Algoritmo para buscar la mejor opción de hardware para un juego

Análisis de Algoritmos

Tema: Divide y Venceras

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

Introducción

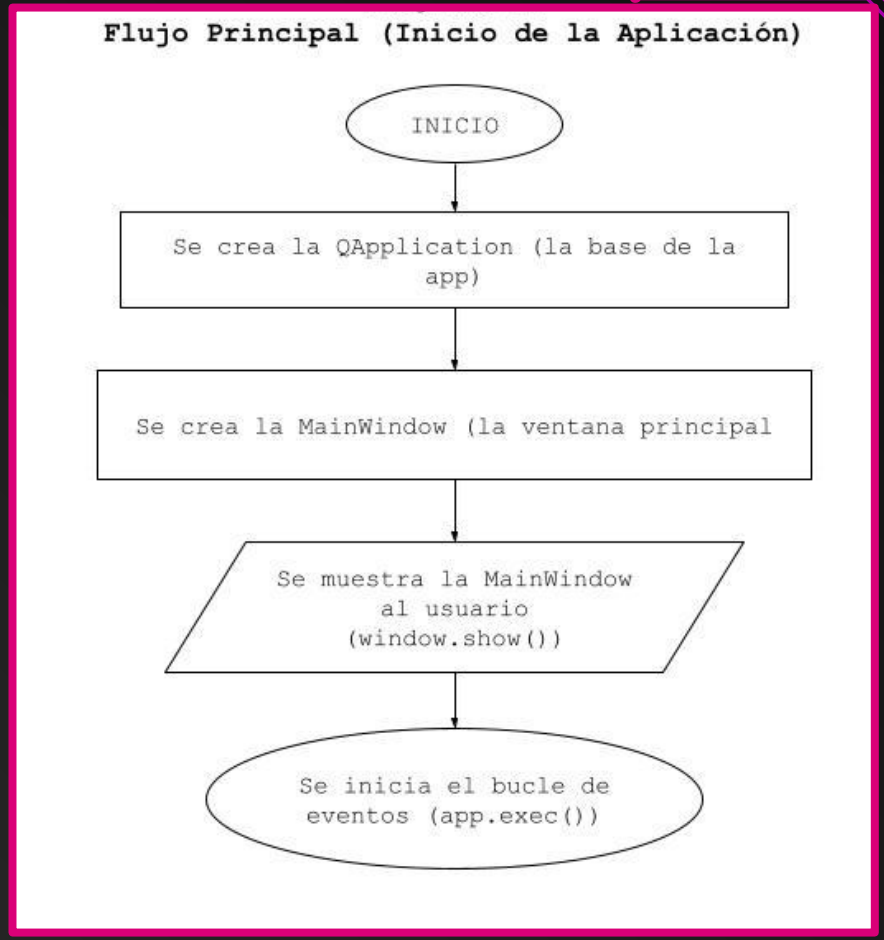
Se presenta un algoritmo diseñado para solucionar el problema común de los ”cuellos de botella” en el ensamblaje de PC, donde una mala combinación de componentes resulta en un gasto ineficiente del presupuesto y una pérdida de rendimiento. El objetivo principal es guiar a los usuarios recomendando una configuración óptima. Para lograrlo la herramienta analiza los requisitos de un juego de Steam y el presupuesto del usuario, aplicando un algoritmo de optimización (basado en búsqueda binaria) para maximizar el rendimiento. El programa utiliza QThread para mantener la fluidez de la interfaz durante las consultas a la API. Cabe destacar que el alcance del proyecto se limita a estos tres componentes: GPU, CPU y RAM, además utiliza una base de datos estática de precios y scores de rendimiento, por lo que las combinaciones se limitan a estos datos.

Justificacion

El proyecto simplifica la abrumadora tarea de elegir hardware, un proceso donde los errores (como los ”cuellos de botella”) cuestan dinero real al usuario. Al automatizar las recomendaciones basándose en los requisitos de juegos y un presupuesto, la herramienta actúa como un experto accesible, asegurando que los usuarios maximicen el rendimiento de su inversión y eviten compras ineficientes.

DIAGRAMA 1

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.



