	Carátula para entrega de prácticas
Facultad de Ingeniería	Laboratorio de docencia

Laboratorios de computación salas A y B

Profesor: M.I. Edgar Tista García

Asignatura: Estructura de Datos y Algoritmos I

Grupo: 1

No de Práctica(s): 11

Integrante(s): Díaz Hernández Marcos Bryan

No. de Equipo de cómputo empleado:

No. de Lista o Brigada: 9

Semestre: 2020-2

Fecha de entrega: 13 de mayo de 2020

Observaciones:

CALIFICACIÓN: _____

Objetivo de la práctica

El objetivo es implementar, al menos, dos enfoques de diseño (estrategias) de algoritmos y analizar las implicaciones de cada uno de ellos.

Introducción

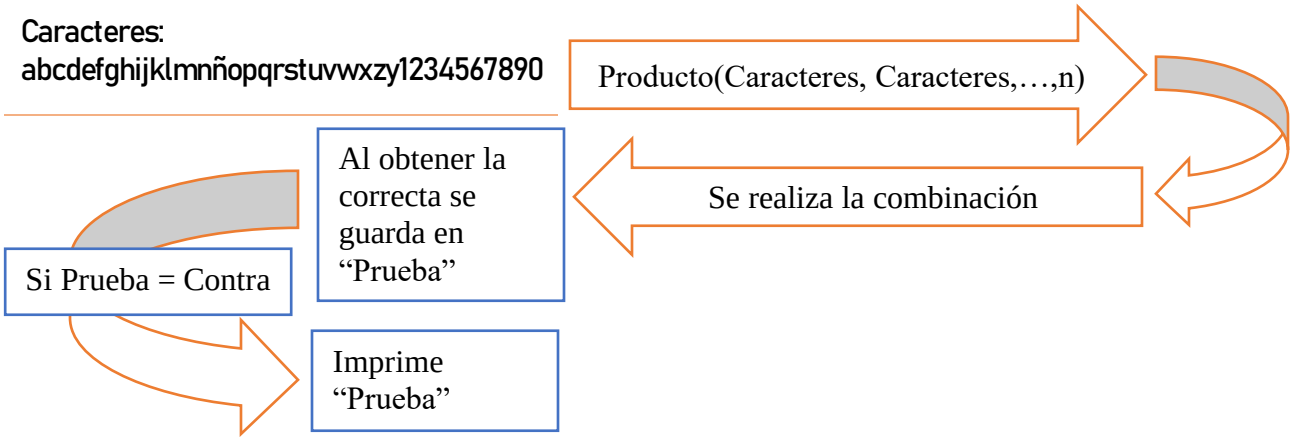
He realizado esta práctica con la intención de poder identificar los tipos de estrategias, por medio de diagramas y análisis escritos, para aprender más sobre construir algoritmos.

Ejercicios de la guía de laboratorio

- Primer ejemplo: Código - Fuerza Bruta

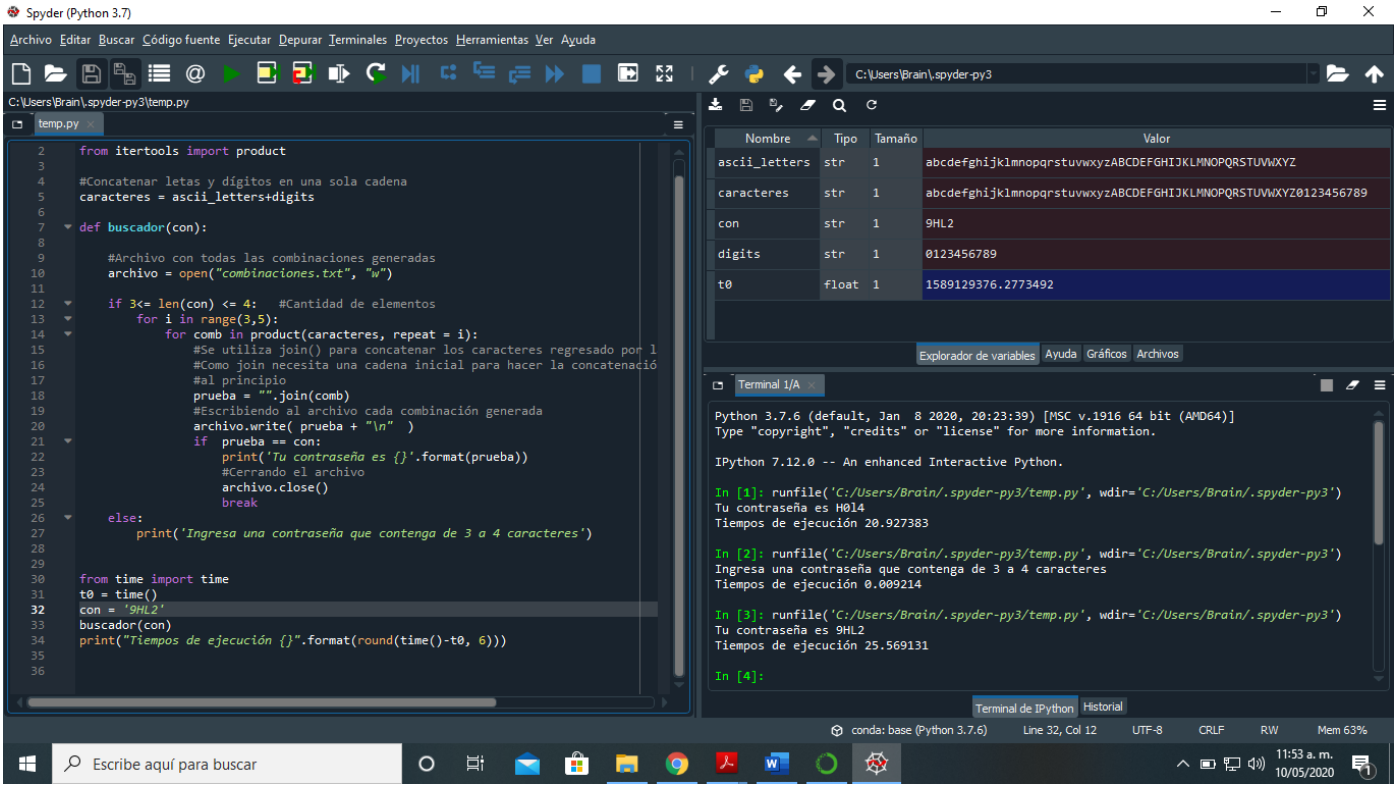
El programa tiene como objetivo buscar una contraseña de 4 o 3 dígitos, por medio de ciclos for anidados, para obtener la contraseña realiza combinaciones de la cadena "caracteres" repetidas veces hasta obtener la idéntica a la insertada, y después imprimirla.

