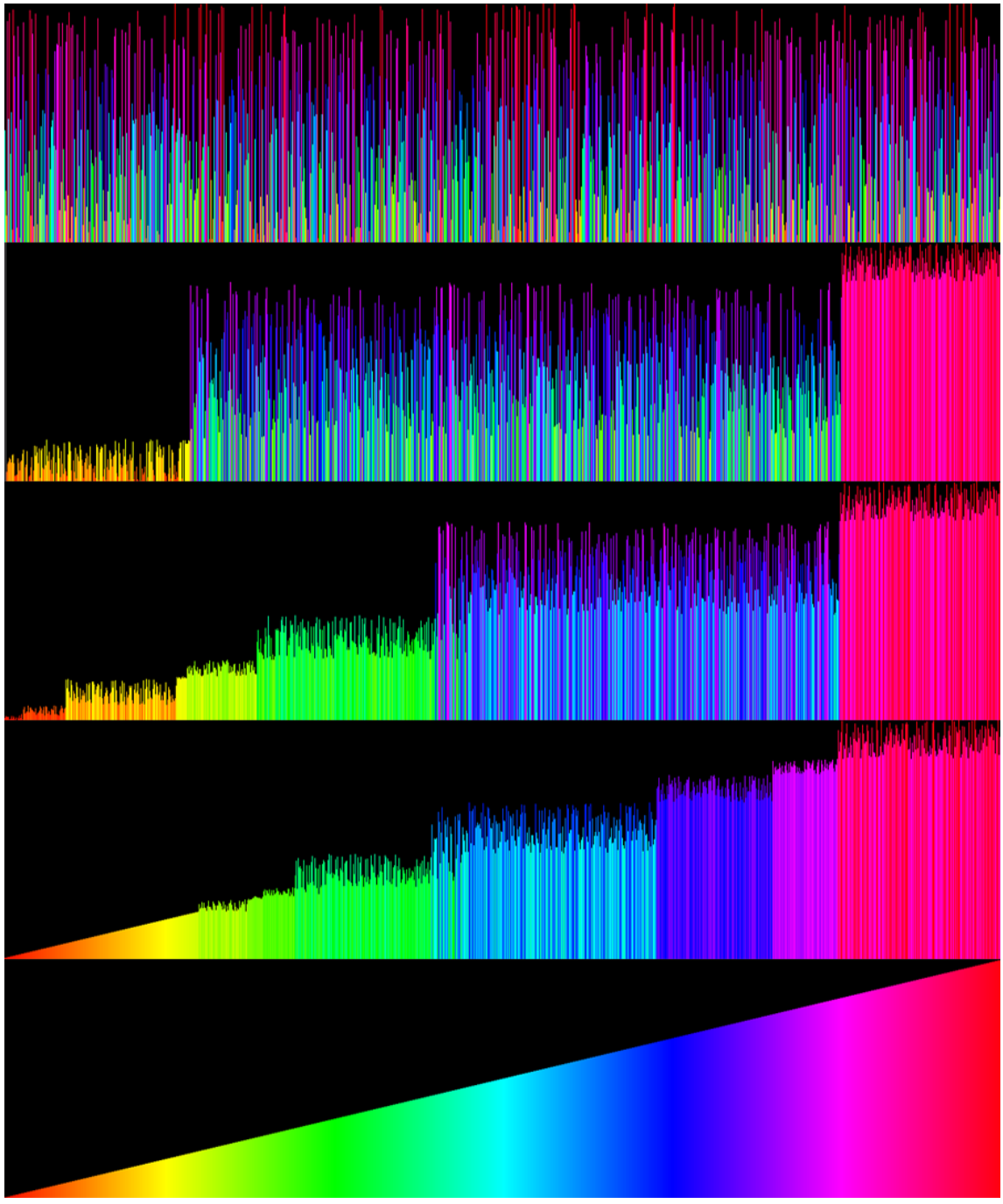
Paso a paso: Visualizador de algoritmos de ordenamiento



En este Paso a paso vamos a hacer un visualizador de algoritmos de ordenamiento usando la librería Pygame.