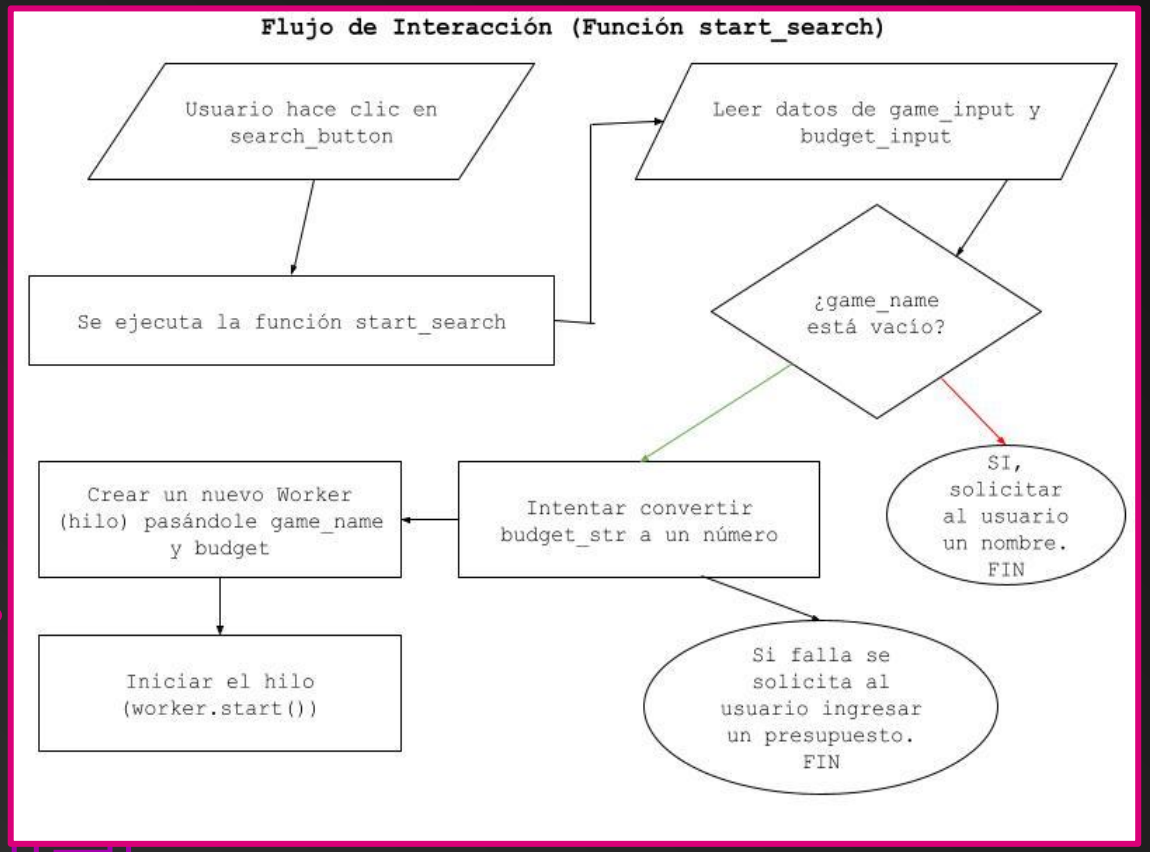
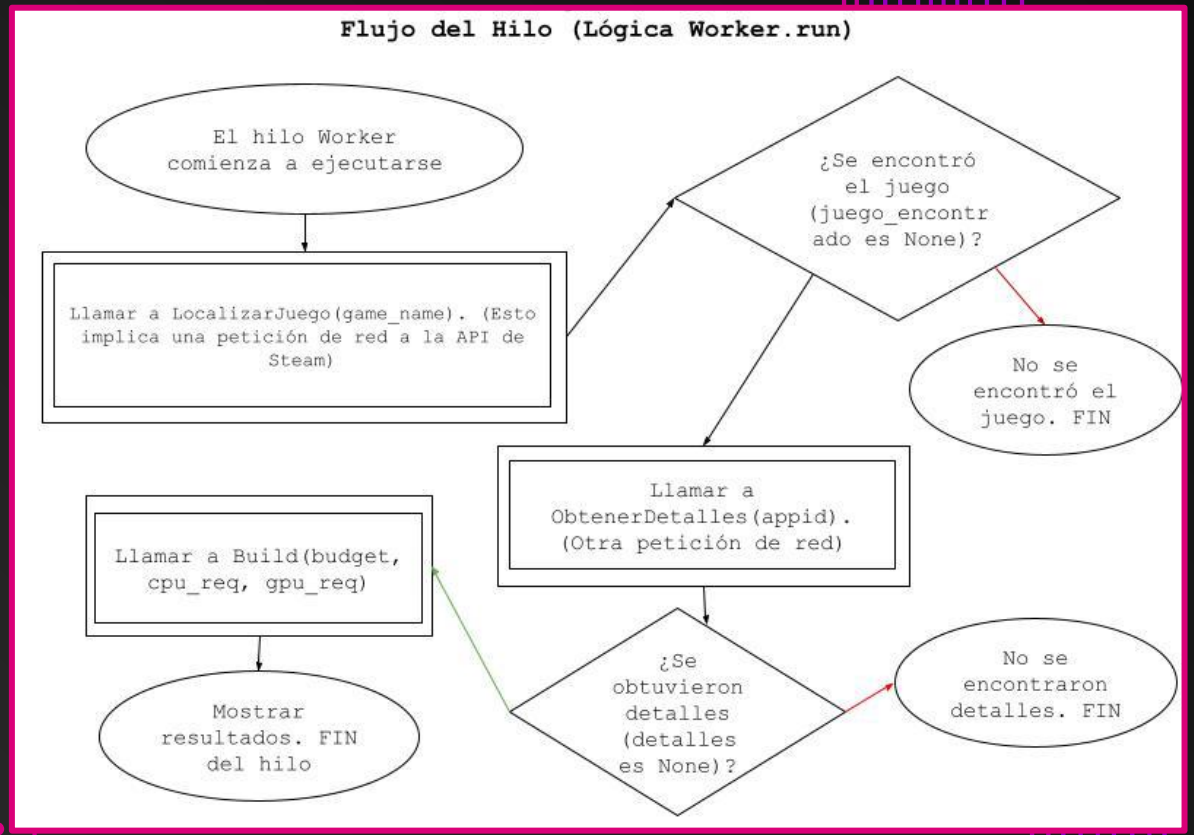