- Diagrama de funcionamiento:



El programa es de fuerza bruta debido a que realiza muchos procesos para poder comprobar una condición, en este caso se podría alargar el número de procesos debido a que los elementos de la contraseña podrían ser más, además de ser un proceso reiterativo donde se busca elemento a elemento y de forma consecutiva se realiza esto, lo que implica obtener un resultado, sin embargo, toma mucho tiempo la ejecución de programa.
En dado caso podría ser Greedy, pero no existen tomas de decisiones que aseguren solución a las subetapas creadas, si se consideran como subetapas las combinaciones que generan los ciclos for, por ello se limita a que todas las posibles decisiones tendrían que retornar y buscar la combinación ideal o la combinación que se establece.

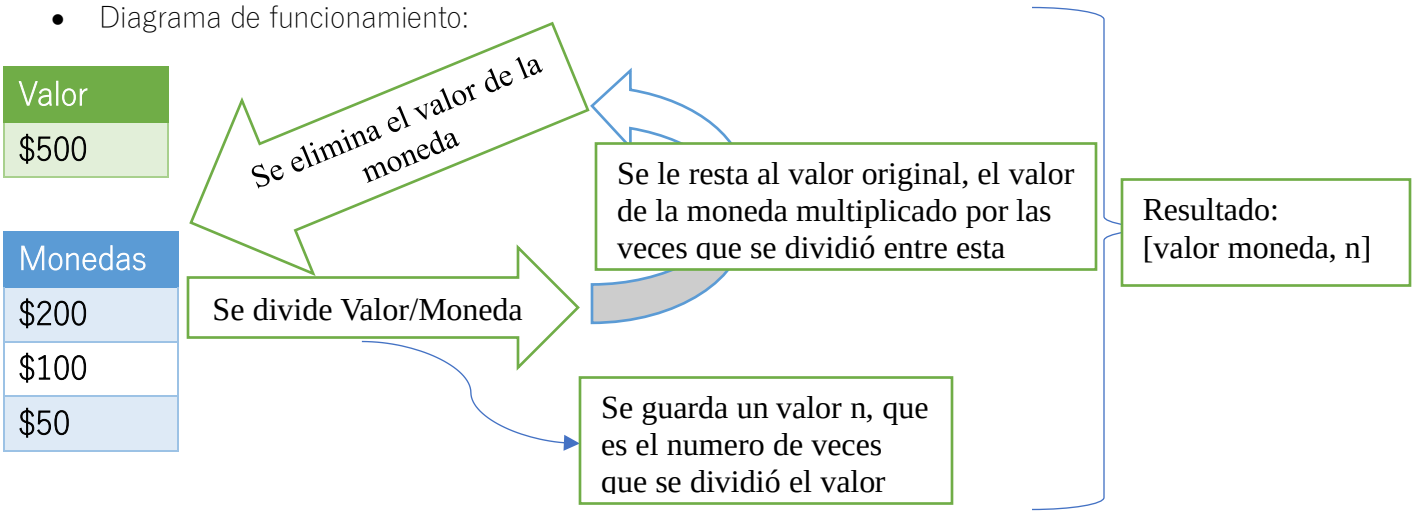
- Evidencia de implementación.



- Segundo ejemplo: Código – Greedy

El segundo ejemplo resuelve el ejercicio de las monedas que consiste en dar cambio en monedas de una determinada cantidad, de ahí que el cambio pueda variar de acuerdo con las decisiones que se hayan tomado, en este caso el programa comienza a tomar las monedad que se encuentren en la primera posición del arreglo que corresponde a “denominaciones” y de esa primera decisión hacer las operaciones que se van a dividir en siguientes decisiones.