Algoritmos:

- Bubble Sort
- Insertion Sort

- Shell Sort
- Heap Sort
- Quick Sort

Parte 1: Preparación del proyecto

Instalar Pygame

En esta parte del proyecto solo vamos a crear un nuevo proyecto en Pycharm y vamos a instalar la librería `Pygame`.

Podemos hacerlo usando la terminal, la cuál la podemos encontrar abajo a la izquierda en Pycharm. Una vez abierta la terminal, escribimos el comando:

```
pip install pygame
```

Y listo.

> *En el caso de estar con las compus de la facultad, capaz no se puede instalar pygame de esta manera porque no tienen los permisos necesarios.*
>
> *En este caso, podemos instalarlo desde Pycharm siguiendo estos pasos:*
>
> 1. *Una vez abierto el proyecto en PyCharm vamos a la barra de menú superior.*
> 2. *Hacemos click en "File" (Archivo) y seleccionamos "Settings" (Configuración).*
> 3. *En la ventana de configuración, buscamos y seleccionamos "Project: [nombre del proyecto]".*
> 4. *Ahora Seleccionamos "Python Interpreter" (Intérprete de Python).*
> 5. *En la parte inferior izquierda de la ventana, hacemos clic en el botón "+" para agregar un nuevo paquete.*
> 6. *En la ventana emergente "Available Packages" (Paquetes Disponibles), escribimos "pygame" en el cuadro de búsqueda.*
> 7. *Seleccionamos "pygame" en la lista de resultados y hacemos clic en el botón "Install Package" (Instalar Paquete).*
> 8. *Esperamos a que PyCharm descargue e instale la librería pygame.*
> 9. *¡Listo!*

Estructurar el proyecto

Vamos a crear solo 2 archivos para este paso a paso.

- El primero se va a llamar `main.py` donde vamos a escribir todo lo relacionado a abrir y manejar la ventana.
- El segundo se va a llamar `algoritmos.py` y es donde vamos a escribir todos los algoritmos que vamos a usar.

En `main.py` vamos a importar lo siguiente:

```
import pygame
import random
```

```
from algoritmos import *
import time
```

Parte 2: Crear ventana

En esta parte no vamos a hacer nada relacionado a la lógica, solo vamos a crear una ventana y vamos a entender cómo recibir en pygame las interacciones del usuario.

Vamos a escribir todo esto en `main.py`.

Para abrir y cerrar una ventana se usa:

```
pygame.init()  # abrir
pygame.quit()  # cerrar
```

Para mantener abierta esta ventana, tenemos que hacer un *while* en el medio que solo termine cuando el usuario haga click en cerrar.

```
pygame.init()

abierto = True
while abierto:
    for event in pygame.event.get():  # iterar eventos de la ventana
        if event.type == pygame.QUIT:  # cerrar
            abierto = False

pygame.quit()
```

Ese ciclo *for* itera por los eventos que recibe la ventana, y `pygame.QUIT` es la variable de pygame que representa el botón de cerrar de la ventana.

Si ejecutamos esto no vamos a ver nada porque nunca instanciamos la ventana. Lo vamos a hacer hacer justo después del `pygame.init()` así:

```
ventana = pygame.display.set_mode((600, 300))  # ancho y alto de la ventana
```

Ahora si ejecutamos `main.py` deberíamos ver una pequeña ventana negra.

El siguiente paso es interactuar con la ventana a través del teclado.

A nosotros solo nos van a interesar estas teclas para este proyecto:

1. El `Espacio` para empezar el algoritmo de ordenamiento.
2. La letra `R` para resetear.
3. Las `flechas de izquierda y derecha` para ir cambiando de algoritmo.

En pygame, cada una de estas teclas están representadas por diferentes variables.

