CUCEI

CENTRO UNIVERSITARIO DE CIENCIAS EXACTAS E INGENIERÍAS

Análisis de Algoritmos

Algoritmo para elección de Hardware en función de un videojuego

Implementado con Divide y Vencerás

Integrantes del Equipo:
Gutierrez Vazquez Axel
Hernandez Macias Axxel Gael
Quintero Arreola Laura Vanessa

Profesor:
Lopez Arce Delgado Jorge Ernesto

Asignatura: Análisis de Algoritmos
Sección: D06 **Calendario:** 2025 B

Fecha de entrega: 26 de octubre de 2025

Indice

Tabla de asignación de roles	. . . Página 3
Introducción y Objetivos	. . . Página 5
Definición de la técnica divide y vencerás	. . . Página 6
Investigación Previa	. . . Página 7
Desarrollo del Algoritmo	. . . Página 9
Resultados del Algoritmo	. . . Página 13
Complejidad Computacional del Algoritmo	. . . Página 17
Conclusiones Personales	. . . Página 20
Bibliografia	. . . Página 20

Tabla de asignación de roles

En la siguiente tabla se escribe cuáles serán las actividades/tareas por desarrollar para cada integrante del equipo, así como cuál es el resultado esperado y para cuando se debe entregar la respectiva tarea, de esta forma se tendrá un control de que es lo que falta para terminar la actividad y que todos sean parte de la elaboración del reporte y el algoritmo.

Integrante del equipo	Tarea a realizar	Resultado	Fecha de entrega
Gutierrez Vazquez Axel	Realizar la introducción para el reporte.	Escribir con sus palabras la introducción del contenido del deporte y la actividad en general.	26 de octubre
	Buscar en internet en qué consiste la técnica de divide y vencerás.	Entregar un resumen de la investigación y escribir las referencias en APA.	25 de octubre
	Investigar para qué sirve la librería REQUESTS.	Entregar un resumen de la investigación y escribir las referencias en APA.	25 de octubre
	Investigar la importancia del hardware en una computadora.	Entregar un resumen de la investigación y escribir las referencias en APA.	25 de octubre
	Cambiar la parte donde se aplica fuerza bruta por divide y vencerás.	Entregar el código con divide y vencerás en la parte en la que se elige la mejor opción de hardware.	22 de octubre
	Explicar cómo se cambió el algoritmo.	Desarrollar la explicación del algoritmo dentro del reporte.	25 de octubre
Hernandez Macias Axxel Gael	Analizar la complejidad computacional del algoritmo.	Escribir los resultados del análisis y graficarlo.	26 de octubre
	Definir cuales son los	Escribir con sus palabras	21 de

Quintero Arreola Laura Vanessa	objetivos de realizar esta actividad.	qué objetivos pretendemos desarrollar con este algoritmo.	octubre
	Investigar qué es y para qué sirve una API.	Entregar un resumen de la investigación y escribir las referencias en APA.	25 de octubre
	Investigar para qué sirve la librería PYSIDE6.	Entregar un resumen de la investigación y escribir las referencias en APA.	25 de octubre
	Definir cómo cambiar el algoritmo de fuerza bruta a divide y vencerás.	Hacer una pequeña introducción en el desarrollo dentro del reporte y justificar el trabajo.	21 de octubre
	Realizar los diagramas de flujo para explicar gráficamente el código.	Entregar y añadir los diagramas de flujo junto con su explicación en el reporte.	26 de octubre
	Hacerle una GUI al algoritmo	Hacer una interfaz gráfica y enviarla al respectivo apartado y añadir evidencias al reporte.	23 de octubre

NOTA* Para el desarrollo de la presentación se usará la información más relevante del reporte, es decir, cada integrante tendrá que aportar al desarrollo de la presentación tomando como punto de partida sus tareas asignadas.

Introducción

En el estudio de los algoritmos, la técnica Divide y Vencerás ocupa un papel indispensable debido a su amplia aplicabilidad en la resolución de problemas. Este método ha permitido el desarrollo de soluciones más rápidas y estructuradas para situaciones en las que otros enfoques resultan demasiado costosos o complejos. Su relevancia no solo se limita al ámbito teórico, sino que también se ve reflejada en la práctica, siendo la base de una gran cantidad de algoritmos modernos utilizados en áreas como la informática, la optimización y el procesamiento de datos.

Para este proyecto, se implementará Divide y Vencerás en un algoritmo que permite crear diferentes combinaciones de hardware para una computadora en base a un presupuesto establecido por el usuario.

Objetivos

- ➢ Investigar y entender cómo funciona la técnica de divide y vencerás para la resolución de algoritmos.
- ➢ Analizar cómo modificar un algoritmo que funciona con fuerza bruta para que pueda funcionar con divide y vencerás.
- ➢ Investigar qué librerías facilitan el uso de API’s en Python.
- ➢ Investigar qué librerías sirven para desarrollar interfaces de usuario.
- ➢ Hacer un análisis de complejidad computacional para saber que tan bueno o malo es el resultado de nuestro algoritmo implementado.
- ➢ Desarrollar diagramas de flujo para explicar de forma gráfica el funcionamiento del algoritmo.

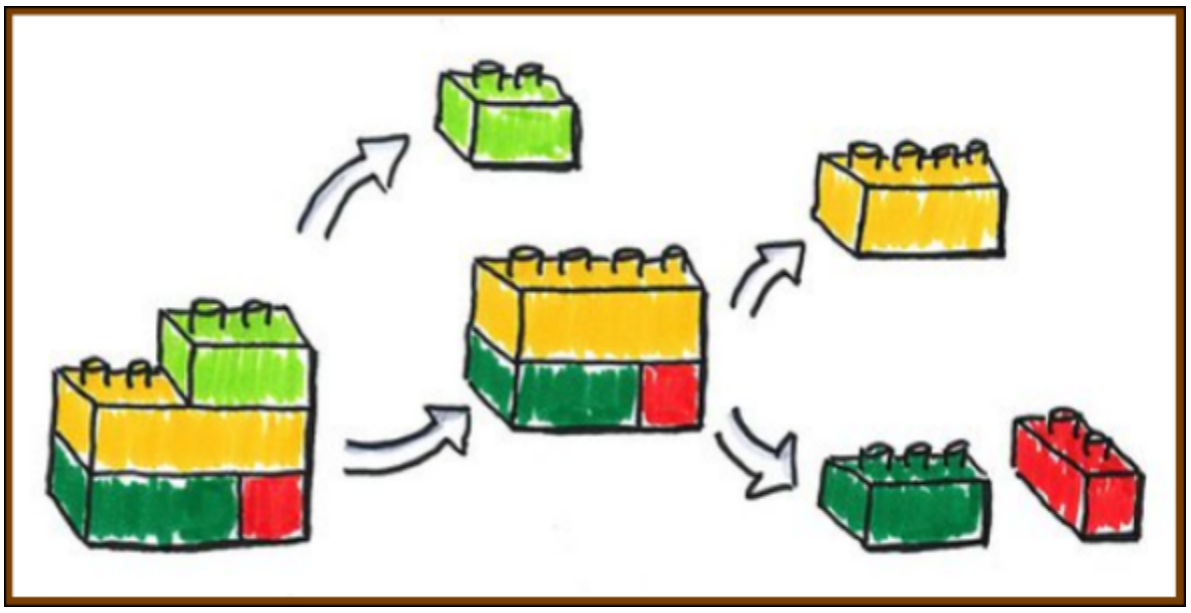
Definición de la técnica divide y vencerás