DIAGRAMA 2

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

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.



Resultados

Recomendador de Builds (Steam)

Recomendador de Builds

Juego: Silk Song

Presupuesto: 4756

Buscar

¡Juego encontrado!

Hollow Knight: Silksong

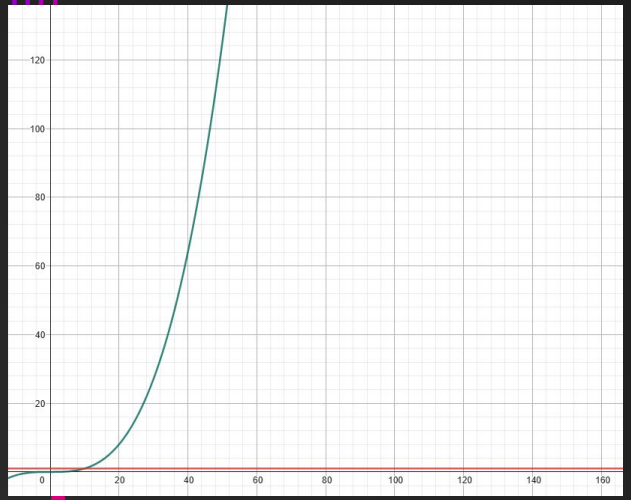
Precio: Mex$ 227.99

Requisitos Mínimos:

Mínimo:

- Requiere un procesador y un sistema operativo de 64 bits
- SO: Windows 10 version 21H1 (build 19043) or newer
- Procesador: Intel Core i3-3240, AMD FX-4300
- Memoria: 4 GB de RAM
- Gráficos: GeForce GTX 560 Ti (1GB), Radeon HD 7750 (1GB)
- DirectX: Versión 10