Sin enroscarnos mucho en detalles relacionados a la librería, así quedaría todo:

```
import pygame
import random
from algoritmos import *
import time


pygame.init()
ventana = pygame.display.set_mode((600, 300))

abierto = True
while abierto:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            abierto = False

        elif event.type == pygame.KEYDOWN:
            # resetear con R
            if event.key == pygame.K_r:
                print("R")

            # ordenar con espacio
            elif event.key == pygame.K_SPACE:
                print("Espacio")

            # cambiar algoritmo actual con flechitas
            elif event.key == pygame.K_RIGHT:
                print("Flecha derecha")
            elif event.key == pygame.K_LEFT:
                print("Flecha izquierda")

pygame.quit()
```

Ahora si ejecutamos esto, tendremos una pequeña ventana negra, y si apretamos alguna de esas teclas podremos ver impreso en consola la tecla que estamos presionando.

Como hemos ido viendo a lo largo del año, vamos a escribir todo eso adentro de una función principal.

Nuestro archivo quedaría así:

```
import pygame
import random
from algoritmos import *
import time


def principal():
    pygame.init()
    ventana = pygame.display.set_mode((600, 300))

    abierto = True
    while abierto:
```

```
        for event in pygame.event.get():  # iterar eventos de la ventana
            if event.type == pygame.QUIT:  # cerrar
                abierto = False

            elif event.type == pygame.KEYDOWN:
                # resetear con R
                if event.key == pygame.K_r:
                    print("R")

                # ordenar con espacio
                elif event.key == pygame.K_SPACE:
                    print("Espacio")

                # cambiar algoritmo actual con flechitas
                elif event.key == pygame.K_RIGHT:
                    print("Flecha derecha")
                elif event.key == pygame.K_LEFT:
                    print("Flecha izquierda")

    pygame.quit()


if __name__ == '__main__':
    principal()
```

Parte 3: Crear array y dibujarla

Crear array desordenada

Nosotros lo que vamos a hacer es tener creada una array ordenada al comienzo y la vamos a desordenar. Cada vez que el usuario quiera resetear el programa, solo vamos a volver a desordenarla.

Esta array va a ser así: [1, 2, 3, 4, 5,..., N]. Siendo `N` una variable que va a representar la cantidad de elementos de la array.

```
N = 30
array = [num for num in range(1, N + 1)]
random.shuffle(array)
```

Esto lo vamos a escribir aquí:

```
def principal():
    # ===== AQUÍ =====
    N = 30
    array = [num for num in range(1, N + 1)]
    random.shuffle(array)
    # ================

    pygame.init()
    ventana = pygame.display.set_mode((600, 300))
```

Dibujar array

Ahora vamos a escribir una función que se llame `dibujar()`, que reciba como parámetros la ventana y el array.

Esta función va a dibujar `N` cantidad de barras verticales en la ventana. Estas barras van a representar el arreglo. Todas las barras juntas van a ocupar toda la ventana a lo ancho y van a variar de altura según el valor de cada número.

> *Entonces, para saber el ancho de cada barra, solo tenemos que dividir el ancho de la ventana por* `N`.
>
> *Y para calcular el alto de cada barra tenemos que hacer regla de 3 simple. El valor más pequeño del arreglo (1), va a ser la barra más baja; y el valor más grande del arreglo (n) va a ser la barra más alta y va a ser tan alta como la ventana. Sabiendo esto podemos determinar la altura de cualquier barra del arreglo.*
>
> *Por ejemplo, si la* `altura de la ventana` *fuese de 300 píxeles, y* `N` *fuese 30, la barra que represente el número* `24` *va a ser* `24 * 300 / 30 = 240`.

Para saber el ancho y el alto en píxeles de la ventana podemos usar las funciones de pygame `.get_width()` y `.get_height()`.

Una vez entendido esto, tenemos que saber cómo se dibuja un rectángulo en pygame.

Esto se hace con la función `pygame.draw.rect()` a la cuál le vamos a pasar 4 parámetros:

- La ventana
- El color del rectángulo
- Una tupla con 4 números enteros:
    - La coordenada x del punto donde va a estar situada la esquina superior izquierda del rectángulo
    - La coordenada y del punto donde va a estar situada la esquina superior izquierda del rectángulo
    - El ancho en píxeles del rectángulo
    - El alto en píxeles del rectángulo

La ventana ya la tenemos, y lo demás deberíamos poder calcularlo ya.