Divide y vencerás (Divide and Conquer) es un paradigma de diseño algorítmico que se basa en la idea de descomponer un problema grande o complejo en varios subproblemas más pequeños y manejables. Cada subproblema se resuelve de manera independiente por lo general de forma recursiva y posteriormente, las soluciones parciales se combinan para formar la solución completa del problema original. Este enfoque permite reducir la complejidad del problema y facilita el desarrollo de algoritmos más eficientes y estructurados.

El proceso suele dividirse en tres fases principales: dividir, conquistar y combinar:

1. En la primera, el problema se separa en partes más pequeñas del mismo tipo.
2. En la segunda, cada parte se resuelve de forma individual, ya sea directamente o mediante nuevas divisiones.
3. En la tercera, la fase de combinación, se integran los resultados obtenidos para construir la solución final.

Esta técnica es ampliamente utilizada en el diseño de algoritmos por su capacidad para mejorar el rendimiento y optimizar el uso de recursos. Además, favorece la paralelización de tareas, ya que los subproblemas pueden resolverse simultáneamente en sistemas multicore o distribuidos. Entre los ejemplos más representativos del enfoque Divide y Vencerás se encuentran los algoritmos Merge Sort, Quick Sort y la Búsqueda Binaria, que aplican este principio para ordenar listas o encontrar elementos de manera más rápida y eficiente.



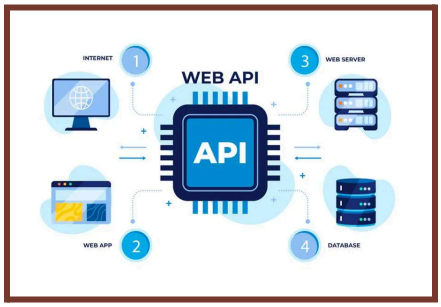
Investigación Previa

La investigación previa abarca aspectos técnicos necesarios para poder desarrollar el algoritmo de la mejor manera, eso incluye la investigación de ciertas librerías en específico y que elementos clave nos ayudarán a resolver la problemática de forma óptima.

API (Application Programming Interface)

Una interfaz de programación de aplicaciones mejor conocida como API por sus siglas en inglés, es un conjunto de reglas que le permite a distintas aplicaciones realizar un intercambio de información y funciones. Las APIs son herramientas muy útiles no solo para el intercambio de información sino también para desarrollar nuevas aplicaciones para atender distintas necesidades, es decir, contar con una API puede hacer que el proceso de desarrollo tome información de una base de datos más sólida que si comenzara desde cero y si tomamos eso en cuenta en retrospectiva se ahorra tiempo.

Además es importante entender cómo funciona una API, la respuesta en concreto sería que la API es por decirlo de una forma, una especie de intermediario entre dos aplicaciones, por un lado tenemos al que necesita la información que podríamos llamarlo cliente y el que tiene la información el cual es el servidor, cuando un cliente requiere cierta información del servidor se realiza una solicitud a la API para conseguir los datos de interés, entonces la API extrae la información que necesita del servidor para poder brindarle el servicio al cliente.



Es fundamental entender que cuando se hace uso de una API no se puede solicitar toda clase de datos, en parte las API tienen esa protección de que la información que se va a compartir es segura y no representa ningún riesgo de vulnerabilidad para el servidor.

Libreria REQUESTS

Es una librería de python que permite realizar solicitudes de HTTP de manera sencilla. Con ella, los desarrolladores pueden enviar y recibir datos a través de la web, conectarse a API's, descargar información de sitios web o interactuar con servicios en línea. Facilita el manejo de

métodos como GET, POST, PUT y DELETE, además de permitir la gestión de encabezados, autenticación, cookies y parámetros en las solicitudes.

Libreria PYSIDE6

PySide6 es la biblioteca oficial que conecta Python con Qt, un framework de C++ usado para crear interfaces gráficas. Ofrece múltiples herramientas para desarrollar aplicaciones gráficas y trabajar con tecnologías como XML, redes, bases de datos y más. Funciona en Windows, Linux y macOS, lo que la hace ideal para crear programas multiplataforma con apariencia nativa. Además, puede utilizarse tanto con licencia comercial como en proyectos de código abierto bajo licencias LGPLv3 o GPLv2.

Para crear una GUI básica utilizando pyside6 se necesitan usar los siguientes elementos básicos:

- Importar las clases necesarias (QApplication, QWidget, QLabel).
- Crear una instancia de QApplication.
- Crear una ventana (QWidget), darle título, tamaño y posición.
- Añadir un widget como QLabel para mostrar texto, por ejemplo “Hola, Mundo!”.
- Mostrar la ventana con .show() y ejecutar la app con .exec().

Importancia de un buen armado de hardware en una computadora

Contar con un hardware correctamente configurado resulta esencial, ya que de ello depende el desempeño general del equipo, su vida útil y su posibilidad de ampliación o actualización con el tiempo. Cuando se eligen de forma adecuada los componentes físicos y se ensamblan con precisión, el ordenador no solo ofrece un funcionamiento más fluido y estable, sino que también puede responder mejor a las exigencias de distintos tipos de tareas, desde las más básicas hasta las más demandantes. Un montaje bien planificado evita fallos innecesarios y permite aprovechar al máximo las capacidades del sistema.

