

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

Facultad de Estudios Superiores Aragón



Ingeniería en Computación

Diseño de Análisis y Algoritmos

Profesor: Jesus Hernandez Cabrera

Alumno: Alcántara Vázquez Alejandro Gael

No. De cuenta: 320085930

Semestre:2027-1

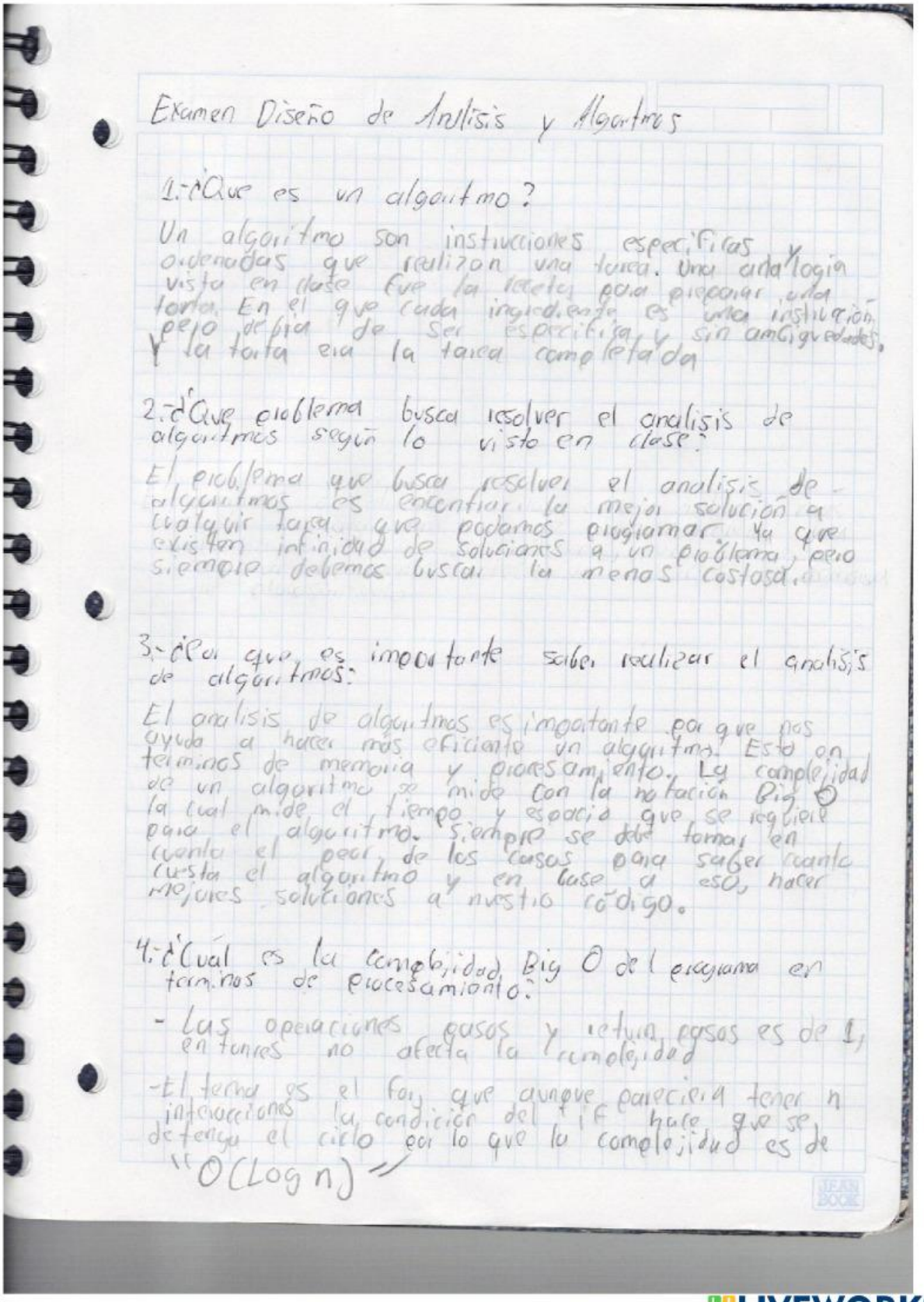
Grupo: 1558

Examen 1

Fecha de entrega: 01/11/2026

LIVEWORKSHEETS

Sección 1



Examen Diseño de Análisis y Algoritmos

1.- ¿Que es un algoritmo?

Un algoritmo son instrucciones específicas y ordenadas que realizan una tarea. Una analogía vista en clase fue la receta para preparar una torta. En el que cada ingrediente es una instrucción pero debía de ser específica y sin ambigüedades. Y la torta era la tarea completada

2.- ¿Que problema busca resolver el analisis de algoritmos según lo visto en clase?