> *Respecto al color, podríamos pasarle cualquier color. Nosotros vamos a hacer que dependiendo de la altura de la barra, tenga un color u otro. Hay una manera de representar los colores que se llama HSV (Hue, Saturation, Value; es decir: Tono, Saturación y Brillo). Son 3 números: el tono va desde el 0 al 360, y la saturación y el brillo van del 0 al 100.*
>
> *Nosotros vamos a poner la saturación y el brillo al máximo para todas las barras, lo único que va a cambiar es el tono.*
>
> *El número más pequeño del array va a estar representado por el color (0, 100, 100) y el número más grande por el color (360, 100, 100). Por regla de 3 simple podemos definir el resto de colores para los otros números del array.*

Así quedaría la función `dibujar()` completa:

```
def dibujar(ventana, array):
    n = len(array)
    ancho_ventana, alto_ventana = ventana.get_width(), ventana.get_height()
```

```
    # mismo ancho para todas las barras
    ancho_barra = ancho_ventana // n

    # fondo negro
    pygame.draw.rect(ventana, 'black', (0, 0, ancho_ventana, alto_ventana))

    for i in range(n):
        # si el alto de la barra con el número más alto es igual al alto de la
ventana, entonces, por regla de 3 simple:
        alto_barra = array[i] * alto_ventana // n

        # coordenadas de la esquina superior izquierda de la barra
        x = i * ancho_barra
        y = alto_ventana - alto_barra

        # en HSV, los colores tienen Hue [0-360] (tono), Saturation [0-100]
(saturación) y Value [0-100] (brillo)
        # si pusiéramos la saturación y el brillo al máximo y el tono dependiera de
la altura de la barra:
        tono = array[i] * 360 // n
        color = pygame.Color(0, 0, 0)
        color.hsva = (tono, 100, 100, 100)

        # dibujar rectángulo
        pygame.draw.rect(ventana, color, (x, y, ancho_barra, alto_barra))

    time.sleep(0.05)
    pygame.display.update()
```

> *La línea donde escribimos `time.sleep(0,05)` es necesaria porque sino la función se va a ejecutar tan rápido como nuestra computadora lo permita y capaz es demasiado rápido y no podemos apreciar el algoritmo. Podemos cambiar este valor a gusto.*

> *La línea donde escribimos `pygame.display.update()` también es necesaria porque sino no vamos a ver ningún cambio en la ventana.*

Ahora vamos a llamar a la función `dibujar()` justo después de que creamos y mezclamos la array.

```
def principal():
    N = 30
    array = [num for num in range(1, N + 1)]
    random.shuffle(array)

    pygame.init()
    ventana = pygame.display.set_mode((600, 300))

    # ===== AQUÍ =====
    dibujar(ventana, array)
    # ================
```

```
    abierto = True
    while abierto:
```

Ahora si ejecutamos el archivo, deberíamos ver una ventana con muchas barras de colores desordenadas.