Desarrollo del Algoritmo

Justificación

Para desarrollar la técnica aprendida en clase de divide y vencerás decidimos seguir utilizando el algoritmo que habíamos desarrollado que usaba fuerza bruta. Para saber que algoritmo podíamos hacer, buscamos una problemática que nos pareciera interesante, y después de analizarlo y pensar en un tema que fuera de nuestro interés, pensamos que sería bueno desarrollar un algoritmo que ayudará al usuario a saber que componentes de hardware comprar para jugar un juego que les llamara la atención.

Nuestra problemática surge porque sabemos que existen muchas personas que juegan videojuegos y muchas de esas personas les gustaría en un futuro tener una mejor experiencia de juego, pero no todo el mundo conoce qué componentes son los mejores o los más adecuados (calidad - precio) para armar su pc. Por eso nuestro algoritmo pretende saber dos cosas del usuario un juego que llame su atención en particular y cuál es el presupuesto que tiene para invertir en su compra, con esta información nos interesa que el algoritmo tenga la capacidad de darle una buena recomendación de hardware al usuario y de esta forma mejorar su experiencia de juego.

Explicación del algoritmo

Se implementó PySide6, que permite buscar un videojuego en Steam y recomendar una combinación de componentes adecuada según los requisitos del juego y el presupuesto del usuario. La interfaz gráfica fue desarrollada con QSS, utilizando una paleta de colores oscuros. El programa se conecta a la API oficial de steam para obtener información del juego, como su nombre, precio, plataformas disponibles, requisitos mínimos y recomendados, así como la imagen del mismo. A partir de los requisitos del sistema, se extraen las especificaciones de CPU. GPU Y RAM necesarias.

La lógica del algoritmo radica en la función Build, que debe encontrar la mejor combinación de componentes que se ajuste al presupuesto y cumpla con los requisitos del juego, para hacer esto de forma eficiente, en lugar de comparar cada CPU contra cada posible combinación, se implementa una optimización fundamental basada en Divide y vencerás.

```python
def Build(presupuesto, cpu_req_name, gpu_req_name):
    """Recomienda una build basada en presupuesto y requisitos"""
    cpus, gpus, ram_options = CargarComponentes()
    cpu_obj = BuscarComponentes(cpus, cpu_req_name)
    gpu_obj = BuscarComponentes(gpus, gpu_req_name)
    cpu_target = cpu_obj["score"] if cpu_obj else 0
    gpu_target = gpu_obj["score"] if gpu_obj else 0

    gpu_ram_combos = []
    for gpu in gpus:
        for ram_size, ram_price in ram_options.items():
            price = gpu["price"] + ram_price
            partial_score = (0.6 * gpu["score"]) + (0.1 * (ram_size * 1000))
            gpu_ram_combos.append({"gpu": gpu, "ram_size": ram_size, "ram_price": ram_price, "price": price, "partial_score": partial_score})

    gpu_ram_combos.sort(key=lambda x: x["price"])

    prefix_best = []
    best_combo = None
    for combo in gpu_ram_combos:
        if best_combo is None or combo["partial_score"] > best_combo["partial_score"]:
            best_combo = combo
        prefix_best.append(best_combo)

    import bisect
    mejores = None
    mejor_val = -1
```

Primero se generan todas las combinaciones posibles de GPU y RAM y se ordenan por precio: `gpu_ram_combos.sort(key=lambda x: x["price"])` – Este paso es indispensable para que funcione el algoritmo.

```python
    gpu_target = gpu_obj["score"] if gpu_obj else 0

    gpu_ram_combos = []
    for gpu in gpus:
        for ram_size, ram_price in ram_options.items():
            price = gpu["price"] + ram_price
            partial_score = (0.6 * gpu["score"]) + (0.1 * (ram_size * 1000))
            gpu_ram_combos.append({"gpu": gpu, "ram_size": ram_size, "ram_price": ram_price, "price": price, "partial_score": partial_score})

    gpu_ram_combos.sort(key=lambda x: x["price"])

    prefix_best = []
    best_combo = None
    for combo in gpu_ram_combos:
        if best_combo is None or combo["partial_score"] > best_combo["partial_score"]:
            best_combo = combo
        prefix_best.append(best_combo)

    import bisect
    mejores = None
    mejor_val = -1
    prices = [c["price"] for c in gpu_ram_combos]

    for cpu in cpus:
        remaining = presupuesto - cpu["price"]
        if remaining < 0:
            continue
        idx = bisect.bisect_right(prices, remaining) - 1
        if idx >= 0:
            best_partial = prefix_best[idx]
```

El código itera sobre cada CPU y calcula el presupuesto restante:

`Remaining = presupuesto - cpu["price"]`

```
prefix_best = []
best_combo = None
for combo in gpu_ram_combos:
    if best_combo is None or combo["partial_score"] > best_combo["partial_score"]:
        best_combo = combo
    prefix_best.append(best_combo)

import bisect
mejores = None
mejor_val = -1
prices = [c["price"] for c in gpu_ram_combos]

for cpu in cpus:
    remaining = presupuesto - cpu["price"]
    if remaining < 0:
        continue
    idx = bisect.bisect_right(prices, remaining) - 1
    if idx >= 0:
        best_partial = prefix_best[idx]
```

En lugar de usar otro bucle para encontrar el mejor combo de GPU/RAM, se utiliza la línea:
`Idx = bisect.bisect_right(prices, remaining) - 1`

La función `bisect` es una implementación del algoritmo de búsqueda binaria y se aplica de la siguiente manera:

- Divide - Toma la lista ordenada de precios y la divide a la mitad.
- Vence - Compara el presupuesto restante con el precio central. Si el presupuesto es mayor, descarta la primera mitad de la lista, si es menor, descarta la segunda.
- Repite - Vuelve a dividir la mitad restante, repitiendo el proceso hasta encontrar el índice exacto del combo más caro que el usuario puede pagar.

Con este método se puede reducir el tiempo de búsqueda de manera significativa, cambiando una complejidad lineal O(n) a una logarítmica O(log n), permitiendo que se recomiende la mejor build de una manera más rápida y eficiente.

Existe un importante caso de respaldo. Si el algoritmo de optimización, después de todas sus búsquedas, no logra encontrar una combinación válida (lo que podría ocurrir si el presupuesto es demasiado bajo incluso para la CPU más barata), el programa no falla. En su lugar, activa una lógica de "mínimos": selecciona la CPU más barata, la GPU más barata y la opción de RAM más económica disponible. Esto asegura que el usuario siempre reciba una recomendación, aunque sea la más básica posible, en lugar de un error.

