**Reporte: Práctica de Laboratorio 4**

| Victoria Rodríguez de León  *Ingeniería en Robótica y Sistemas Digitales*  *Instituto Tecnológico y de Estudios Superiores de Monterrey*  CDMX, México  [A01656328@tec.mx](mailto:A01661890@tec.mx) | Israel Macías Santana  *Ingeniería en Robótica y Sistemas Digitales*  *Instituto Tecnológico y de Estudios Superiores de Monterrey*  CDMX, México  [A01027029@tec.mx](mailto:A01661649@tec.mx) | Rodríguez Alanis Lisa Valeria  *Ingeniería en Robótica y Sistemas Digitales*  *Instituto Tecnológico y de Estudios Superiores de Monterrey*  CDMX, México  [A01656306@tec.mx](mailto:A01656306@tec.mx) |
| --- | --- | --- |

***Abstract —***

***El uso de PWM para el control de motores así como el uso de diferentes métodos de medición y tratamiento de los datos es fundamental en la industria para diferentes aplicaciones. En el presente estudio se busca realizar un control del sentido de giro de un motor de corriente directa por medio de señales enviadas por la NUCLEO H745ZI-Q y el posterior acoplamiento de un MPU6050 para la toma de muestras en 3 ejes a partir del funcionamiento del motor en distintos ciclos útiles. Se realiza esto para posteriormente utilizar diferentes algoritmos de estructuras de datos con la finalidad de realizar un análisis de eficiencia de los mismos. Los métodos utilizados demuestran una ejecución exitosa validada a través del hardware empleado así como en los valores obtenidos. A través de este estudio se ponen en práctica competencias clave en uso de actuadores y ejecución de algoritmos que serán clave para futuros desafíos.***

*Keywords* — STM32CubeIDE, NUCLEO H745ZI-Q, Motor DC, MPU6050, L298, LED, PWM, Duty Cycle

I. INTRODUCCIÓN

El avance tecnológico en la industria moderna ha llevado a una creciente dependencia de sistemas embebidos para automatizar y optimizar procesos. Uno de los componentes más críticos en estos sistemas es el motor como actuador base, cuyo control preciso es vital para una amplia gama de aplicaciones, desde la manufactura hasta la robótica y la logística. Este estudio se centra en el desarrollo de una práctica para abordar esta brecha por medio del uso de la tarjeta NUCLEO H745ZI-Q que cumplirá el rol de enviar señales de modulación de ancho de pulso (PWM) para el control de un motor DC. Para la comprobación de la adecuada generación de PWM, se utilizará un LED como actuador de referencia. Además, se abordará el uso del sensor MPU6050 para la adquisición de muestras en los 3 grados de libertad para posteriormente aplicar algoritmos de ordenamiento (Selection Sort, Bubble Sort, Insertion Sort, Merge Sort, Quick Sort) y de búsqueda (Binary Search, Linear Search) para el análisis de los datos.

1. MARCO TEÓRICO

I2C (Inter-Integrated Circuit), es un protocolo de comunicación serial que se utiliza para conectar múltiples dispositivos de bajo ancho de banda como sensores, microcontroladores y otros periféricos a un microprocesador. Su funcionamiento se basa en dos líneas bidireccionales para la comunicación de datos llamadas SDA (Serial Data) y SCL (Serial Clock). SDA se encarga de la transferencia de datos entre el maestro y el esclavo. SCL transporta la señal del reloj que sincroniza dicha transferencia. Cada bit de datos transferido en la línea SDA se sincroniza mediante un pulso alto o bajo de la línea SCL (Ajitgupta, 2021).

Según los protocolos 12C, la línea de datos no puede cambiar cuando la línea del reloj está alta. Sólo puede cambiar cuando la línea del reloj está baja. Las dos líneas son de drenaje abierto, por lo que se requiere una resistencia en pull-up para que las líneas estén altas ya que los dispositivos están activos en nivel bajo.

Los datos se transmiten en forma de paquetes de 9 bits. La secuencia de estos bits es:

* Condición de inicio: 1 bit
* Dirección de esclavo: 8 bits
* Confirmación: 1 bit

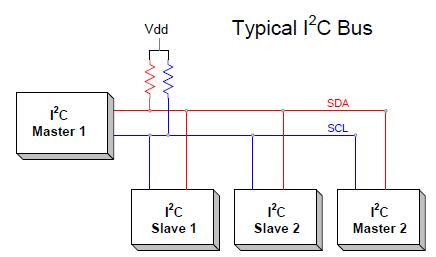
También se tienen condiciones de inicio y de parada, conocidas como START y STOP. Esto se puede realizar manteniendo alta la línea SCL y cambiando el nivel de SDA, esto se clarifica de mejor manera a continuación:

* START: SDA HIGH ->LOW, SCL HIGH
* STOP: SDA LOW -> HIGH, SCL HIGH

Entre cada par de condiciones de inicio y parada, el bus se considera ocupado y ningún maestro puede tomar control del bus. La trama de dirección es la primera trama después de bit de inicio. La dirección del esclavo con la que el maestro desea comunicarse es enviada y cada esclavo compara su propia dirección con la enviada y envía ACK. También se tienen los bits de lectura/escritura, que permiten conocer si el maestro está enviando datos al esclavo o recibiendo datos del mismo.

Después de cada trama de datos, sigue un bit ACK/NACK. Si la trama de datos se recibe correctamente, el receptor envía el bit ACK al remitente.  
 Respecto al formato de comunicación, los datos se transmiten en forma de paquetes de 9 bits de largo, de los cuales los primeros 8 se colocan en SDA y el último está reservado para ACK/NACK. La condición de START más el paquete de dirección más un paquete de datos más la condición de STOP forman colectivamente una transferencia de datos completa.

La mayoría de los dispositivos I2C pueden comunicarse a 100 kHz o 400 kHz. Algunas desventajas de este protocolo es la velocidad y distancia limitada a comparación de otros protocolos de comunicación como ISP o UART; sin embargo, se puede decir que combina las mejores características de SPI y UART lo que lo vuelve un modelo muy rentable.



*Figura 1. Funcionamiento de I2C* [2]

1. **Motor DC**

Un motor de corriente continua (motor DC), es un tipo de motor eléctrico que funciona con corriente continua. Esto quiere decir, que la dirección del flujo de corriente eléctrica es constante. Los motores DC se componen principalmente de dos partes: un rotor (la parte giratoria) y un estator (la parte fija). La velocidad y la dirección de giro del motor puede controlarse variando la polaridad de la corriente eléctrica suministrada al rotor.

Este tipo de motores tienen varias ventajas, principalmente la de controlar la velocidad y dirección de giro de manera precisa. Sin embargo, generalmente se combina con sistemas de control electrónico más sofisticados para lograr un rendimiento óptico.

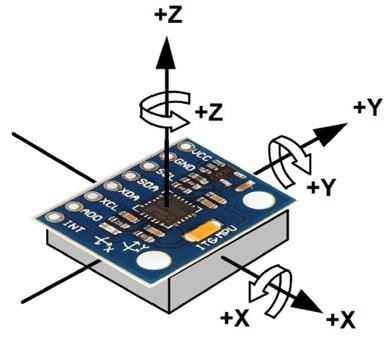
1. **Puente H L298**

Un puente H, es un dispositivo electrónico capaz de controlar motores de corriente directa (DC) o motores paso a paso. Este circuito permite controlar la dirección de flujo de corriente en un motor, determinando la dirección de rotación. El puente H L298 es un CI (Circuito Integrado) de puente H doble, lo que significa que puede controlar dos motores de corriente directa de manera independiente. Está diseñado para trabajar con motores que requieren una corriente más alta y ofrece protección contra sobrecorriente.