> *Ya podríamos pasar al siguiente paso, pero vamos a agregarle un par de detallitos a lo que ya hicimos.*
>
> *Con la ventana de 600x300 que tenemos, si N fuese un número no divisible por 600, quedaría un sobrante del lado derecho. Por ejemplo, si N = 187, el ancho máximo posible de cada barra va a ser 600//187=3. Pero no alcanzaría a ocupar el ancho total de la ventana, porque 3 * 187 = 561, entonces sobrarían 39 píxeles.*
>
> *Vamos a tener esto presente a la hora de definir el ancho y el alto de la ventana.*
>
> *Definamos el alto de la ventana, el cual no va a cambiar. Por ejemplo,* `alto = 360`.
>
> *Ahora definamos un ancho máximo para la ventana. Por ejemplo,* `max_ancho = 1080`.
>
> *El ancho que le vamos a dar a la ventana va a ser el ancho justo que necesitamos, ignorando ese sobrante. Eso lo podemos hacer de la siguiente manera:* `ancho = max_ancho // n * n`.
>
> *Perfecto, solo falta validar una última cosa: el ancho MÍNIMO de cada barra va a ser 1 píxel. Por lo tanto, como el ancho de la barra es ancho_ventana // n, vamos a hacer que si* `N` *es un número mayor a la cantidad de píxeles del ancho de la ventana, entonces que* `N` *sea igual al ancho de la ventana.*
>
> *Todo esto quedaría así:*
>
> ```
> # determinamos una ventana de aproximadamente 1080x360
> max_ancho = 1080
> alto = 360
> n = min(max_ancho, N)
> ancho = max_ancho // n * n
> ```
>
> *Borremos las siguientes líneas de la función principal:*
>
> ```
> N = 30
> array = [num for num in range(1, N + 1)]
> random.shuffle(array)
>
> pygame.init()
> ventana = pygame.display.set_mode((600, 300))
> ```
>
> *Y las intercambiemos por las siguientes:*
>
> ```
> N = 50
>
> max_ancho = 1080
> alto = 360
> n = min(max_ancho, N)
> ancho = max_ancho // n * n
>
> array = [num for num in range(1, n + 1)]
> random.shuffle(array)
> ```

```
    pygame.init()
    ventana = pygame.display.set_mode((ancho, alto))
```

Ahora deberíamos poder cambiar el valor de `N` a nuestro antojo sin pensar en ese cachito sobrante a la derecha.

Parte 4: Agregar algoritmo y visualizar dinámicamente

Ordenar

Ya tenemos todo listo para ir a nuestro archivo `algoritmos.py` y escribir nuestro primer algoritmo de ordenamiento.

Vamos a empezar con el `Bubble Sort`. Como ya hemos visto en clase, el algoritmo es así:

```
def bubble_sort(v):
    n = len(v)
    for i in range(n - 1):
        for j in range(n - i - 1):
            if v[j] > v[j + 1]:
                v[j], v[j + 1] = v[j + 1], v[j]
```

Nosotros queremos ver dinámicamente cómo va a actuando el algoritmo. Para eso vamos a dibujar el array **cada** vez que se haga un intercambio de dos números.

Si a `bubble_sort()` le pasáramos la ventana y la función de `dibujar()`, podríamos hacerlo de la siguiente manera:

```
def bubble_sort(ventana, v, dibujar):
    n = len(v)
    for i in range(n - 1):
        for j in range(n - i - 1):
            if v[j] > v[j + 1]:
                v[j], v[j + 1] = v[j + 1], v[j]
                dibujar(ventana, v)
```

Ya podríamos agregar una invocación a `bubble_sort()` al apretar **espacio**, en `main.py`. Aquí:

```
            elif event.key == pygame.K_SPACE:
                # ===== AQUÍ =====
                bubble_sort(ventana, array, dibujar)
                # ================
```

¡Listo! Ya podemos visualizar nuestro primer algoritmo. Podemos probar con distintos valores para `N` y cambiando la velocidad de la función `dibujar()` para ver cómo se comporta el programa.

Resetear

Vamos a agregar también que al apretar la tecla **R** se resetee el programa. Es decir, al apretar la tecla **R** vamos a mezclar y dibujar el array desordenada.

Como esto es algo que vamos a repetir un par de veces más a lo largo del programa lo vamos a escribir en una función que se llame `resetear()` :

```
def resetear(ventana, array):
    random.shuffle(array)
    dibujar(ventana, array)
```

Entonces, al apretar la **R**, llamemos esta función:

```
                if event.key == pygame.K_r:
                    # ===== AQUÍ =====
                    resetear(ventana, array)
                    # ================
```