Para obtener mayor eficiencia se utiliza la clase Worker (que hereda de QThread), un hilo de ejecución secundario. El propósito de este hilo es realizar todo el trabajo "pesado" (localizar el juego, obtener sus detalles, descargar la imágen y ejecutar la función Build) en segundo plano. De esta forma, la ventana principal permanece siempre receptiva. Este Worker se comunica con la interfaz mediante "Señales" de PySide6, que son la forma segura de enviar datos entre hilos. Emite una señal cuando tiene los resultados de texto, otra cuando tiene la imagen descargada y una tercera si ocurre algún error.

La clase MainWindow es la ventana principal que ve el usuario. Al iniciarse, esta clase construye todos los elementos visuales definidos en el código: los campos para el nombre del juego y el presupuesto, el botón de búsqueda y el área de resultados. Esta clase también almacena internamente los datos SVG (gráficos vectoriales) para los íconos de plataformas (Windows, Mac y Linux) que se mostrarán junto a la imagen.

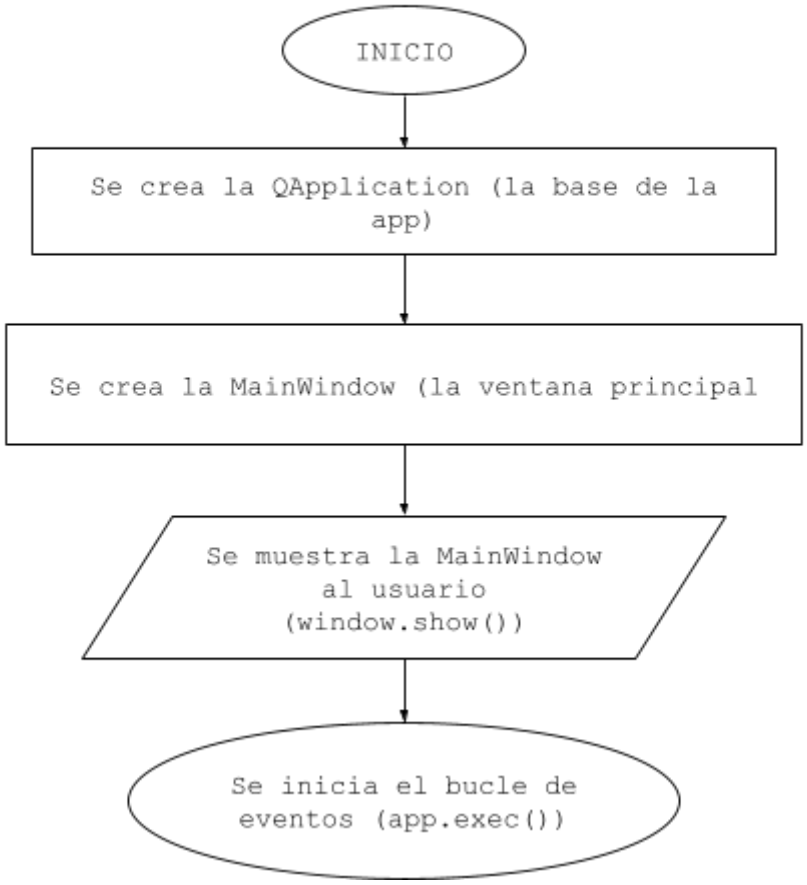
Cuando el usuario presiona el botón "Buscar", se activa la función start_search. Esta función primero valida que las entradas no estén vacías y que el presupuesto sea un número. Luego, deshabilita el botón de búsqueda (para evitar clics duplicados) y muestra un mensaje de "Buscando...". Lo más importante es que aquí se crea la instancia del Worker, se le pasan el nombre del juego y el presupuesto, y se conectan sus señales a las funciones correspondientes de la MainWindow.

Finalmente, la MainWindow simplemente espera a recibir las señales del Worker. Cuando llega la señal result_ready, la función show_results se activa, formatea toda la información recibida (detalles del juego y la "build" sugerida) y la muestra en el área de resultados. De igual manera, show_game_image recibe la imagen, la escala y la muestra. Si se recibe una señal de error, la función show_error lo notifica al usuario y vuelve a habilitar el botón de búsqueda. El bloque final (if __name__ == "__main__":) es el punto de entrada estándar de Python que simplemente inicia la aplicación y muestra la ventana principal.

Resultados del Algoritmo

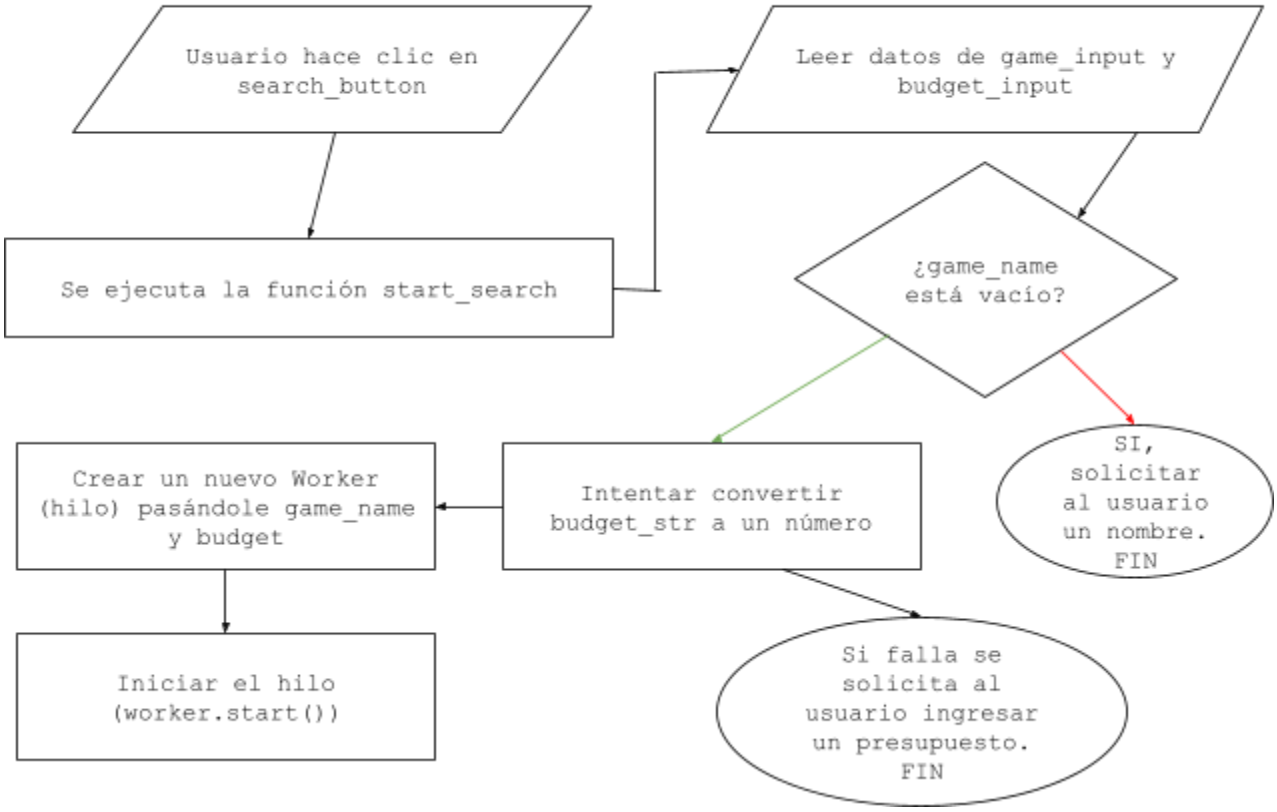
Para poder entender mejor el algoritmo que implementamos, se muestran a continuación diagramas de flujo que nos explica el paso a paso de forma gráfica del funcionamiento interno de nuestro código. Como nuestro algoritmo cuenta con una interfaz gráfica funciona de una forma más compleja el código por tanto representaremos esta complejidad mediante tres diagramas de flujo que nos explican cosas diferentes del código.