Este puente H, generalmente se usa con un microcontrolador o una placa de desarrollo para controlar la dirección y velocidad del motor. Los pines del L298 se conectan a los terminales del motor y se utiliza una lógica de control para aplicar tensiones adecuadas a los terminales del motor. Esto permite que el motor gire en la dirección deseada y a la velocidad requerida.

1. **MPU6050 (IMU)**

La Unidad de Medición Inercial, también conocida como IMU, es un dispositivo electrónico que combina un acelerómetro de 3 ejes y un giroscopio de 3 ejes para ofrecer 6 grados de libertad. Con el acelerómetro, podemos medir la aceleración lineal, es decir, el cambio de velocidad en el tiempo a lo largo de los ejes X, Y, Z. Con el giroscopio se puede medir la velocidad angular alrededor de esos mismos ejes, o en otras palabras, la tasa de cambio del ángulo. La comunicación del dispositivo es por I2C (Barak, 2021). En esta práctica se utilizó el modelo MPU6050.



**Figura 3.** Sensor IMU [3]

1. **Algoritmos de ordenamiento**

*1. Bubble Sort*

El ordenamiento de burbuja, también conocido como *bubble sort,* es un algoritmo simple que funciona iterando a través de una lista de elementos y realizando una comparación por pares. Si el elemento en la posición “x” es mauor que el elemento a su derecha (es decir, “x+1”), se intercambian. Esto se repite hasta que ya no se requieran más intercambios (Kolade, 2022). La complejidad temporal de este algoritmo es de para el peor de los casos y el caso promedio. Para el mejor de los casos es de

*2. Selection Sort*

El ordenamiento por selección, también conocido como *selection sort,* es considerablemente simple, ya que divide la lista en dos partes: los elementos ya ordenados (de izquierda a derecha) y los elementos que faltan por ordenar. De esta manera, busca el elemento de la lista no ordenada e intercambia ese elemento mínimo con el primer elemento de la misma, moviéndolo así a su posición correcta en la lista ordenada. Sigue este proceso hasta que la lista no ordenada esté vacía (Brihadiswaran, 2021). La complejidad temporal de este algoritmo es de en todos los casos. No es recomendable para listas grandes en donde el uso de recursos temporales es alto.

*3. Insertion Sort*

El ordenamiento por inserción, también conocido como *insertion sort,* es un algoritmo que funciona construyendo una lista ordenada elemento por elemento. A diferencia de *bubble sort y selection sort,* tiene un mejor rendimiento en ciertos escenarios, especialmente cuando la lista está parcialmente ordenada. El algoritmo funciona comparando el elemento actual con los elementos anteriores de la lista ordenada. Lo compara y si el elemento actual es menor al elemento con el que se está comparando, se desplaza el elemento más grande una posición a la derecha. Repite este proceso hasta encontrar la posición correcta para el elemento actual. La complejidad temporal de este algoritmo es de en el peor de los casos el caso promedio, pero mejora a en el mejor caso, que es cuando la lista ya está ordenada.

*4. Merge Sort*

El ordenamiento por mezcla, también conocido como *merge sort,* es un algoritmo que funciona basado en un enfoque “divide y conquista”, también conocido como *divide and conquer.* Su principio es dividir la lista no ordenada en n sub-listas, cada una conteniendo un elemento. Después se combinan repetidamente las sub-listas para producir nuevas listas ordenadas hasta que sólo quede una sub-lista. Este proceso se repite hasta que todas las sub-listas se combinan en una lista completamente ordenada. Su complejidad temporal es para todos los casos, lo cual vuelve a este algoritmo significativamente más rápido que el resto. Una desventaja sería que requiere memoria adicional (Cormen, Leiserson et al., 2009).

*5. Quick sort*

El ordenamiento por sorteo rápido, también conocido como *quick sort,* es un algoritmo que usa el mismo enfoque de *divide and conquer* que tiene el *merge sort*, sin embargo, realiza la ordenación “in situ”, lo que significa que no requiere memoria adicional significativa. El algoritmo se basa en una selección de pivote, que es un elemento de la lista, y esto hace que se divida en partes no iguales (a diferencia del *merge sort*). Después realiza la partición reorganizando la lista de manera que todos los elementos menores que el pivote queden a su izquierda y los elementos mayores queden a la derecha. Esto se aplica de manera recursiva a las 2 sub-listas hasta que quede completamente ordenada la lista original. La eficiencia de este tipo de algoritmo depende mucho de la elección del pivote. En el mejor de los casos y el caso promedio, la complejidad temporal es de sin embargo, en el peor de los casos, puede ser de lo cual no es ideal (Cormen, Leiserson et al., 2009).

1. **Algoritmos de búsqueda**

Para los algoritmos de búsqueda, se utilizaron dos para esta práctica: Binary Search y Linear Search. Se explican a continuación:

1. *Linear Search*

Este es un algoritmo de búsqueda muy simple ya que su funcionamiento general es recorrer la lista elemento por elemento hasta encontrar el valor deseado. Inicia con el primer elemento de la lista. Si no se encuentra el valor, retorna un valor que indica que no se encuentra, usualmente es -1. La complejidad temporal de este algoritmo es de en el peor de los casos, que es cuando el elemento está al final de la lista o no se encuentra. En el mejor de los casos (el elemento está en la primera posición), la complejidad es de

1. *Binary Search*

Este es un algoritmo de búsqueda mucho más eficiente que el *linear search*. Funciona dividiendo repetidamente a la mitad la posición de la lista que podría contener el elemento y después lo compara con el elemento del valor objetivo. La complejidad temporal de este algoritmo es de lo que lo hace muy eficiente para listas grandes siempre que estén ordenadas. Para el mejor de los casos, que es que el valor esté justo en el medio, la complejidad temporal es de

1. METODOLOGÍA

Para esta práctica el material y equipo necesario es:

1. Una computadora.
2. Una tarjeta de desarrollo Núcleo H745ZI-Q
3. Un cable microUSB - USB A
4. Un monitor serial
5. Una pantalla LCD 16\*2
6. Un LED en configuración pull-up
7. Un motor de corriente directa RS390
8. Un puente H L298N
9. Un sensor MPU6050

A lo largo de este apartado se va a abordar la explicación y elaboración de los ejercicios resueltos en la práctica. Los 4 ejercicios implican el uso de *timers* para la generación de PWM y la integración de la comunicación de sensores por medio del protocolo I2C, así cómo el envío de información por comunicación serial usando el protocolo USART. Además, en el último ejercicio se hace un análisis a profundidad de la complejidad en tiempo y espacio que tienen los algoritmos de ordenamiento y búsqueda aplicados en microcontroladores. En cada uno de los ejercicios se emplea al menos un *timer* interno de la tarjeta para generar una salida de PWM. La configuración general de cada uno de los *timers* está dada por los valores que se le dan al *prescaler* y *counter period* dentro de la pestaña de *Pinout & Configuration* del entorno de configuración. El *prescaler* indica el tamaño del paso o la frecuencia de muestreo que tiene el *timer* seleccionado, y el *counter period* hace referencia al tiempo de corte que va a tener ese *timer* o a su frecuencia total. En esta práctica se hizo uso únicamente de 3 timers para los 4 ejercicios. El cálculo del valor del *prescaler* está dado por las ecuaciones 1 y 2  
 … (1)

… (2)

donde la frecuencia de conteo del timer (*ftimx*)está dada por la relación entre el prescaler general que tiene el canal APBx (*fAPBx*) y el valor del *prescaler* personalizado (*PSC*).  
 Para los ejercicios 1, 2 y 3 se usó el *timer 2* de 32 bits para realizar la generación de PWM por el pin de salida del canal 3 que se encuentra en el pin 32 del CN10 de la tarjeta. Para el ejercicio 3 se añadieron los *timers* 3 y 13 de 16 bits para el control del conteo de tiempo de recopilación y envío de datos.