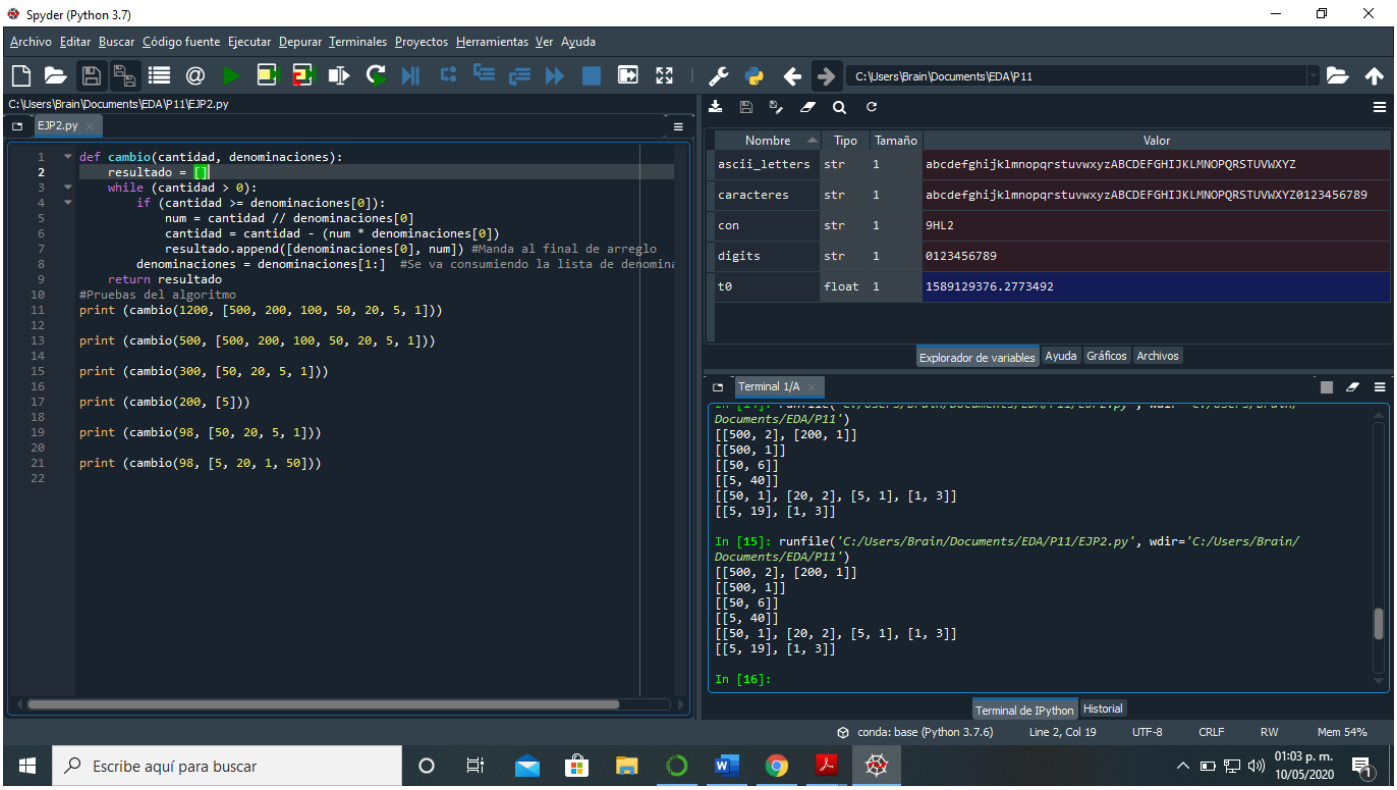
- Diagrama de funcionamiento:



Para ser Greedy es necesario que tome decisiones por etapas y que a partir de esas etapas sea posible encontrar una solución al problema, de acuerdo con lo anterior es necesario el que se seleccione un punto de donde partir, lo que hace el programa es que se selecciona el primer valor, sin importar si es el de mayor o menor valor, lo que puede resultar en un programa con solución pero que no es eficiente, y tampoco existe una selección como tal ya que el programa simplemente marca una posición que debe tomar e ir de forma consecutiva con los siguientes valores, lo que no es del todo una toma de decisiones, porque no decide de cual de todos los valores tomar, no hay un criterio extra.

Sin embargo, cumple con la definición de resolver por fases y dependiendo de las decisiones encontrar una solución, que pude ser eficiente o no, además al cambiar las posiciones de los valores, las soluciones eran menos eficientes, lo que da como conclusión que si es Greedy.