Diagrama 1.
Flujo Principal (Inicio de la Aplicación)



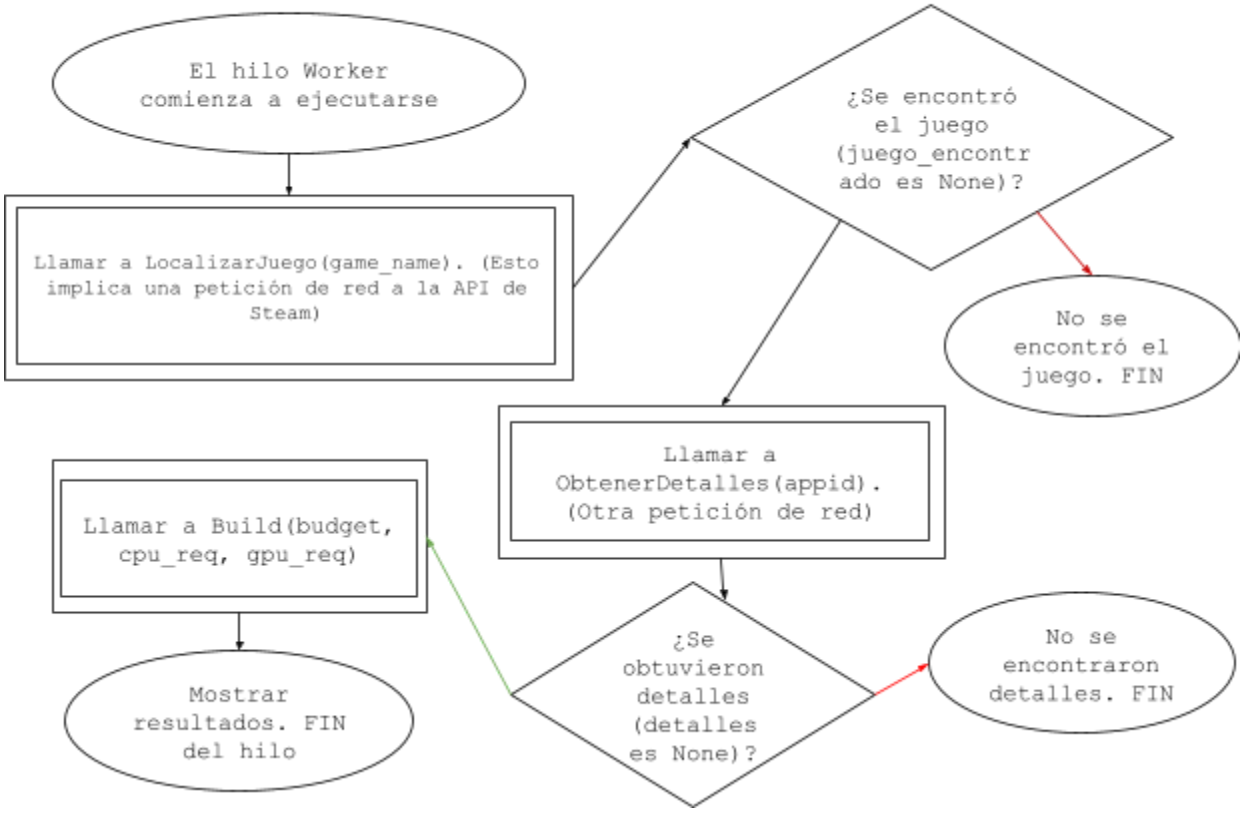
En este primer diagrama se explica el flujo principal del código, donde se inicia la interfaz gráfica y se crea la ventana principal del código que a su vez hace que se inicien diferentes procesos que incluye los elementos gráficos que podemos apreciar en la GUI, el apartado de los resultados, y la señales que se conectan a otras funciones dentro del código para hacer la búsqueda del hardware. Por último se queda esperando en un bucle a que el usuario interactúe con la interfaz.

Diagrama 2.
Flujo de Interacción (Función start_search)