El problema que busca resolver el analisis de algoritmos es encontrar la mejor solución a cualquier tarea que podamos programar. Ya que existen infinidad de soluciones a un problema, pero siempre debemos buscar la menos costosa.

3.- ¿Por que es importante saber realizar el analisis de algoritmos?

El analisis de algoritmos es importante por que nos ayuda a hacer más eficiente un algoritmo. Esto en terminos de memoria y procesamiento. La complejidad de un algoritmo se mide con la notación Big O la cual mide el tiempo y espacio que se requiere para el algoritmo. Siempre se debe tomar en cuenta el peor de los casos para saber cuanto cuesta el algoritmo y en base a eso, hacer mejores soluciones a nuestro código.

4.- ¿Cuál es la complejidad Big O del programa en terminos de procesamiento?

- Las operaciones pasos y return pasos es de 1, entonces no afecta la complejidad

- El ciclo es el for que aunque parecería tener n interacciones la condición del IF hace que se detenga el ciclo por lo que la complejidad es de "O(Log n)"

LIVEWORKSHEETS

Sección 2

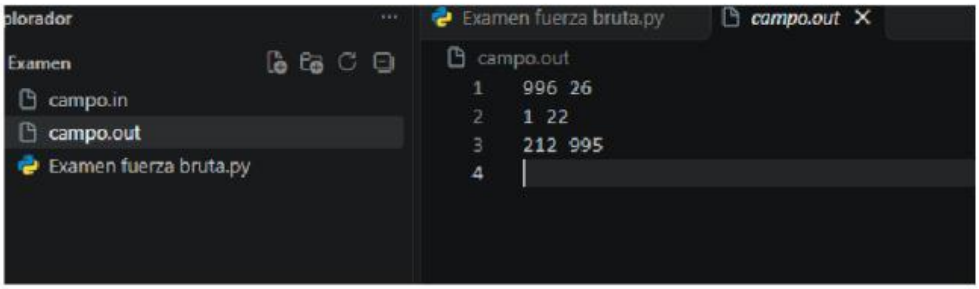
Código

```
puntos = []
with open('campo.in', 'r') as f:
    for linea in f:
        partes = linea.split()
        if len(partes) == 2:
            x = int(partes[0])
            y = int(partes[1])
            # condicion de paro
            if x == -1 and y == -1:
                break
            puntos.append((x, y))
max_area = 0
mejores_postes = []
n = len(puntos)
# evaluamos todas las combinaciones de 3 postes
for i in range(n - 2):
    for j in range(i + 1, n - 1):
        for k in range(j + 1, n):
            xA, yA = puntos[i]
            xB, yB = puntos[j]
            xC, yC = puntos[k]
            # frmula del area de un triángulo
            area = abs(
                xA * (yB - yC) +
                xB * (yC - yA) +
                xC * (yA - yB)
```

LIVEWORKSHEETS

```
            ) / 2.0
            # guardamos el area más grande
            if area > max_area:
                max_area = area
                mejores_postes = [
                    (xA, yA),
                    (xB, yB),
                    (xC, yC)
                ]
# generamos el archivo de salida con las coordenadas
with open('campo.out', 'w') as f:
    for poste in mejores_postes:
        f.write(f"{poste[0]} {poste[1]}\n")
```

Este código es el de fuerza bruta, hace cada combinación posible hasta poder encontrar el área del triángulo más grande. Utilice 3 for anidados para la complejidad por lo que su Big O es de N^ 3. Por lo mismo que hace cada combinación posible le tardó a mi computadora procesar un tiempo de 21 segundos para generar el archivo de salida. Las coordenadas del archivo de salida fueron:

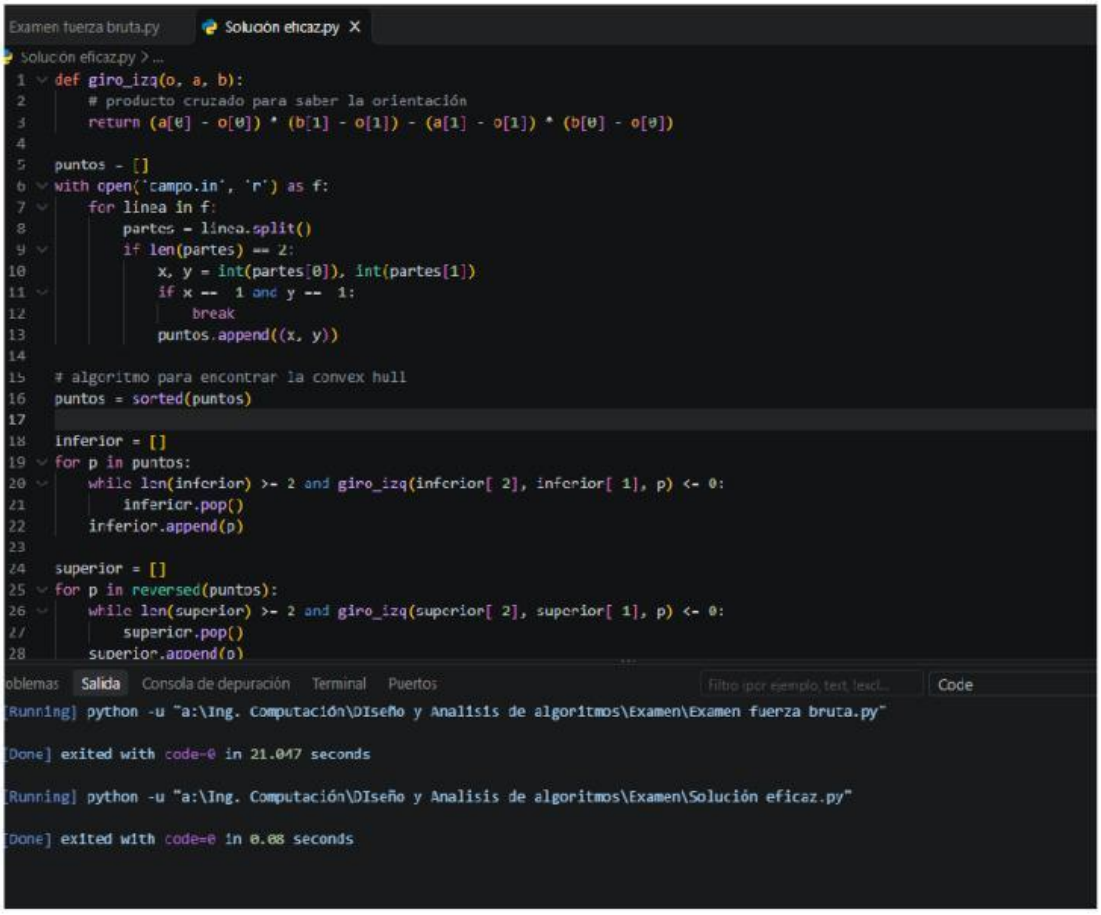


LIVEWORKSHEETS

Solución Mejorada

Para encontrar la mejor solución estuve viendo varios conceptos geométricos aplicados en algoritmos y el que parece aplicar es el de “Convex Hull”.

Para encontrar los postes ideales para el triángulo más grande es evidente que serán de los postes de las orillas, por lo que primero se calculará ese perímetro primero, así descartando muchos más postes irrelevantes para la solución. Haciendo que los 3 for se enfoquen solo es los de la orilla




```
def giro_izq(o, a, b):
    # producto cruzado para saber la orientación
    return (a[0] - o[0]) * (b[1] - o[1]) - (a[1] - o[1]) * (b[0] - o[0])
```

LIVEWORKSHEETS

```
puntos = []
with open('campo.in', 'r') as f:
    for linea in f:
        partes = linea.split()
        if len(partes) == 2:
            x, y = int(partes[0]), int(partes[1])
            if x == -1 and y == -1:
                break
            puntos.append((x, y))

# algoritmo para encontrar la convex hull
puntos = sorted(puntos)

inferior = []
for p in puntos:
    while len(inferior) >= 2 and giro_izq(inferior[-2], inferior[-1], p) <= 0:
        inferior.pop()
    inferior.append(p)

superior = []
for p in reversed(puntos):
    while len(superior) >= 2 and giro_izq(superior[-2], superior[-1], p) <= 0:
        superior.pop()
    superior.append(p)

# unimos las partes superiores e inferiores quitando los puntos duplicados en los extremos
envolvente = inferior[:-1] + superior[:-1]
```

LIVEWORKSHEETS

```
# aplicamos fuerza bruta solo sobre los puntos de la envolvente
max_area = 0
mejores_postes = []
h = len(envolvente)

for i in range(h - 2):
    for j in range(i + 1, h - 1):
        for k in range(j + 1, h):
            xA, yA = envolvente[i]
            xB, yB = envolvente[j]
            xC, yC = envolvente[k]

            # calculamos el área
            area = abs(xA * (yB - yC) + xB * (yC - yA) + xC * (yA - yB)) / 2.0

            if area > max_area:
                max_area = area
                mejores_postes = [(xA, yA), (xB, yB), (xC, yC)]

# generamos el resultado
with open('campo.out', 'w') as f:
    for poste in mejores_postes:
        f.write(f"{poste[0]} {poste[1]}\n")
```

LIVEWORKSHEETS