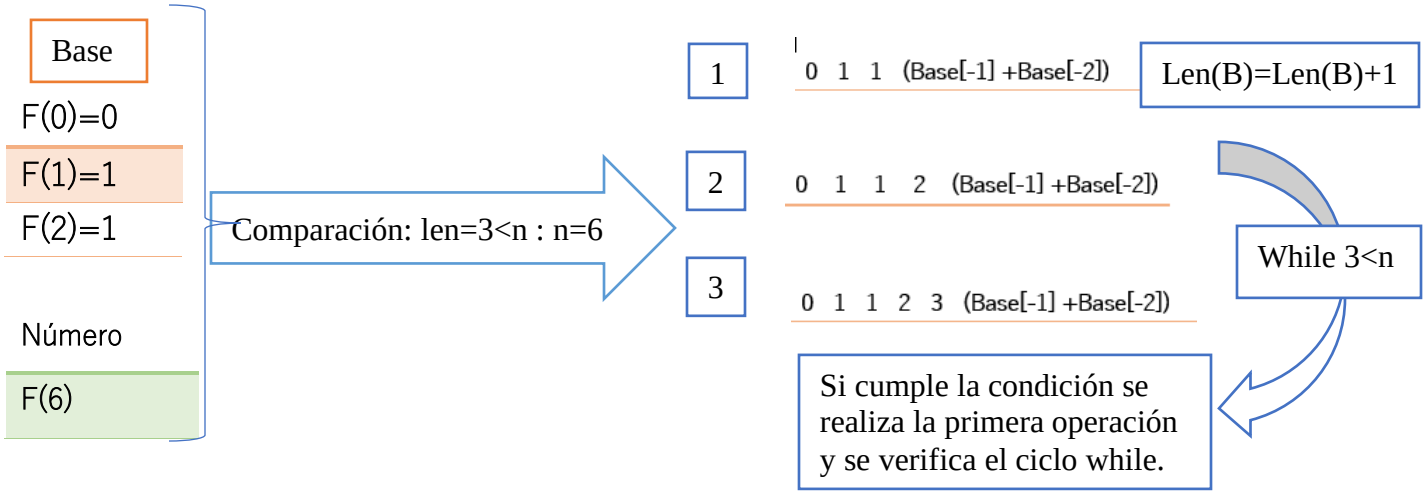
- Evidencia de implementación.



- Tercer ejemplo: Código – Bottom - Up

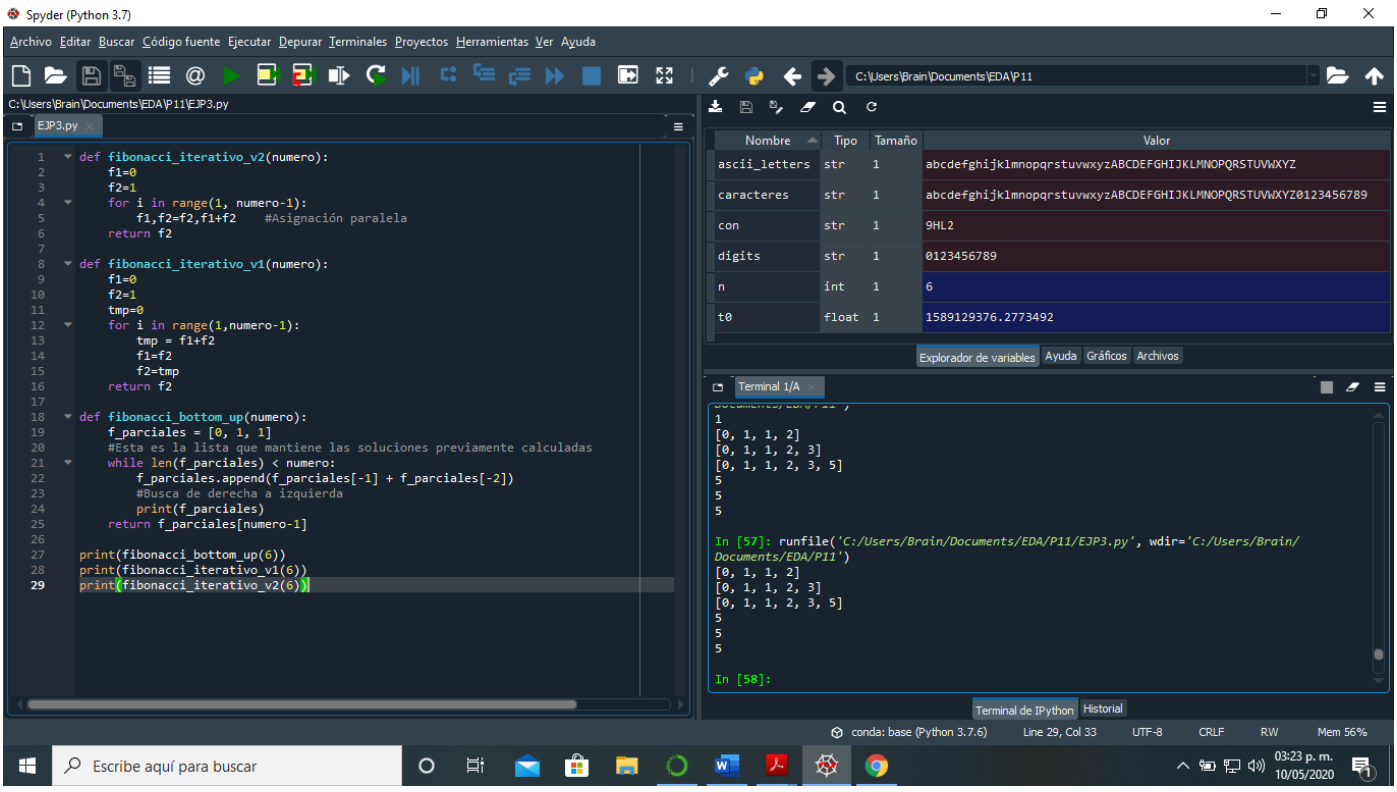
El programa busca encontrar el elemento n de la sucesión de Fibonacci, para ello plantea tres formas de lograrlo donde la última es la que utiliza el método Bottom – Up, para ello es necesario tener elementos clave que puedan ayudar a resolver cualquier caso, tal como lo son los valores de F(0)=0 y F(1)=1, con estos se podrían obtener todos los valores ya que son la base.