> *Ya podríamos pasar al siguiente paso, pero vamos a pulir algunos detallitos.*
>
> *Ahora mismo la tecla **R** nos permite resetear el programa, pero solo si no está ejecutándose un algoritmo.*
>
> *Esto es porque mientras se está ejecutando el algoritmo estamos adentro de una iteración del ciclo for (en la que iteramos los inputs del usuario). Recién al terminar el algoritmo vamos a pasar a la siguiente iteración y recién ahí pygame va a darse cuenta de que el usuario apretó la tecla **R**.*
>
> *Nosotros necesitamos saber si el usuario apretó la tecla **R** después de cada iteración del algoritmo. Esto lo vamos a hacer creando una clase `Exception` . Como esto se escapa de los temas de la materia, no vamos a explicarlo ahora.*
>
> *Resumidamente, vamos a agregar el siguiente código al comienzo de `main.py` :*
>
> ```
> class StopSorting(Exception):
>     pass
>
>
> def mostrar_intercambio(ventana, v):
>     dibujar(ventana, v)
>
>     # interrumpir el algoritmo de ordenamiento si se aprieta la tecla R
>     pygame.event.pump()
>     for event in pygame.event.get():
>         if event.type == pygame.KEYDOWN:
>             if event.key == pygame.K_r:
>                 raise StopSorting>
> ```
>
> *Ahora, en la parte del código dónde llamábamos al algoritmo vamos a borrar lo que habíamos puesto y vamos a poner:*
>
> ```
>                 elif event.key == pygame.K_SPACE:
>                     # ================
>                     try:
>                         bubble_sort(ventana, array, mostrar_intercambio)
>                     except StopSorting:
> ```

```
                    resetear(ventana, array)
                # ================
```

*¡Listo! Ya deberíamos poder resetear el programa con la tecla **R** mientras se está ejecutando el algoritmo.*

Parte 5: Agregar algoritmos

Agregamos el código de los algoritmos

Vamos a agregar los otros algoritmos vistos en clase:

- `Insertion Sort`
- `Shell Sort`
- `Heap Sort`
- `Quick Sort`

La lógica es la misma: debemos escribir en `algoritmos.py` el algoritmo; pasarle, además del arreglo, la ventana y la función `mostrar_intercambio()` ; y llamarla cada vez que se hace un intercambio de dos números.

Así quedaría el archivo `algoritmos.py` completo con los 5 algoritmos:

```
def bubble_sort(ventana, v, mostrar_intercambio):
    n = len(v)
    for i in range(n - 1):
        for j in range(n - i - 1):
            if v[j] > v[j + 1]:
                v[j], v[j + 1] = v[j + 1], v[j]
                mostrar_intercambio(ventana, v)


def insertion_sort(ventana, v, mostrar_intercambio):
    n = len(v)
    for j in range(1, n):
        y = v[j]
        k = j - 1
        while k >= 0 and y < v[k]:
            v[k + 1] = v[k]
            mostrar_intercambio(ventana, v)
            k -= 1
        v[k + 1] = y


def shell_sort(ventana, v, mostrar_intercambio):
    n = len(v)
    h = 1
    while h <= n // 9:
        h = 3 * h + 1

    while h > 0:
        for j in range(h, n):
            y = v[j]
```

```
            k = j - h
            while k >= 0 and y < v[k]:
                v[k + h] = v[k]
                k -= h
            v[k + h] = y
            mostrar_intercambio(ventana, v)
        h //= 3


def heap_sort(ventana, v, mostrar_intercambio):
    n = len(v)

    for i in range(n):
        e = v[i]
        s = i
        f = (s - 1) // 2
        while s > 0 and v[f] < e:
            v[s] = v[f]
            s = f
            f = (s - 1) // 2
        v[s] = e
        mostrar_intercambio(ventana, v)

    for i in range(n - 1, 0, -1):
        valori = v[i]
        v[i] = v[0]
        f = 0
        if i == 1:
            s = -1
        else:
            s = 1
        if i > 2 and v[2] > v[1]:
            s = 2
        while s >= 0 and valori < v[s]:
            v[f] = v[s]
            f = s
            s = 2 * f + 1
            if s + 1 <= i - 1 and v[s] < v[s + 1]:
                s += 1
            if s > i - 1:
                s = -1
        v[f] = valori
        mostrar_intercambio(ventana, v)


def quick(ventana, v, izq, der, mostrar_intercambio):
    pivot = v[(izq + der) // 2]

    i, j = izq, der
    while i <= j:

        while v[i] < pivot and i < der:
```

```
            i += 1

        while v[j] > pivot and j > izq:
            j -= 1

        if i <= j:
            v[i], v[j] = v[j], v[i]
            mostrar_intercambio(ventana, v)
            i += 1
            j -= 1

    if izq < j:
        quick(ventana, v, izq, j, mostrar_intercambio)

    if i < der:
        quick(ventana, v, i, der, mostrar_intercambio)


def quick_sort(ventana, v,  mostrar_intercambio):
    quick(ventana, v, 0, len(v) - 1, mostrar_intercambio)
```