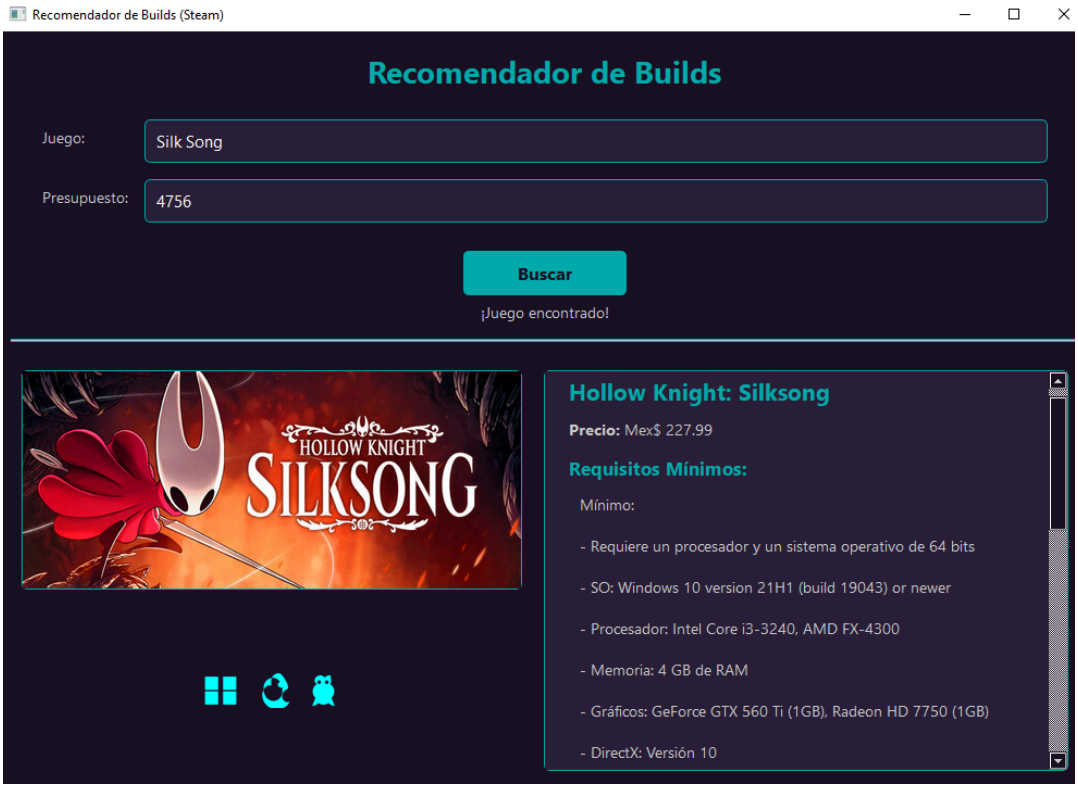
En este segundo diagrama se aprecia el flujo que hay cuando el usuario interactúa con la interfaz gráfica, donde el usuario hace clic y escribe los valores correspondientes en cada apartado, nombre del juego y su presupuesto, al hacer clic en buscar se leen los datos ingresados de cada apartado y en caso de ser válidos se crea un hilo que recibe la información.

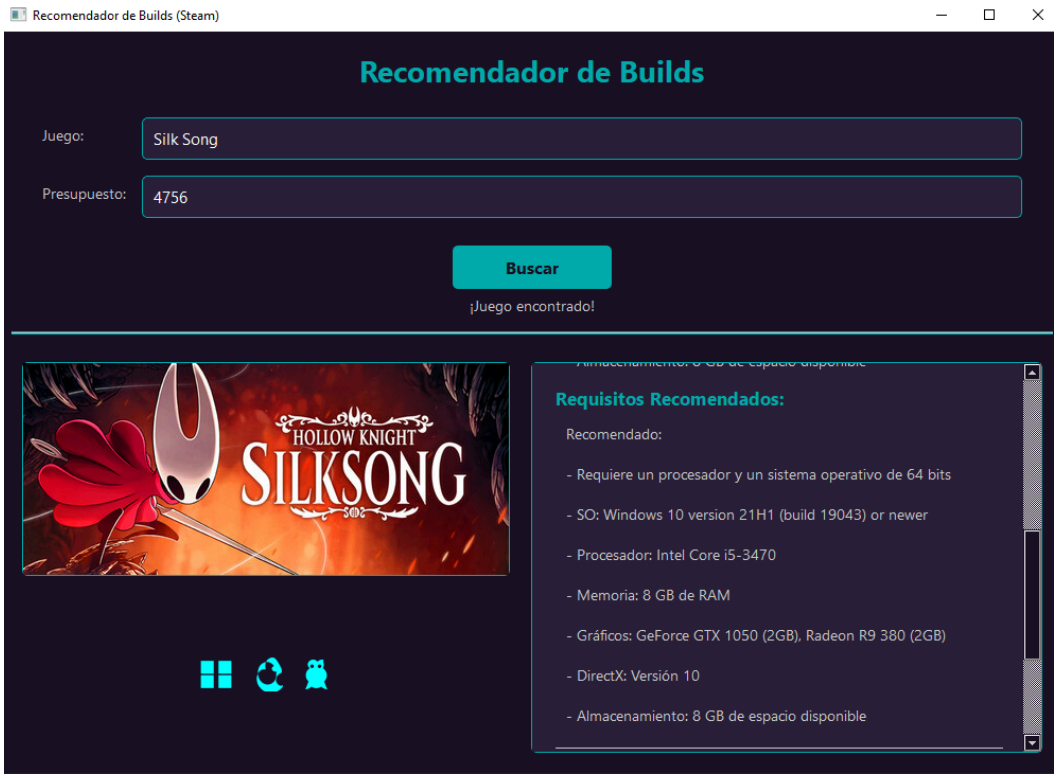
Diagrama 3.
Flujo del Hilo (Lógica Worker.run)