- Diagrama de funcionamiento:



El programa corresponder al método Bottom – Up porque se necesita de una base para poder solucionar los problemas, es como un caso base para todas las soluciones, en este caso son los dos primeros términos, de ahí que se pueda resolver el problema modificando estos valores, y que estos casos base son sencillos. De otra forma parece que es una ecuación de recurrencia, que de hecho es posible resolverla sucesión de Fibonacci por medio de ecuaciones de recurrencia, y es prácticamente la misma metodología porque necesita casos base, solo que es más teórica y conceptual.

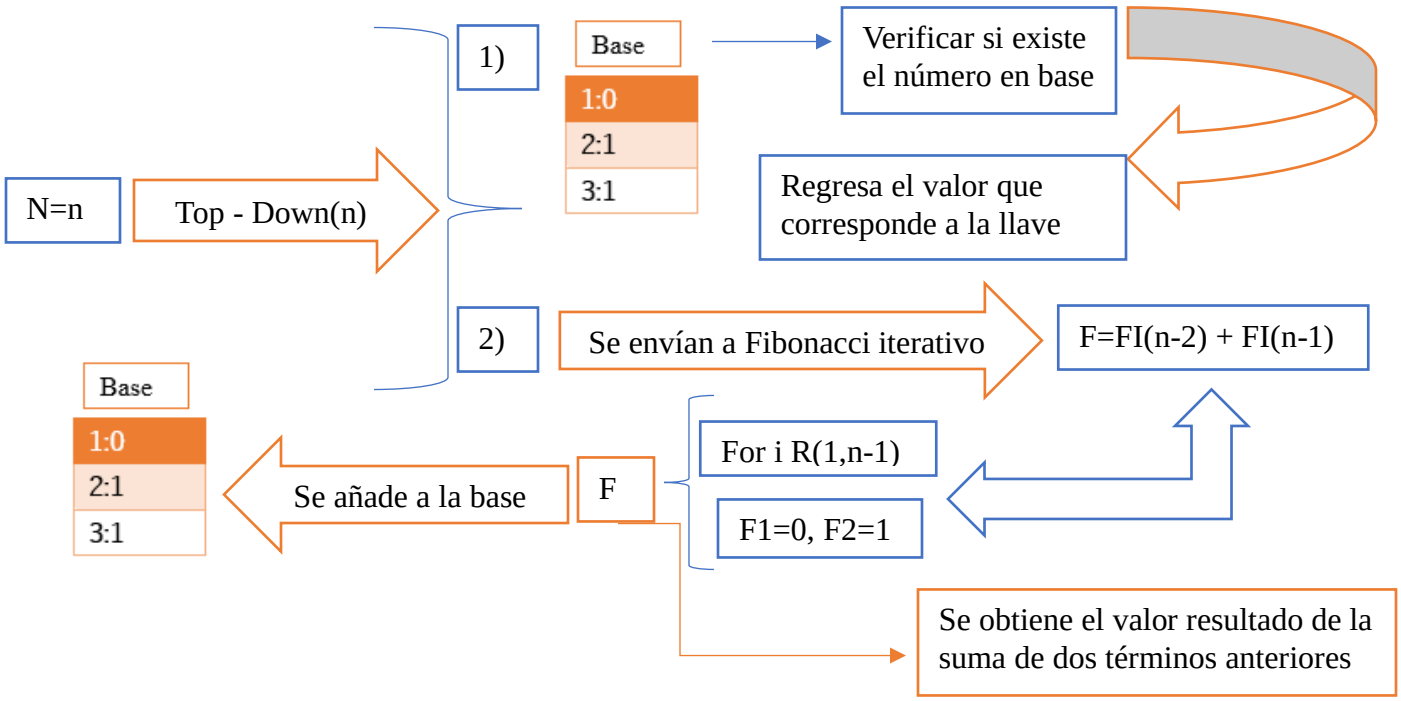
- Evidencia de implementación.