En los ejercicios 3 y 4 se utilizó el protocolo I2C para recibir datos del sensor MPU6050 y mandar información a una pantalla LCD de 16\*2. Para ambos ejercicios se usó el puerto 4 de I2C que ofrece la tarjeta para conectar los pines de SDA y SCL de cada conexión. El pin de SCL y SDA del puerto 4 se encuentran sobre el puerto del CN9 de la tarjeta en los pines 19 y 21 respectivamente. La configuración del puerto de I2C dentro del entorno gráfico de configuración se dejó por *default.* **Funciones para Timers**  
La función necesaria dentro del código para inicializar el conteo de un *Timer* es *HAL\_TIM\_Base\_Start,* la cual inicializa los parámetros base que se generaron cuando se selecciona la configuración del *timer* seleccionado*.* La función recibe como parámetro el apuntador al handler de inicio del *timer*, dónde se encuentran las variables configuradas previamente.  
 La función *HAL\_TIM\_Get\_Counter,* obtiene el valor del registro de conteo del *timer* en el tiempo de ejecución y lo retorna. Esto significa que va a devolver el valor que se encuentre en el registro del *timer* en el momento exacto en que se manda a llamar. El parámetro que recibe es el apuntador al handler del *timer* que se seleccionó.

La función*HAL\_TIM\_PeriodElapsed\_Callback( )* junto con la función *HAL\_TIM\_Base\_Start\_IT***,** permiten el funcionamiento de los *Timers* por interrupción. Al igual que la función de *HAL\_TIM\_Base\_Start( ),* la función HAL\_TIM\_Base\_Start\_IT sirve para inicializar el conteo del timer y además habilitar su funcionamiento por interrupción. De forma que una vez que su conteo llegó al final, la función *HAL\_TIM\_PeriodElapsed\_Callback( )* ejecuta el código en su interior cuando se activa la interrupción por *Timer*.  
 **Funciones para protocolo USART**

La función que permite transmitir datos por medio del protocolo USART es*HAL\_UART\_Transmit,*la cual recibe 4 parámetros: un puntero al *handler* con las variables de inicialización del protocolo, (como el valor del *baud rate*, la paridad, si tiene *oversampling*, el tamaño del *buffer* de transmisión y recepción, entre otras), un puntero al buffer de transmisión, la cantidad de elementos que se van a enviar, y el tiempo de duración o *timeout.*

La función para recibir por interrupción, al igual que en el caso de los *timers* se hace por medio de la función correspondiente con terminación IT, que, además de recibir, lo hace por medio de interrupción. La función *HAL\_UART\_Receive\_IT* permite la recepción de datos cada vez que la interrupción por recepción se active. Una vez que se detecta este proceso, se pueden llevar a cabo acciones para tratar la información que se recibió. Para eso está la función *HAL\_UART\_RxCpltCallback,*la cual permite la ejecución y proceso de la información una vez que la trama de información que se recibió está completa.  
 **Funciones para la LCD vía I2C**

Las funciones para utilizar la pantalla LCD por medio de I2C vienen dadas en las librerías de los archivos  *i2c-lcd.c* e *i2c-lcd.h*. Las funciones que ya vienen definidas en los archivos son las siguientes:

* *lcd\_init( )*: inicializa el display
* *lcd\_clear( ):* pone el display en blanco
* *lcd\_put\_cur( )*: cambia el cursor de lugar
* *lcd\_send\_cmd( ):* envía información a un registro específico
* *lcd\_send\_data ( ):*  envía un carácter a la vez
* *lcd\_ send\_ string( ):* envia un string

**Funciones para enviar y recibir datos por I2C**La función *HAL\_I2C\_Mem\_Read()* lee la información proveniente del dispositivo externo dada una dirección de memoria específica. La función *HAL\_I2C\_Mem\_Write( )* manda una trama de datos al dispositivo conectado dada una dirección de memoria específica. Ambas funciones reciben 6 parámetros que sirven para identificar el puerto de I2C que se usa en la tarjeta, el dispositivo que se está leyendo, el tamaño de la trama que se va a recibir y la dirección de memoria dónde se va a guardar la información.

Los definición de los parámetros específicos para ambas funciones son los siguientes:

* **&h12c**: Apuntador a la estructura I2C\_HadleTypeDef.
* **DevAddress:** Dirección del dispositivo de destino.
* **MemAddress:** Dirección de la memoria interna.
* **MemAddSize:** Tamaño de la dirección interna de memoria.
* **pData:** Apuntador al *buffer* de información.
* **Size:** Cantidad de información a mandar.
* **Timeout:** Duración del tiempo de respuesta.

**Ejercicio 1: Control de PWM con un LED**El objetivo de este ejercicio es controlar la modulación del ancho de pulso de una señal y verlo reflejado en la intensidad de luz de un LED. Para ello se configuró el *timer 2* en modalidad de PWM para generar una salida de 10 KHz utilizando un frecuencia de muestreo de 1µs. Dentro de la configuración se habilita el canal 3 cómo salida de PWM y se habilita la opción para que el *timer* funcione con el reloj interno del núcleo. El valor del *prescaler* para que el paso sea de 1µs tiene que ser de 74 y el valor del *counter period* para que la frecuencia sea de 10 kHz tiene que ser de 1000. Ambos valores se obtienen utilizando las ecuaciones 1 y 2.

Para que la señal de PWM se vea reflejada en el LED se tiene que inicializar la función del *timer* que permite la generación de la señal de PWM con el *handler* del *timer 2* y la especificación del canal a usar.Después se conecta la salida física del pin con el LED en configuración *pull-up.*

Dentro del código, en el ciclo infinito de la función *main* se tiene que hacer un programa que permita cambiar el ciclo útil de la señal de forma ascendente hasta llegar al 100% del ciclo útil y que descienda de nuevo a 0% en un lapso de 200ms. Para esto se tiene que especificar el timer, el canal por el que se manda el PWM y el ciclo útil de la señal con un apuntador al registro de captura del timer que guarda el valor del *counter period.* Para asegurar que la duración de la señal sea de 2ms se añaden dos retardos al finalizar el envío de señal. La rutina del ejercicio se encuentra en el diagrama de flujo del Anexo 1. El video demostrativo se encuentra en el enlace del Anexo 0.

**Ejercicio 2: Control de PWM con un Motor de corriente continua**

El objetivo de este ejercicio es controlar la modulación del ancho de pulso de una señal y verlo reflejado en la velocidad de un motor DC, dónde el resultado esperado es observar el aumento paulatino de su velocidad hasta que alcance un 100% de ciclo útil. Para ello se configuró el *timer 2* en modalidad de PWM para generar una salida de 10 KHz utilizando un frecuencia de muestreo de 1µs. Dentro de la configuración se habilita el canal 3 cómo salida de PWM y se habilita la opción para que el *timer* funcione con el reloj interno del núcleo. El valor del *prescaler* para que el paso sea de 1µs tiene que ser de 74 y el valor del *counter period* para que la frecuencia sea de 10 kHz tiene que ser de 1000. Ambos valores se obtienen utilizando las ecuaciones 1 y 2.

Para que la señal de PWM se vea reflejada en la velocidad del motor se tiene que inicializar la función del *timer* que permite la generación de la señal de PWM con el *handler* del *timer 2* y la especificación del canal a usar.Después se conecta la salida física del pin hacia el pin de habilitación del puente H para controlar el flujo de corriente hacía el motor, y mandar la señal de PWM incremental.