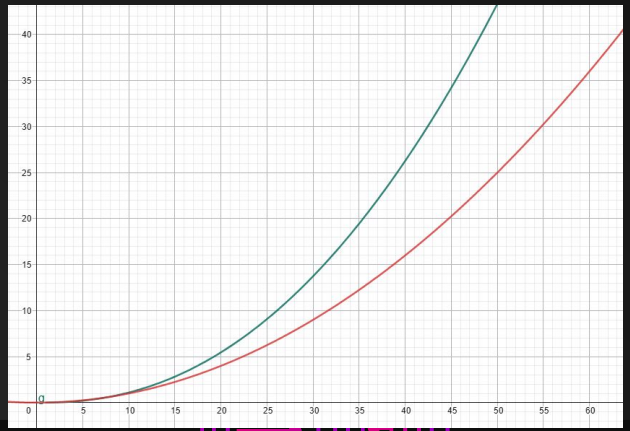
Complejidades computacionales



Divide y Venceras

$O((G \cdot R + C) \cdot \log(G \cdot R)) \approx O(n^2) \log n$

La complejidad espacial es $O(G \cdot R) \approx O(n^2)$



Fuerza Bruta

$O(c \cdot g \cdot r) \approx O(n^3)$

La complejidad espacial es $O(1)$

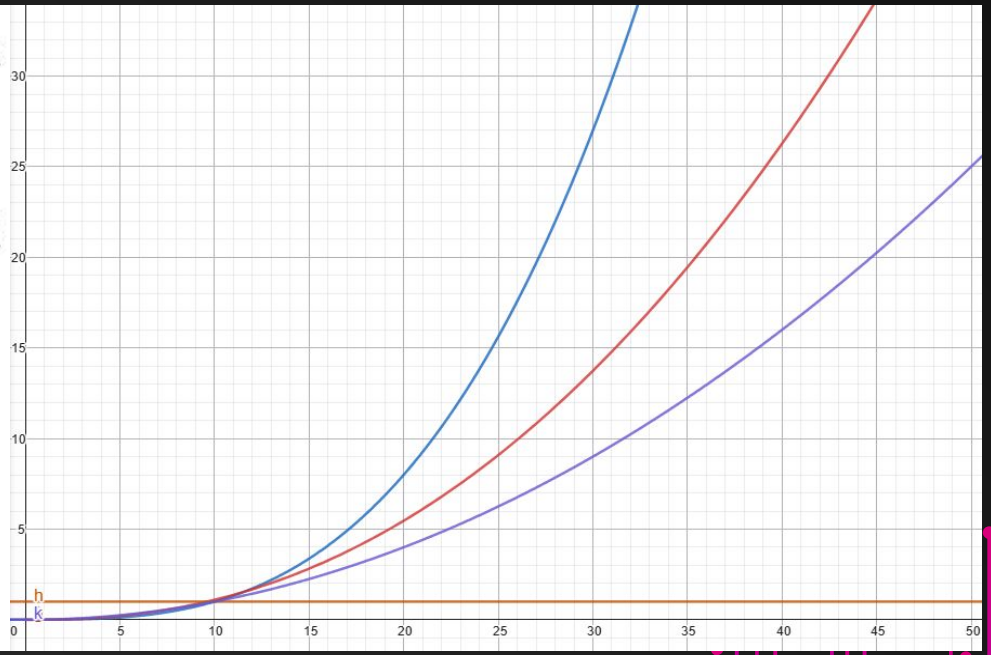
Fuerza Bruta vs Divide y Vencerás

Complejidad temporal
FB=O(n^3)

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

Complejidad espacial
O(1)

Complejidad espacial
O(n^2)



Conclusión

Se logró desarrollar un programa que resuelve el problema de los ”cuellos de botella” al ensamblar una PC, integrando la API de Steam y aplicando un algoritmo eficiente de búsqueda binaria para optimizar las recomendaciones. Se comprendió la importancia crítica de la optimización algorítmica (bisect) y la concurrencia (QThread) para garantizar un software receptivo y rápido.

Se podría expandir la funcionalidad reemplazando la base de datos estática por una fuente de precios en tiempo real y ampliando las recomendaciones para incluir componentes adicionales como la placa madre y la fuente de poder, ofreciendo así una solución de ensamblaje completa.

Gracias por su atención !