- Cuarto ejemplo: Código – Top – Down

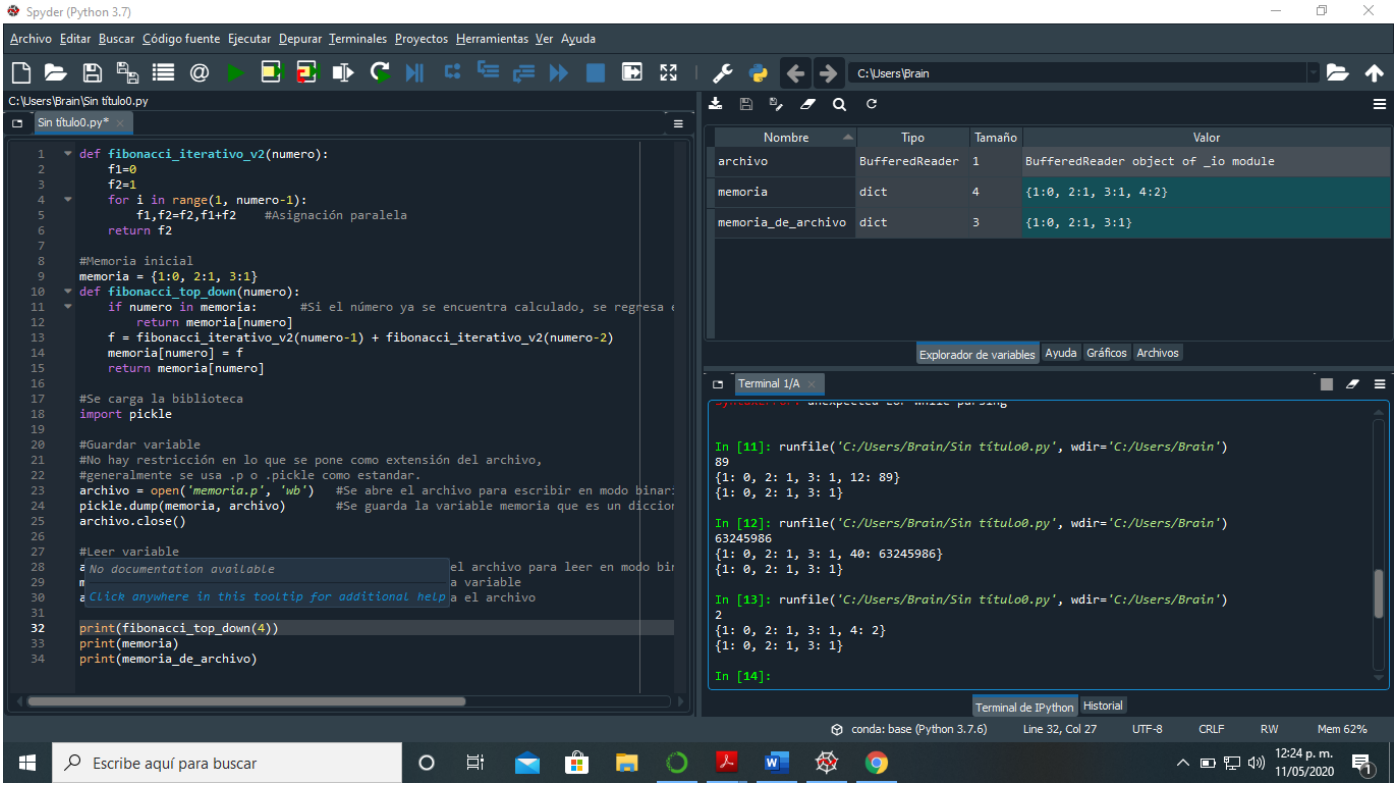
El cuarto ejemplo busca solucionar la sucesión de Fibonacci, para ello utiliza la función iterativa Fibonacci, y un diccionario para poder guardar operaciones que se realizaron con anterioridad, por ello parte del caso general n, y una vez resuelto el caso, comienza a guardar los valores para no repetir operaciones. En este caso se busca el asociar procesos para poder obtener una solución fluida o esquemática donde la solución final este relacionada con cada etapa.

- Diagrama de funcionamiento:



El ejercicio recorre a utilizar una función anteriormente definida, con la cual se pueden obtener los dos valores anteriores, que al sumarse dan el valor del término n en la sucesión de Fibonacci, este método pertenece a Top – Down porque parte de una parte general y busca la solución para el caso general, de modo que establece relaciones entre etapas de solución como lo fue el asociar funciones y utilizar la definición de la sucesión de Fibonacci, además de utilizar partes básicas. El conjunto de todo esto da una solución donde cada parte funciona si la anterior hizo su parte de forma adecuada.