Dentro del código, en el ciclo infinito de la función *main* se tiene que hacer un programa que permita cambiar el ciclo útil de la señal de forma ascendente hasta llegar al 100% del ciclo útil y que descienda de nuevo a 0% en un lapso de 2s. Para esto se tiene que especificar el timer, el canal por el que se manda el PWM y el ciclo útil de la señal con un apuntador al registro de captura del timer que guarda el valor del *counter period.* Para asegurar que la duración de la señal sea de 2s se añaden dos retardos al finalizar el envío de señal. La rutina del ejercicio se encuentra en el diagrama de flujo del Anexo 2. El video demostrativo se encuentra en el enlace del Anexo 0.

**Ejercicio 3: Muestreo de velocidad angular en motor DC con PWM de 25%, 50% y 75% de ciclo útil.**

El objetivo de este ejercicio es controlar la modulación del ancho de pulso de una señal y verlo reflejado en la velocidad de un motor DC esperando observar tres velocidades angulares del motor dependiendo del ciclo útil que se envíe. Además, con el fin de analizar las vibraciones y las frecuencias de cada señal, se acopló un sensor IMU MPU6050 al casco del motor para registrar los datos de la velocidad angular por medio del giroscopio integrado. El giroscopio tiene una frecuencia máxima de salida de 8k Hz la cual permite obtener un máximo de 8000 muestras. Sin embargo, para usos prácticos del ejercicio, la cantidad máxima de muestras se va a establecer en 2048 datos por segundo, con divisiones de 1024 y 512 muestras para distinguir sus diferencias en el ejercicio 4 posterior. Estas muestras se guardaron en listas dependiendo el ciclo útil del PWM aplicado, y la perturbación generada en el ambiente. Al final se guardaron 18 listas de 512 elementos con los valores de los ejes X, Y, y Z otorgados por el giroscopio. Estos datos se almacenaron en un archivo externo con extensión *header* para su análisis posterior en el ejercicio 4.

Para la muestra de los 2048, 1024 y 512 datos se configuró el *timer 3*  de 16 bits y se habilitó su función de interrupción por *timer.* La función de término de conteo por medio de *callback* se va a llamar cada 488 µs y su frecuencia de muestreo va a ser de 1µs. Estos valores garantizan la obtención de 2049 datos en un segundo con un error de recopilación mínimo. La activación del conteo de este *timer* dentro del programa se hace desde la sección del código que permite escribir al usuario la inicialización de funciones y variables locales añadiendo la función de inicialización del *timer* antes de comenzar con el ciclo infinito del programa. Una vez que el timer comenzó el conteo va a pasar un segundo hasta que termine de recopilar todos los datos muestreados y los almacene en los arreglos. Es importante mencionar que por usos prácticos solo se guardaron las listas de 512 elementos. Mientras tanto, en el ciclo infinito del programa se va a estar generando una señal de PWM constante de ciclo útil variable, este ciclo va a cambiar entre un 25%, un 50% y un 75%. La recopilación de los datos se hizo únicamente con un ciclo útil para no mezclar señales. Después de haberse recopilado la información de cada uno de los casos, se mandan las listas por serial para descargarlas y añadirlas en el archivo externo.

Para el envío por serial, se habilitó el *timer13* de 16 bits para y su función de interrupción por *timer* cada 0.5 ms. El valor de *prescaler* que recibe es de 74 para que tenga un paso de 1µs y un *counter period* de 500. La función que inicializa el *timer 13* se va a activar una vez que el *timer 3,* encargado de la recopilación de datos haya concluido. La función que detiene el *timer* se manda a llamar después de haber llenado las listas de muestras. Dentro de la función que se llama por interrupción del *timer*, se usa la función de transmisión por protocolo USART que envía los datos de las listas al monitor serial. La transmisión de datos se completa en aproximadamente un segundo y después de verificar que las listas se hayan enviado completas se detiene el timer para que no siga enviando datos vacíos. La rutina del ejercicio se encuentra en el diagrama de flujo del Anexo 3. El video demostrativo se encuentra en el enlace del Anexo 0.

**Ejercicio 4: Análisis de algoritmos de ordenamiento y búsqueda.**El objetivo de este ejercicio consiste en ordenar las listas generadas en el ejercicio anterior por los siguientes algoritmos de ordenamiento:

1. Bubble sort
2. Insertion sort
3. Selection sort
4. Merge sort
5. Qui**c**k sort

Posteriormente se tienen que correr los algoritmos de búsqueda lineal y binaria sobre las listas ordenadas para encontrar el promedio de los tiempos que tardan los algoritmos en ser ordenados y en encontrar el valor buscado. Se espera que el promedio de los tiempos de los algoritmos tengan relación y cambien dependiendo su estructura y su complejidad temporal y espacial. La visualización de los promedios se va a realizar a través de la pantalla LCD 16x2. Los algoritmos se tiene que añadir cómo funciones en un zona del código editable por el usuario.   
 Para el ejercicio se configuraron los *timers 2 y 3,* en este caso, el *timer* 2 no fue usado para la generación de una señal PWM, en cambio se usó para medir los tiempos de ejecución de los algoritmos. El timer 2 de 32 bits permite medir tiempos muy grandes debido a su tamaño de palabra. El *timer* 3 fue configurado para que mande cada 0.5 ms las cadenas ordenadas por serial y verificar que los algoritmos realicen de forma correcta su función.  
 Dentro del código del programa, fuera del ciclo infinito se mandó llamar la función que inicializa el conteo del timer 2 para medir el tiempo de ejecución de los algoritmos. Seguidamente se mandan llamar las funciones de ordenamiento y búsqueda dentro de un ciclo para repetir la operación 100 veces. Con los datos guardados en un arreglo se calculó el promedio de los tiempos y se desplegó la información en la LCD. Dentro del ciclo infinito del *main* se usa la función de la librería de la LCD para mandar por I2C los valores como cadenas de caracteres.  
 La lectura de los datos se hace desde el archivo generado en el ejercicio 3 con las 18 listas que contienen las muestras de los ejes que envió el giroscopio. Este archivo se tiene que incluir como un archivo *header* en la parte superior del código.

1. RESULTADOS

**Resultados ejercicio 1: Control de PWM con un LED**Los resultados esperados para este ejercicio eran la observación del cambio en la intensidad de luz del LED a medida que el porcentaje del ciclo útil en la señal de PWM fuera aumentando. En el video del anexo 0 se muestra el resultado obtenido en el ejercicio. En este video se observa cómo en un lapso de 5 segundos la intensidad de luz aumenta paulatinamente hasta llegar a la máxima intensidad de luz posible y vuelve a 0 detenidamente. Esto se debe a la implementación de los *delays* por cada ciclo iterativo del programa que ralentizan el aumento del valor del ciclo útil enviado por el canal 3 del *timer* 2.

**Resultados ejercicio 2: Control de PWM con un Motor de corriente continua**

Los resultados esperados para este ejercicio eran la observación del cambio en la velocidad angular del motor a medida que el porcentaje del ciclo útil en la señal de PWM fuera aumentando. En el video del anexo 0 se muestra el resultado obtenido en el ejercicio. En este video se observa cómo en un lapso de 2 segundos la velocidad del motor incrementa paulatinamente hasta llegar a la velocidad máxima, se queda girando por 100 ms en esa velocidad y seguidamente comienza a desacelerar en un lapso de 2 segundos hasta quedarse quieto. Este aumento y decremento paulatino de velocidad se debe a la implementación de *delays* por cada ciclo iterativo del programa que que ralentizan el aumento del valor del ciclo útil enviado por el canal 3 del *timer* 2.