Ahora, si vamos a donde llamamos a `bubble_sort()` y ponemos el nombre de otro algoritmo, deberíamos poder ver otro algoritmo.

Cambiar algoritmo actual con flechitas

Esto que hicimos hasta ahora no tiene mucha gracia ya que para ver otro algoritmo deberíamos cerrar el programa y escribir manualmente la función de otro algoritmo.

Para hacerlo más dinámico, vamos a guardar todos los algoritmos en una lista y vamos a guardar en una variable el índice del algoritmo actual.

Cada vez que queramos cambiar de algoritmo con las flechitas, cambiamos el índice y listo.

Entonces, vayamos a `main.py` y agreguemos estas dos nuevas líneas:

```
def principal():
    N = 50

    max_ancho = 1080
    alto = 360
    n = min(max_ancho, N)
    ancho = max_ancho // n * n

    array = [num for num in range(1, n + 1)]
    random.shuffle(array)

    pygame.init()
    ventana = pygame.display.set_mode((ancho, alto))

    dibujar(ventana, array)
```

```
    # ===== AQUÍ =====
    algoritmos = [bubble_sort, insertion_sort, shell_sort, heap_sort, quick_sort]
    actual = 0
    # ================

    abierto = True
    while abierto:
```

Ahora vamos a escribir una función que sume o reste 1 a `actual` para cambiar de algoritmo.

Cuando el usuario apriete la flechita de la derecha, va a sumar 1; y cuando apriete la flechita de la izquierda, va a restar 1.

Tenemos que tener presente que si el índice está por debajo de 0 o por arriba de 4 (que es el largo de la lista con los algoritmos) va a dar error. Dicho esto, la función quedaría así:

```
def cambiar_algoritmo(algoritmos, actual, cambio):  # cambio va a ser 1, o -1
    actual += cambio
    n = len(algoritmos)
    # validar que no tengamos un índice fuera de rango
    if actual == n:
        actual = 0
    elif actual < 0:
        actual = n - 1

    print('Algoritmo:', algoritmos[actual].__name__)  # mostrar el nombre de la
función del algoritmo actual
    return actual
```

Ahora tenemos que llamar a esta función cuando el usuario apriete las flechitas. Quedaría así:

```
                elif event.key == pygame.K_RIGHT:
                    # ================
                    actual = cambiar_algoritmo(algoritmos, actual, 1)
                    resetear(ventana, array)
                    # ================

                elif event.key == pygame.K_LEFT:
                    # ================
                    actual = cambiar_algoritmo(algoritmos, actual, -1)
                    resetear(ventana, array)
                    # ================
```

Ahora que el usuario puede elegir el algoritmo, tenemos que llamar la función del algoritmo actual.

Vamos a ir a la parte donde preguntamos si el usuario apretó la tecla **espacio**, vamos a borrar la llamada al algoritmo que teníamos antes y vamos a escribir lo siguiente:

```
                elif event.key == pygame.K_SPACE:
                    try:
                        # ================
```

```
                    algoritmos[actual](ventana, array, mostrar_intercambio)
                    # ================
                except StopSorting:
                    resetear(ventana, array)
```

¡Listo! Programa terminado.