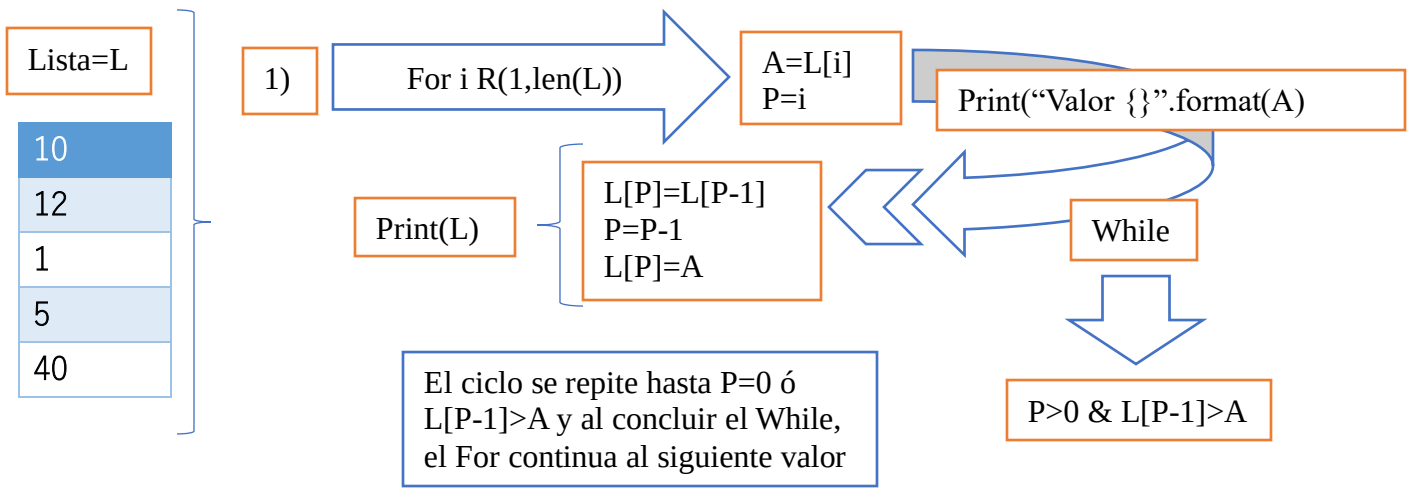
- Evidencia de implementación.



- Quinto ejemplo: Código – Incremental

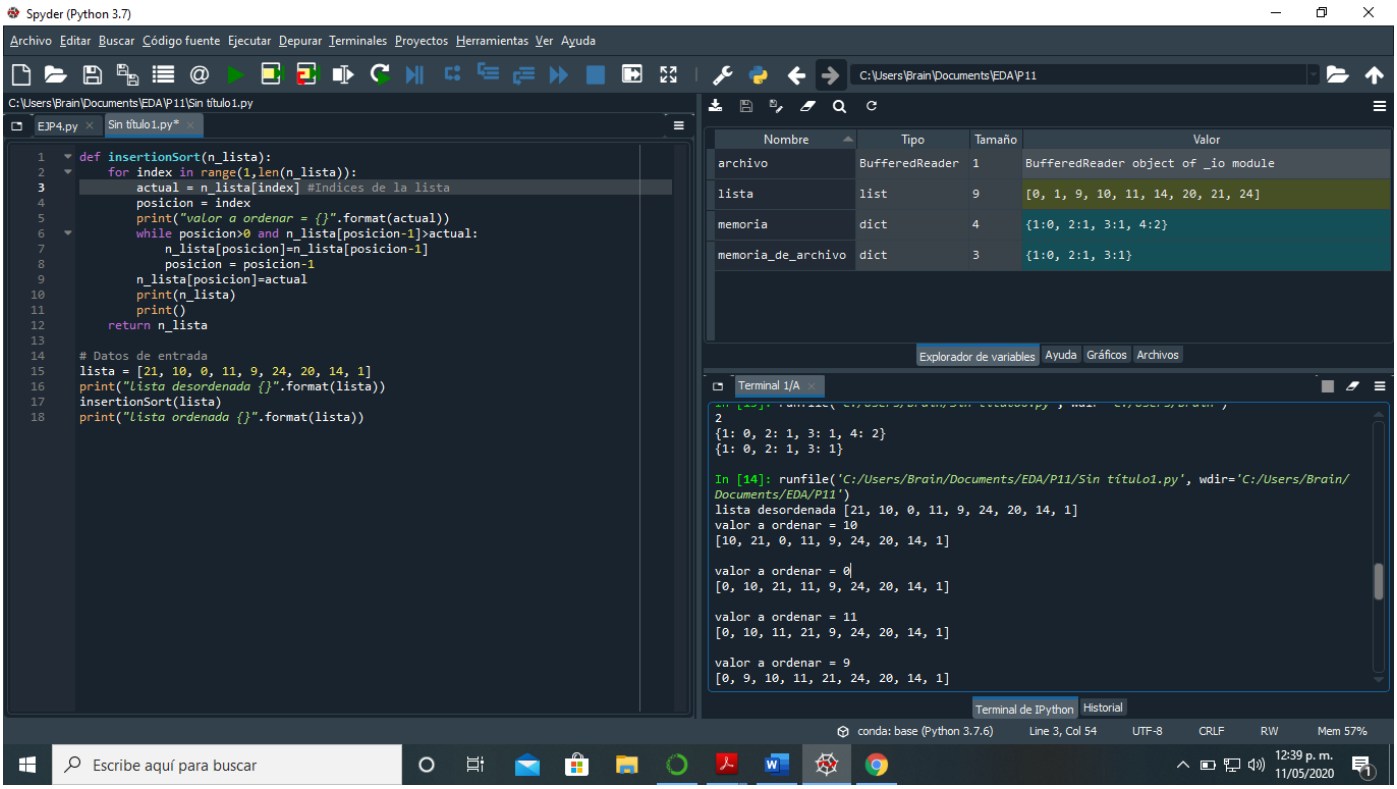
El programa busca el ordenar los elementos de una lista, los cuales están desordenados, este proceso y esta forma de diseño son progresivos debido a que la solución no es instantánea y depende de los valores que se generan en cada proceso de la solución, debido a que no es posible obtener una solución absoluta, ya que se van agregando valores tras cada tarea ejecutada.