**Resultados ejercicio 3: Muestreo de velocidad angular en motor DC con PWM de 25%, 50% y 75% de ciclo útil.**Los resultados esperados para este ejercicio eran la obtención de 18 listas de 512 datos muestreados por el IMU anclado al casco del motor. Las listas contienen los datos de los diferentes ciclos útiles con los que estuvo girando el motor además de la consideración en la variación de frecuencia que había dependiendo la perturbación en el ambiente e interferencia de la señal. En el video del anexo 0 se muestra la captura de las muestras obtenidas en el ejercicio con el motor funcionando a un 25% de ciclo útil. En este video también se observa cómo enciende un LED rojo para indicar el inicio de la toma de datos y los LEDs de color verde y amarillo para indicar el envio de información al monitor serial. El LED rojo se queda prendido aproximadamente un segundo mientras que los otros dos LEDs se prenden y apagan rápidamente, indicando que la transmisión de datos se efectúa en poco más de 100ms. En el archivo *Datos\_Motor\_Ejes.h* se encuentran las 18 listas obtenidas en el ejercicio.

**Resultados ejercicio 4: Análisis de algoritmos de ordenamiento y búsqueda**

Los resultados esperados para este ejercicio son la visualización de los promedios de los tiempos de ejecución de los algoritmos de búsqueda usando cada uno de los algoritmos de ordenamiento. La visualización de los datos se tiene que mostrar en la pantalla LCD. En el video del anexo 0 se muestran los resultados obtenidos en el ejercicio.  
 A parte, en el análisis de resultados se aborda la explicación y comparación de los tiempos de ejecución de cada algoritmo, dónde se detalla la complejidad espacial y temporal de cada uno de ellos y su resalta el mejor y peor de los casos usando la notación Big O, Big Theta, y Big Omega.   
 El algoritmo de búsqueda lineal o *linear search* presenta una clara desventaja en complejidad temporal con respecto al algoritmo de búsqueda binaria, pues el recorrido que hace por toda la lista implica un retardo en la búsqueda del valor, incluso más si es un valor que se encuentra al final de la lista. En comparación, el algoritmo de búsqueda binaria presenta un promedio de tiempos similares independiente del largo de la cadena y de la posición del valor.   
 En la tabla 1 se muestran los promedios de la implementación de los algoritmos de búsqueda para cada uno de los algoritmos de ordenamiento. Los datos obtenidos en esta tabla se hicieron considerando un caso promedio donde el valor a buscar se encuentra presente en la lista. En la tabla se aprecia notablemente la diferencia entre los algoritmos recursivos y los algoritmos iterativos. Los algoritmos que funcionan bajo recursión cómo el *quick sort* o el *merge sort*, presentan una diferencia significativa contra los algoritmos que se resuelve por iteración empleando la función *swap* o cambio de posición. Este tipo de programación ayuda mucho a darle limpieza y estructura a un código, pues permite realizar operaciones simétricas o que no tienen un cambio en su ejecución tal como lo pueden ser los algoritmos de búsqueda. Sin embargo, la creación de nuevo espacio después de que las funciones se mandan llamar, implica un gran consumo de recursos en la memoria del programa, por lo que su implementación para sistemas dónde el espacio es limitado se ve reducido y muy poco útil para tratar señales de mayor longitud.

Por lo que, a pesar de que los algoritmos de recursión presentaron un mejor desempeño en el ordenamiento de los datos, en un escenario donde se tengan que muestrear señales más grandes, el uso de estos algoritmos puede presentar un problema en el agotamiento de recursos.

**Tabla 1.** Promedio de tiempos de búsqueda por algoritmo de ordenamiento

|  | *Linear search* | *Binary search* |
| --- | --- | --- |
| *Bubble sort* | *143.69 ms* | *133.2 ms* |
| *Selection sort* | *92 ms* | *79 ms* |
| *Insertion Sort* | *55.8 ms* | *45.3 ms* |
| *Merge Sort* | *20.16 ms* | *9.6 ms* |
| *Quick sort* | *16.3 ms* | *5.7 ms* |

1. CONCLUSIONES INDIVIDUALES

**Lisa Valeria Rodríguez Alanís**

En esta práctica se plantearon ejercicios muy retadores para abordar funciones esenciales en la núcleo como interrupción por timer, generación de pwm por timer y algoritmos de ordenamiento y ordenamiento. En este caso gracias a la generación de pwm por timer pudimos seguir practicando el control de giro y velocidad de un motor DC por medio de un puente H, esto nos ayudará mucho para la elaboración del reto. Por otro lado, la implementación de comunicación seria y la extracción de los datos de los ejes de giroscopio, presentaron un reto para el equipo donde desarrollamos las habilidades prácticas para juntar dicha información para su posterior análisis. Con esta información, después pudimos repasar nuestro conocimientos de programación y de algoritmos para ordenar estas listas, concluyendo que esta fue una práctica muy completa donde se juntaron temas esenciales del bloque en estos ejercicios de laboratorio

**Israel Macías Santana**

La práctica resultó muy retadora y de gran aprendizaje al poner a prueba el conocimiento recabado en prácticas pasadas. Trabajar de nuevo con *timers*, comunicación por protocolo USART, conexión de dispositivos externos por I2C y además implementar la generación de señales de PWM para controlar las velocidades y la orientación de giro del motor fueron una experiencia muy desgastante y retadora al mismo tiempo, pues la configuración de cada una de estas herramientas por medio de las hojas de datos requiere paciencia y tiempo para observar los fallos que puede haber en tu código. Igualmente me pareció muy nutritiva esta práctica ya que te ayuda a fortalecer los conocimientos previos y mejorar los esquemas y paradigmas de programación. Particularmente me gustó poder corroborar el funcionamiento correcto de cada uno de los ejercicios por medio de funciones como envío de datos por serial, o la activación de LEDs para checar el estado del programa. Además, la práctica sirve de mucho para entender la configuración de *timers* y la forma de mandar llamar secciones de código por medio de *callbacks* de interrupción.  
Incluso, la aplicación de algoritmos recursivos e iterativos para tratar los datos que se obtienen del sensor, ayuda a ver la velocidad y la capacidad que tiene la tarjeta para procesar este tipo de funciones y así conocer sus limitantes para prácticas futuras.

**Victoria Rodriguez de León**  
 Esta práctica fue muy significativa ya que representa la integración de conceptos de hardware tratados desde las prácticas pasadas (cómo usar la tarjeta *Nucleo H745ZI-Q* para realizar control de sensores y actuadores con aplicación basada en la configuración de la interfaz en *STM32CubeIDE,* modificación del código fuente con base en la interpretación de la documentación y datasheet para la implementación de funciones) hasta la implementación de algoritmos para el tratamiento de los datos. Se realizó primeramente una configuración del Timer para la generación de PWM, que va al *enable* del puente H para el control de la velocidad del motor. Para las señales de control de dirección, se configuraron los pines de GPIO correspondientes. Para la parte del MPU6050, se tuvo que realizar mucha investigación para poder leer los datos del giroscopio y acelerómetro en sus 6 grados de libertad en total correctamente. Una vez logrado esto, se tomaron diversas muestras para diferentes ciclos útiles y después se aplicaron algoritmos de estructuras de datos para ordenamiento y búsqueda. Se puede decir a grandes rasgos que en esta práctica se concentran varios conceptos esenciales en sistemas embebidos, que es el manejo del hardware y software de forma conjunta para un propósito específico.