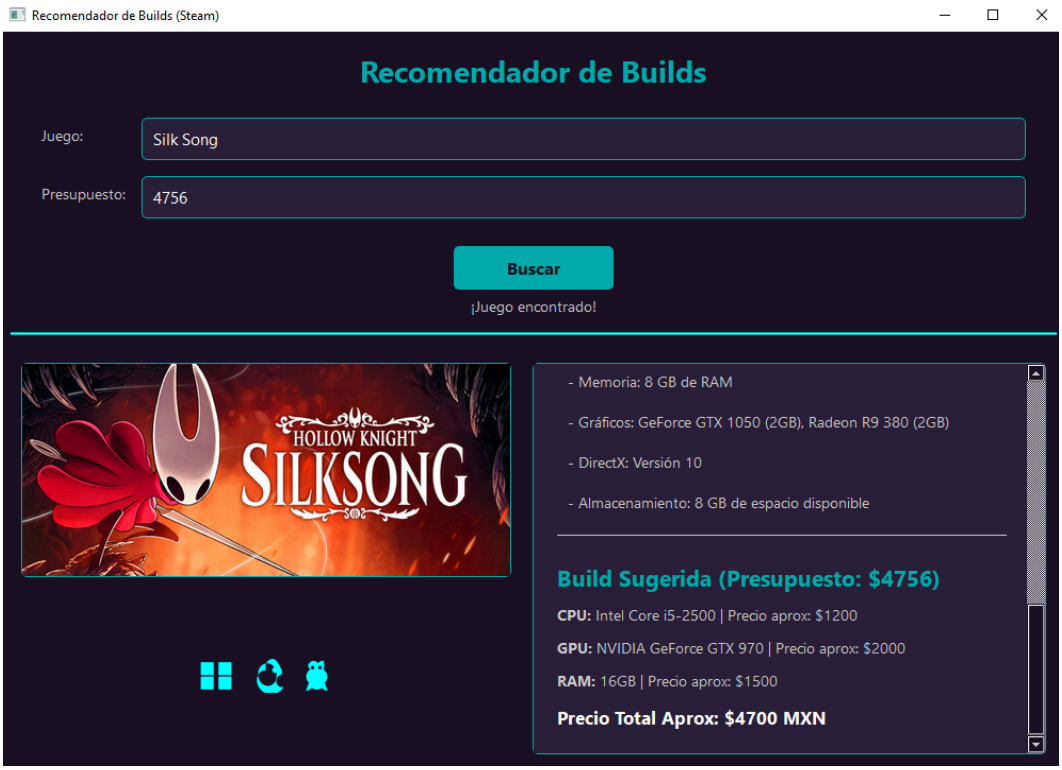


Por último tenemos el tercer diagrama que maneja la lógica de como saber cual es la mejor combinación de hardware en función del juego y el presupuesto, con la información que nos brinda el usuario, hacemos una petición a una API que pertenece a Steam, para buscar el juego y obtener los detalles, por último se usa la función para armar la mejor build y mostrar los resultados en la interfaz gráfica.

Interfaz grafica







Complejidad Computacional del Algoritmo

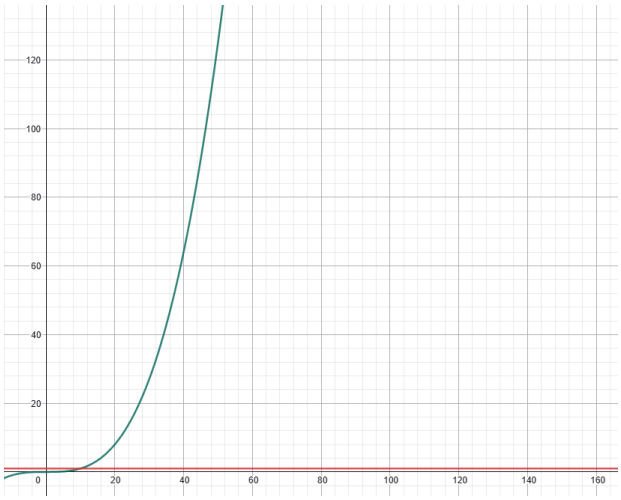
El análisis de la complejidad computacional se realizó con el propósito de poder comparar el comportamiento temporal y espacial de nuestras estrategias algorítmicas aplicadas al mismo problema.

Para analizar este estudio se establecen complejidades teóricas y se representan en gráficas con el fin de observar el crecimiento relativo de cada una conforme aumenta el tamaño del problema.

Para lograrlo, se emplea una herramienta fundamental que es la notación asintótica. Esta nos permitirá estudiar nuestros respectivos códigos y su comportamiento de crecimiento.

En esta práctica se utilizó la notación asintótica O(Big O) ya que nos permite identificar de manera eficiente su complejidad temporal y espacial describiendo el peor escenario que puede presentar el algoritmo. Esta notación se enfoca en la descripción del crecimiento del tiempo de ejecución y en el uso de memoria a medida que (n) aumenta su tamaño de entrada. Se enfoca en las operaciones más relevantes para centrarse en la tasa de crecimiento más significativa y así poder comparar la eficiencia de ambos algoritmos.

Complejidad computacional del algoritmo Fuerza Bruta



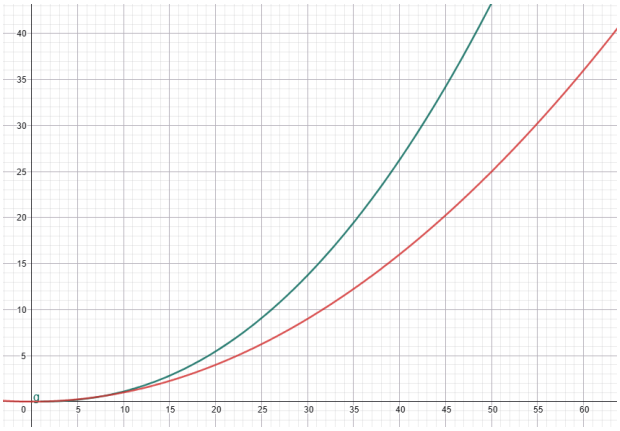
La complejidad temporal fue determinada como O(c·g·r) donde cada una de las variables representan nuestras posibles combinaciones entre CPU, GPU y RAM.
Este comportamiento corresponde a un crecimiento cúbico $O(c \cdot g \cdot r) \approx O(n^3)$

La complejidad espacial es O(1) donde se muestra como una constante en 1.

Un dato relevante al tomar en cuenta es que la complejidad de este algoritmo es que como contra, el tiempo crece cubicamente, sería algo inaceptable si los valores crecen a listas mayores y como un pro es que funciona con el espacio mínimo por lo que es aceptado en entornos con memoria limitada.