Código de `main.py` completo:

```
import pygame
import random
from algoritmos import *
import time


class StopSorting(Exception):
    pass


def mostrar_intercambio(ventana, v):
    dibujar(ventana, v)

    # interrumpir el algoritmo de ordenamiento si se aprieta la tecla R
    pygame.event.pump()
    for event in pygame.event.get():
        if event.type == pygame.KEYDOWN:
            if event.key == pygame.K_r:
                raise StopSorting


def cambiar_algoritmo(algoritmos, actual, cambio):  # cambio va a ser 1, o -1
    actual += cambio
    n = len(algoritmos)
    # validar que no tengamos un índice fuera de rango
    if actual == n:
        actual = 0
    elif actual < 0:
        actual = n - 1

    print('Algoritmo:', algoritmos[actual].__name__)  # mostrar el nombre de la
función del algoritmo actual
    return actual


def resetear(ventana, array):
    random.shuffle(array)
    dibujar(ventana, array)


def dibujar(ventana, array):
    n = len(array)
```

```
    ancho_ventana, alto_ventana = ventana.get_width(), ventana.get_height()

    # mismo ancho para todas las barras
    ancho_barra = ancho_ventana // n

    # fondo negro
    pygame.draw.rect(ventana, 'black', (0, 0, ancho_ventana, alto_ventana))

    for i in range(n):
        # si el alto de la barra con el número más alto es igual al alto de la
ventana, entonces, por regla de 3 simple:
        alto_barra = array[i] * alto_ventana // n

        # coordenadas de la esquina superior izquierda de la barra
        x = i * ancho_barra
        y = alto_ventana - alto_barra

        # en HSV, los colores tienen Hue [0-360] (tono), Saturation [0-100]
(saturación) y Value [0-100] (brillo)
        # si pusiéramos la saturación y el brillo al máximo y el tono dependiera de
la altura de la barra:
        tono = array[i] * 360 // n
        color = pygame.Color(0, 0, 0)
        color.hsva = (tono, 100, 100)

        # dibujar rectángulo
        pygame.draw.rect(ventana, color, (x, y, ancho_barra, alto_barra))

    time.sleep(0.05)
    pygame.display.update()


def principal():
    # cantidad de elementos del array
    N = 50

    # determinamos ancho y alto de la ventana. El máximo es 1080x360
    max_ancho = 1080
    alto = 360
    n = min(max_ancho, N)
    ancho = max_ancho // n * n

    # inicializamos la ventana
    pygame.init()
    ventana = pygame.display.set_mode((ancho, alto))

    # crearmos, mezclamos y dibujamos array
    array = [num for num in range(1, n + 1)]
    resetear(ventana, array)

    # guardamos todos los algoritmos en un arreglo para poder ir cambiándolo durante
la visualización
```

```
    algoritmos = [bubble_sort, insertion_sort, shell_sort, heap_sort, quick_sort]
    actual = 0

    # loop que mantiene abierta la ventana
    abierto = True
    while abierto:
        for event in pygame.event.get():
            # cerrar
            if event.type == pygame.QUIT:
                abierto = False

            elif event.type == pygame.KEYDOWN:
                # resetear con R
                if event.key == pygame.K_r:
                    resetear(ventana, array)

                # ordenar con espacio
                elif event.key == pygame.K_SPACE:
                    try:
                        algoritmos[actual](ventana, array, mostrar_intercambio)
                    except StopSorting:
                        resetear(ventana, array)

                # cambiar algoritmo actual con flechitas
                elif event.key == pygame.K_RIGHT:
                    actual = cambiar_algoritmo(algoritmos, actual, 1)
                    resetear(ventana, array)
                elif event.key == pygame.K_LEFT:
                    actual = cambiar_algoritmo(algoritmos, actual, -1)
                    resetear(ventana, array)

    # cerramos ventana
    pygame.quit()


if __name__ == '__main__':
    principal()
```