1. REFERENCIAS

[1] Anda N. (2018). Motor DC de Factor Mx. Recuperado de: <https://www.factor.mx/portal/base-de-conocimiento/motor-dc/>

[2] Ajitgupta, P. (2021, February). I2C Communication Protocol. GeeksforGeeks; GeeksforGeeks.<https://www.geeksforgeeks.org/i2c-communication-protocol/>

[3] Barak, O. (2021, July 31). What is IMU? - Towards Data Science. Medium; Towards Data Science. [https://towardsdatascience.com/what-is-imu-9565e55b44chttps://towardsdatascience.com/what-is-imu-9565e55b44c](https://towardsdatascience.com/what-is-imu-9565e55b44c)

[4] Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to Algorithms (3rd ed.). The MIT Press.

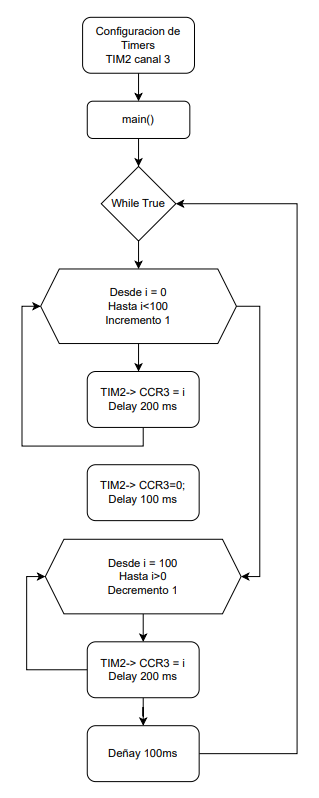
[5] Gunavaran Brihadiswaran. (2021, March 2). Introduction to Selection Sort - Nerd For Tech - Medium. Medium; Nerd For Tech. <https://medium.com/nerd-for-tech/introduction-to-selection-sort-19de8e72c89f>

[6] Kolade, C. (2022, September 29). Bubble Sort – Algorithm in Java, C++, Python with Example Code. FreeCodeCamp.org; freeCodeCamp.org. <https://www.freecodecamp.org/news/bubble-sort-algorithm-in-java-cpp-python-with-example-code/>

1. Anexos

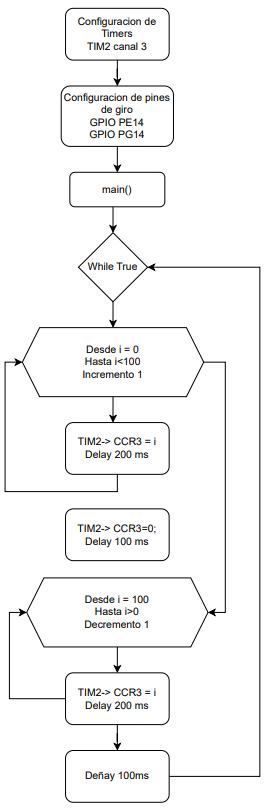
Video de evidencia: <https://youtu.be/F57ipB-0sus>

Anexo 1. Diagrama de flujo ejercicio 1



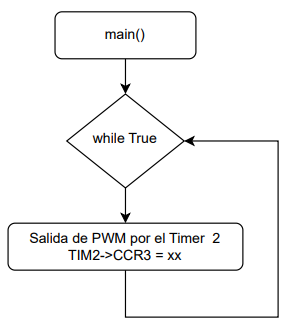
**Diagrama 1.** Ejercicio 1

Anexo 2. Diagrama de flujo ejercicio 2

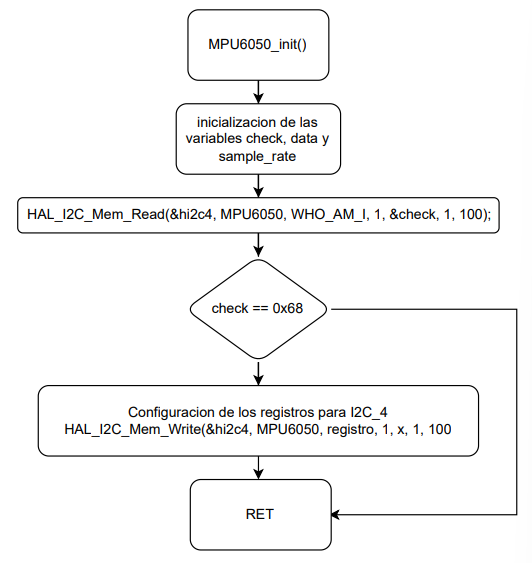


**Diagrama 2.** Ejercicio 2

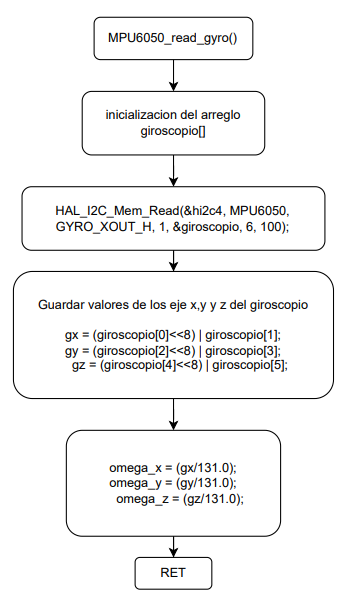
**Anexo 3.** Diagrama de flujo ejercicio 3



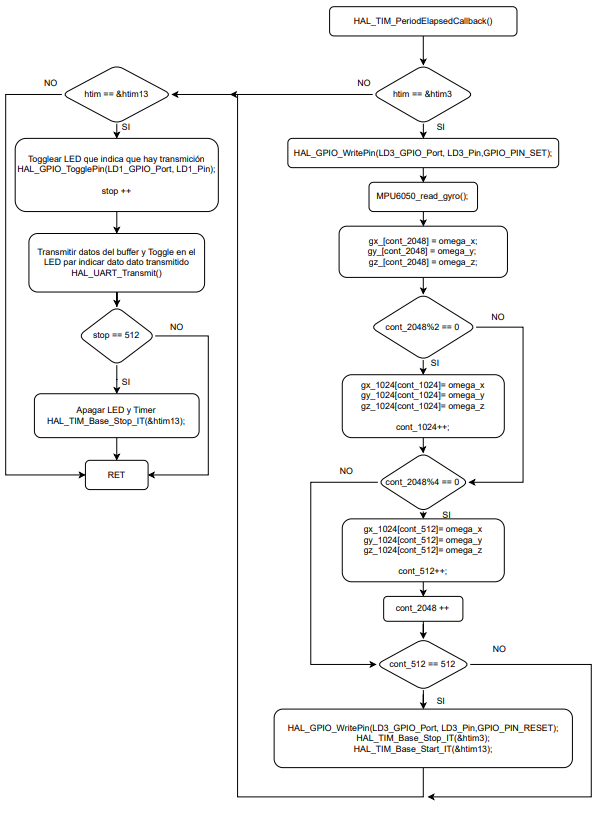
**Diagrama 3.** Ejercicio 3 función main