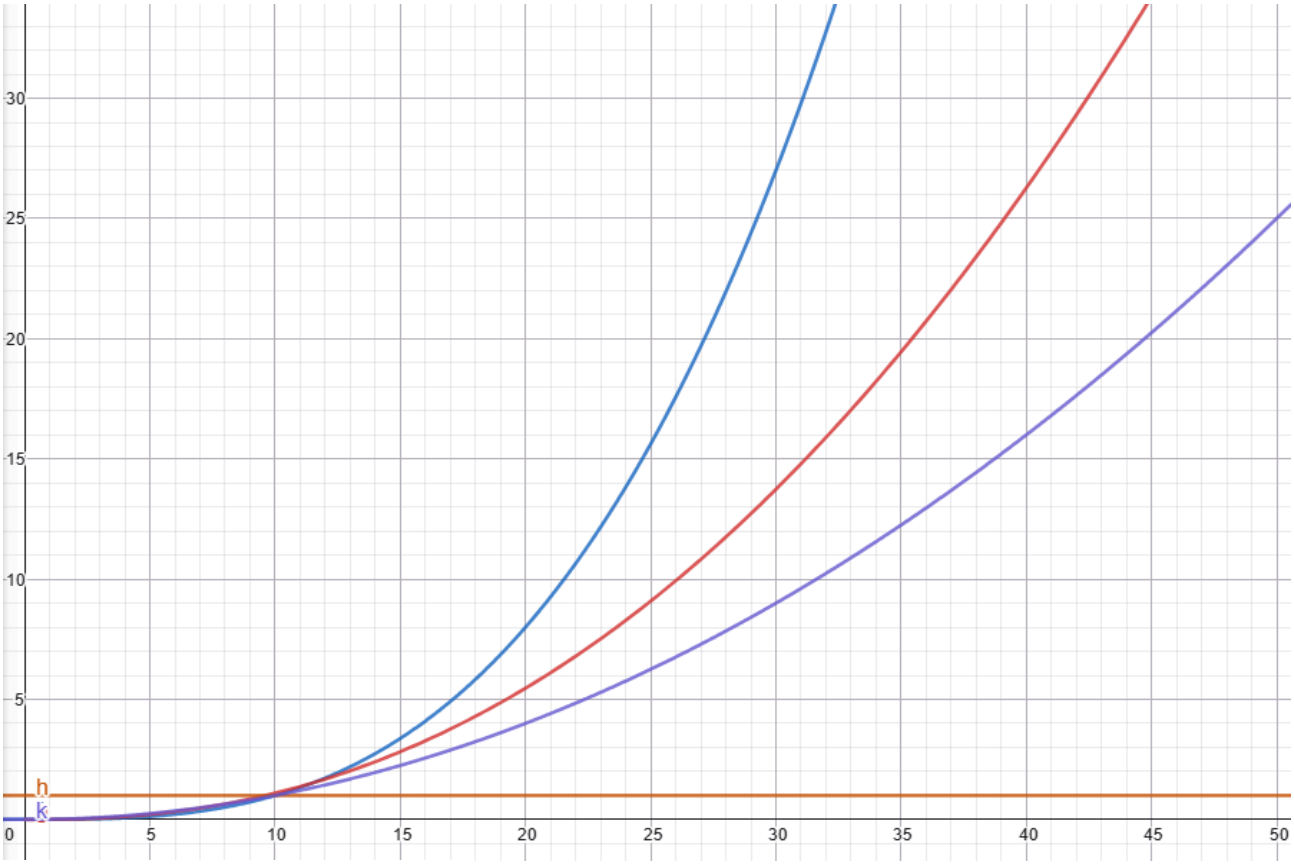
Complejidad computacional del algoritmo Divide y Vencerás

La complejidad temporal fue determinada como $O((G \cdot R+C) \cdot \log(G \cdot R))$ donde cada una representa las posibles combinaciones entre CPU, GPU y RAM.
Este comportamiento corresponde a un crecimiento
$O((G \cdot R+C) \cdot \log(G \cdot R)) \approx O(n^2) \log n)$
La cual se está aproximando a un crecimiento cuadrático con un factor logarítmico.



La complejidad espacial es $O(G \cdot R) \approx O(n^2)$ esto significa que crece de forma cuadrática

Fuerza bruta Vs Divide y Vencerás



Complejidad temporal FB=O(n^3)

Complejidad temporal DyV=O(n^2)log n)

Complejidad espacial O(1)

Complejidad espacial O(n^2)

Como se puede observar, al unir ambos algoritmos en una sola gráfica. El método de Fuerza Bruta incrementa en tiempo mucho más que el de Divide y vencerás.

Esto significa que Fuerza bruta muestra un crecimiento desproporcionado y solo será viable en problemas pequeños.

Mientras que el algoritmo divide y vencerás presenta un crecimiento más controlado, lo que lo convierte en una mejor alternativa para cuando se requiera eficiencia y escalabilidad. En mayor proporción de datos.

En conclusión divide y vencerás usa más memoria temporal ($O(n^2)$), ya que necesita guardar resultados intermedios, pero ese costo espacial se compensa con la reducción en tiempo y lo hace la mejor opción en este Algoritmo.

Conclusiones Personales

Gutierrez Vazquez Axel
Durante el desarrollo de este proyecto se logró comprender con mayor claridad el funcionamiento del algoritmo Divide y Vencerás, al aplicarlo en la descomposición del problema en etapas independientes, como la búsqueda del juego, la obtención de sus requisitos y la optimización de la build de hardware. Además, se observó que este enfoque permite un procesamiento más rápido y eficiente, al emplear métodos de búsqueda y cálculo más optimizados en comparación con la técnica de fuerza bruta.

Hernandez Macias Axxel Gael
El desarrollo de esta actividad me aportó demasiado en mi conocimiento sobre algoritmos, él como fuerza bruta garantiza explorar todas las combinaciones posibles, su costo temporal es muy alto y la adaptación de Fuerza Bruta a Divide y vencerás realmente es interesante cómo se pueden agilizar procesos de manera tan notoria como lo fue en la complejidad computacional que nos demuestra como en el peor de los casos el dividir por subproblemas compensando con mayor memoria optimiza el procesamiento de los datos. Fue un reto comprender a detalle este estudio pero el algoritmo que escogimos ayudó a que fuera interesante y despertara la curiosidad.

Quintero Arreola Laura Vanessa
Esta actividad fue muy útil para saber cómo implementar la técnica de divide y vencerás en un código que funcionaba de forma distinta, si bien ya había trabajado con esta técnica en algoritmos de ordenamiento, buscarle una aplicación para un tema de nuestro interés si fue un reto, pero al final pudimos lograr adaptar el código, además de que aprendimos a como usar otra librería para hacer GUIs, aunque de todas formas me sigue gustando mas usar tkinter para hacer interfaces, quizá porque ya estoy más familiarizada con esa libreria.

Bibliografía

- ¿Qué es y para qué sirve la librería Python Request? (2022, 9 junio). Tokioschool. Recuperado 25 de octubre de 2025, de https://www.tokioschool.com/noticias/python-request/
- ¿Es importante un diseño de hardware adecuado? (s. f.). Quora. Recuperado 25 de octubre de 2025, de https://www.quora.com/Is-proper-hardware-design-important-Why
- IBM. (9 de abril de 2024). API. Ibm.com. https://www.ibm.com/mx-es/think/topics/api
- Pozo Ramos, L. (2025). Python Scouts. Pythonscouts.com. https://pythonscouts.com/iniciando-con-pyside6-python/
- Study.com. (s. f.). *Divide-and-Conquer Algorithm: Definition, Approach & Example*. Recuperado de https://www.study.com/academy/lesson/solving-divide-and-conquer-recurrences.html