[0], puntoPrueba[1], puntoPrueba[2]] == 0 #n\*O(1) -> #O(n)  
 if avance and disponble: #n\*O(1) -> #O(n)  
 distancia = calcularDistancia(puntoPrueba, puntoFin) #n\*O(1) -> #O(n)  
 tupla: tuple = (distancia, puntoPrueba) #n\*O(1) -> #O(n)  
 if tupla not in listaPuntos: listaPuntos.append(tupla) #n\*O(1) -> #O(n)  
  
 listaPuntos.sort(reverse=True) #n\*O(nlong(n)) -> O((n^2)long(n))  
 puntoMasCercano: list = list(listaPuntos.pop()[1]) #n\*O(1) -> #O(n)  
  
 puntoAnt = puntoActual #n\*O(1) -> #O(n)  
 puntoActual = puntoMasCercano #n\*O(1) -> #O(n)  
 veces+=1 #n\*O(1) -> #O(n)  
 ruta.append(tuple(puntoFin)) #O(1)  
  
 **#Tiempo estimado del algoritmo =>> O((n^2)log(n))**  
  
 return ruta

**Estimación teórica del tiempo asintótico de ejecución**

O((n^2) log(n))

**Gráfico del tiempo real de ejecución**



**Análisis**

El algoritmo puede ser catalogado como un algoritmo voraz debido a que se selecciona el punto al que se va a desplazar según el punto que tenga la menor distancia hacia el punto final, luego se marca como visitado al punto en el que se estuvo anteriormente para no volver a visitarlo.

Cabe recalcar que el algoritmo aplica la ley del mínimo esfuerzo para que este se ejecute de manera más rápida. Es decir, siempre busca realizar una línea recta entre el punto inicial y el final, y en caso de encontrase con un obstáculo, simplemente lo rodea.