**Diagrama 4.** Ejercicio 3 función MPU\_init ()

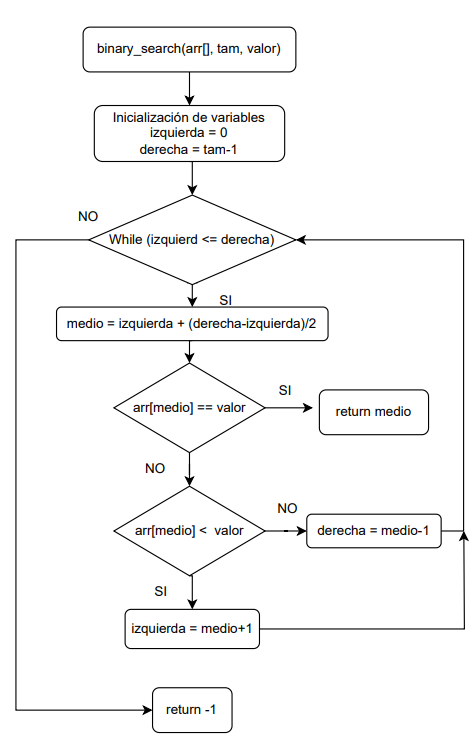


**Diagrama 5.** Ejercicio 3 MPU6050\_read\_gyro()

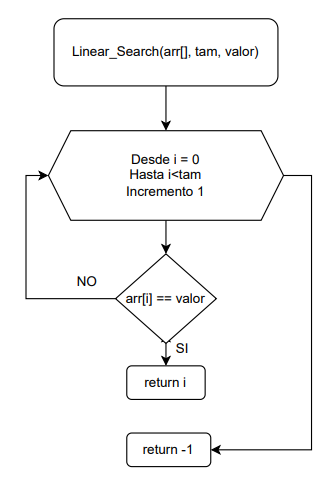


**Diagrama 7.** Ejercicio 3 interrupción por Timer

**Anexo 4.** Diagramas de flujo ejercicio 4



**Diagrama 8.** Ejercicio 4 búsqueda binaria



**Diagrama 8.** Ejercicio 4 búsqueda lineal

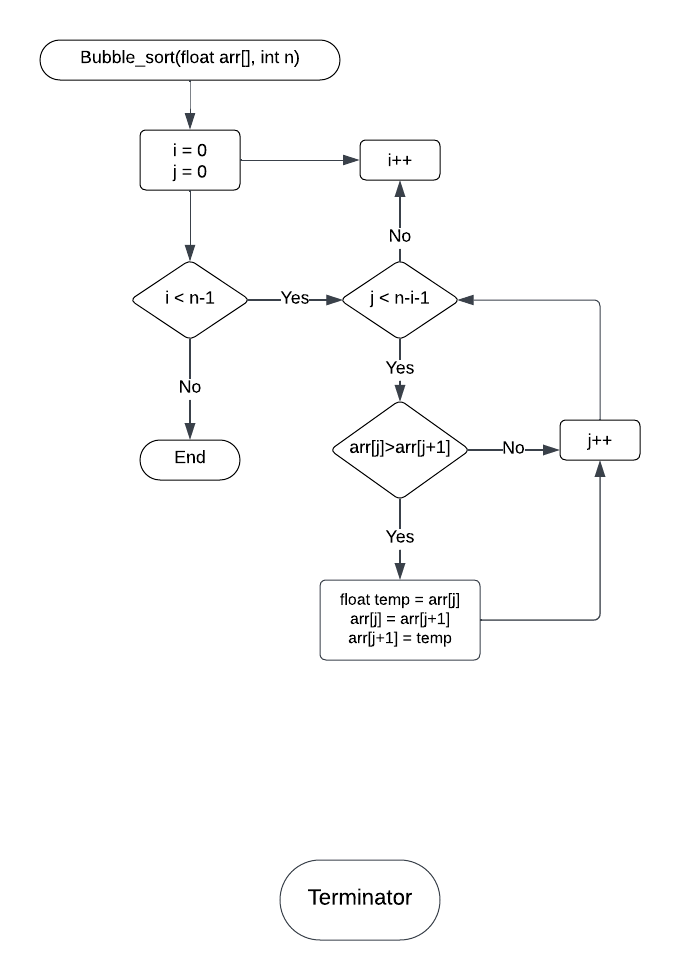


Diagrama 9. Ejercicio 4: Bubble Sort

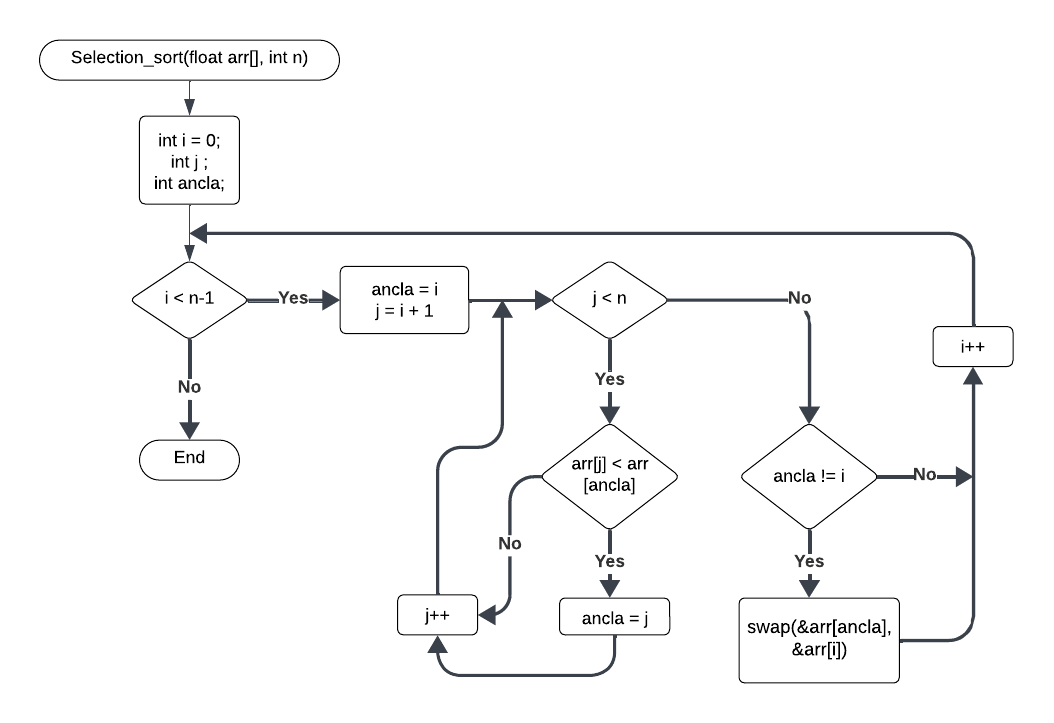


Diagrama 9. Ejercicio 4: Selection Sort

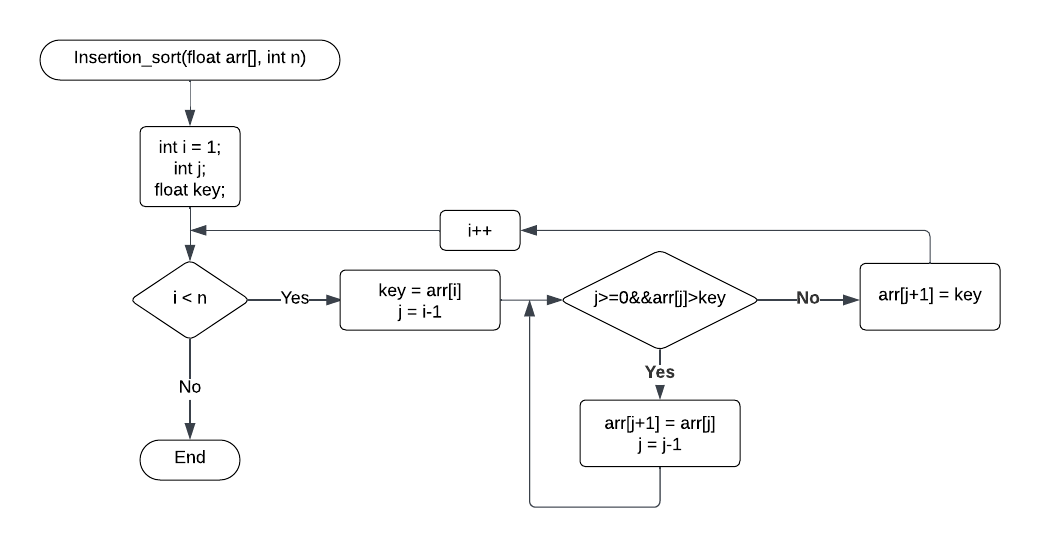
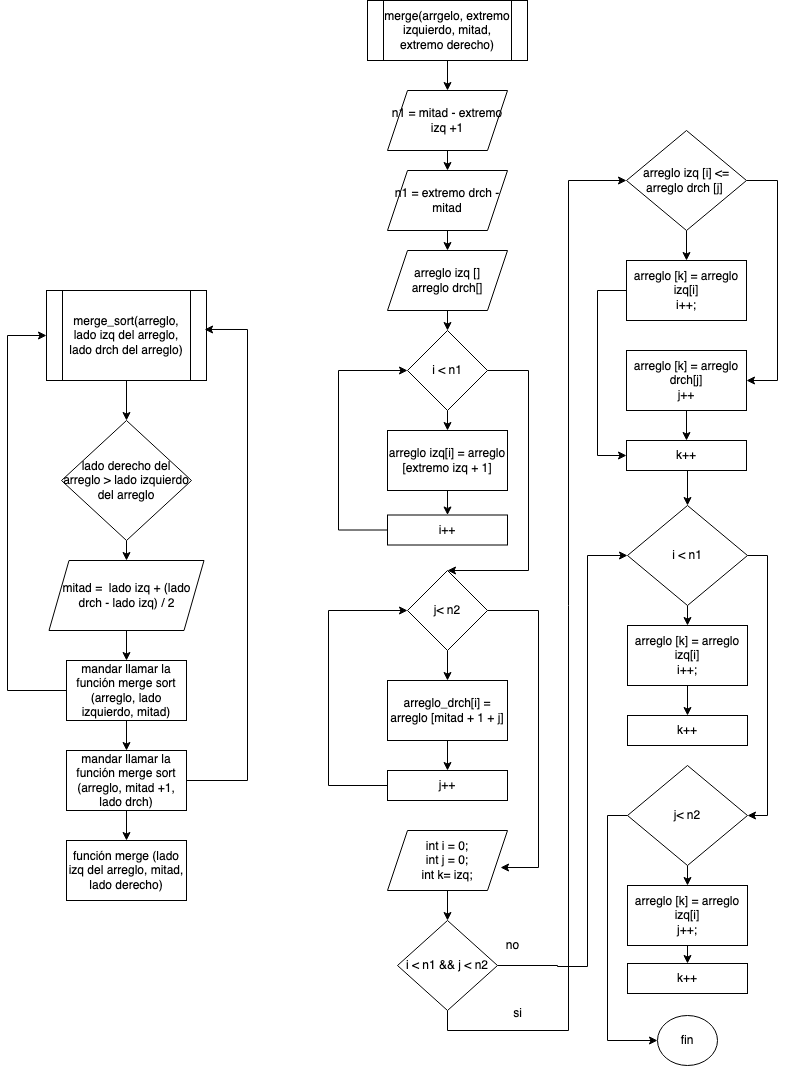
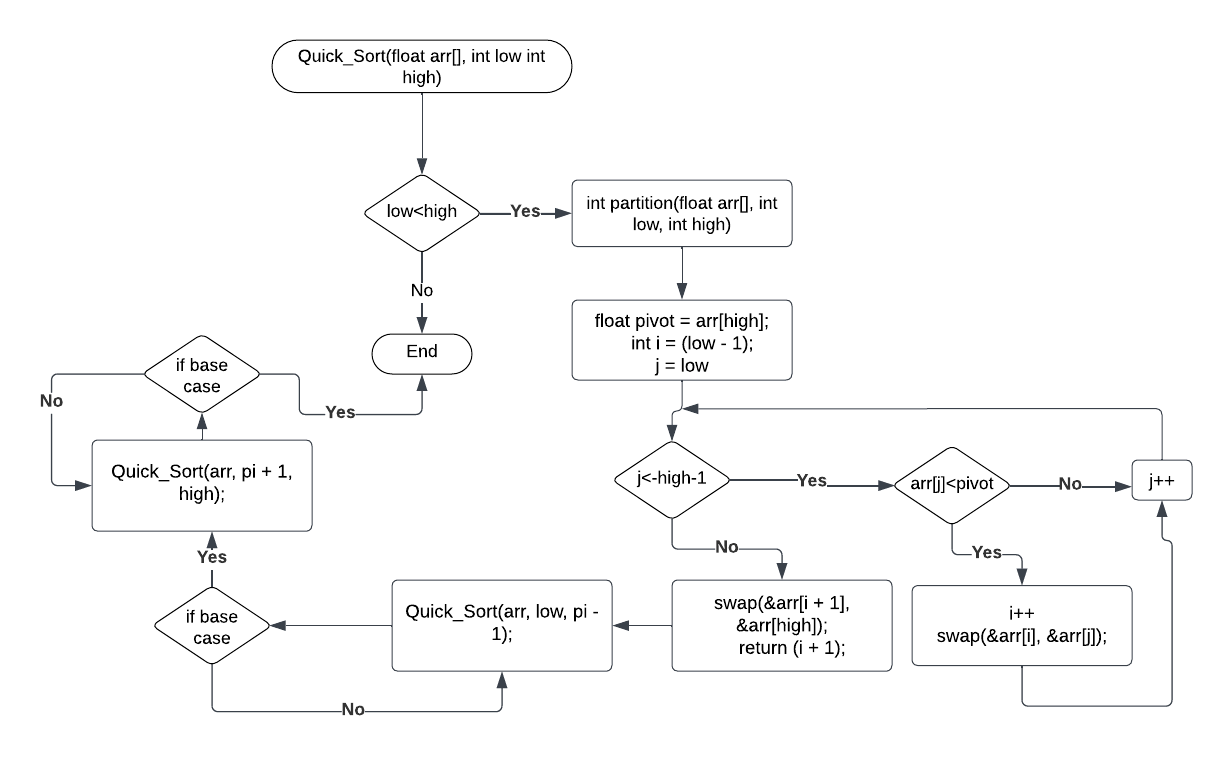


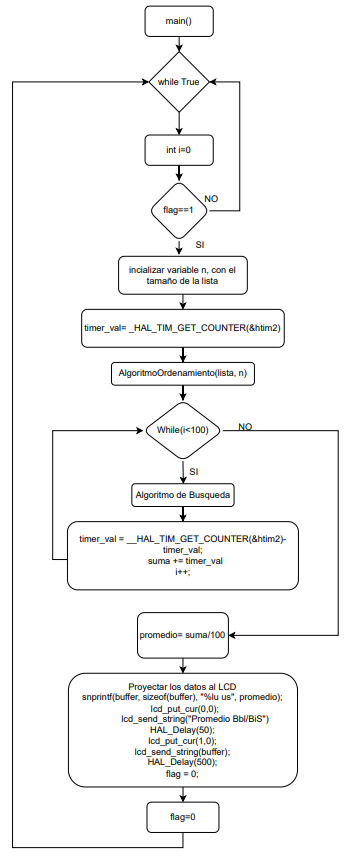
Diagrama 10. Ejercicio 4: Insertion Sort



**Diagrama 11**. Ejercicio 4: Merge sort



**Diagrama 11**. Ejercicio 12: Quick sort



**Diagrama 12**. Ejercicio 4: Main