- Diagrama de funcionamiento:



El código utiliza una lista como entrada, para poder organizarla de menor a mayor, los parámetros que utiliza son el for y el mas importante el while, que comprueba que todos los valores antecesores del actual sean menores y sino se cumple hace el intercambio, además pertenece a la categoría incremental debido a que los valores de las posiciones se van modificando, y se tienen verificar estos valores de nueva cuenta para poder realizar el ajuste que corresponda de acuerdo al código.
Podría organizarse por medio de Divide y Vencerás porque se podrían acomodar por parte los valores y después juntarlos, para que la final se compruebe que todos los valores están ordenados, y eso hace el siguiente ejemplo.

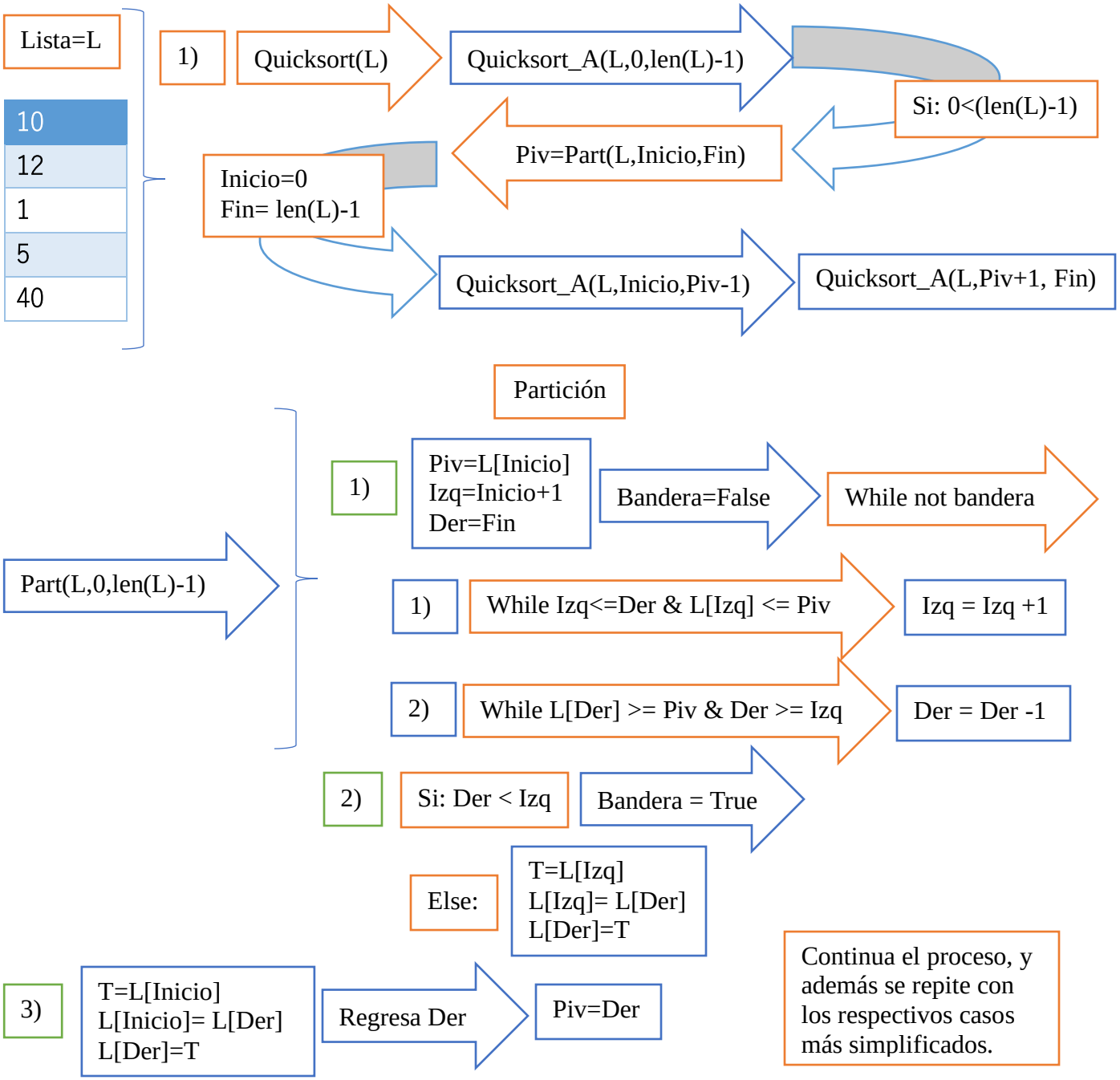
- Evidencia de implementación.



- Sexto ejemplo: Código – Divide y Vencerás

Este ejemplo busca el ordenar una lista de valores por medio de la división de sus elementos, esto es, crear problemas mas sencillos que sean posibles de resolver y obtener la solución final, que es el conjunto de las pequeñas soluciones. Para poder realizar este proceso es necesario tener valores “pivote”, “derecho”, e “izquierdo” que nos permitan realizar las operaciones en base a parámetros medibles, esto asegura que la solución sea eficaz y optima, además que se puedan verificar los pasos de la solución.

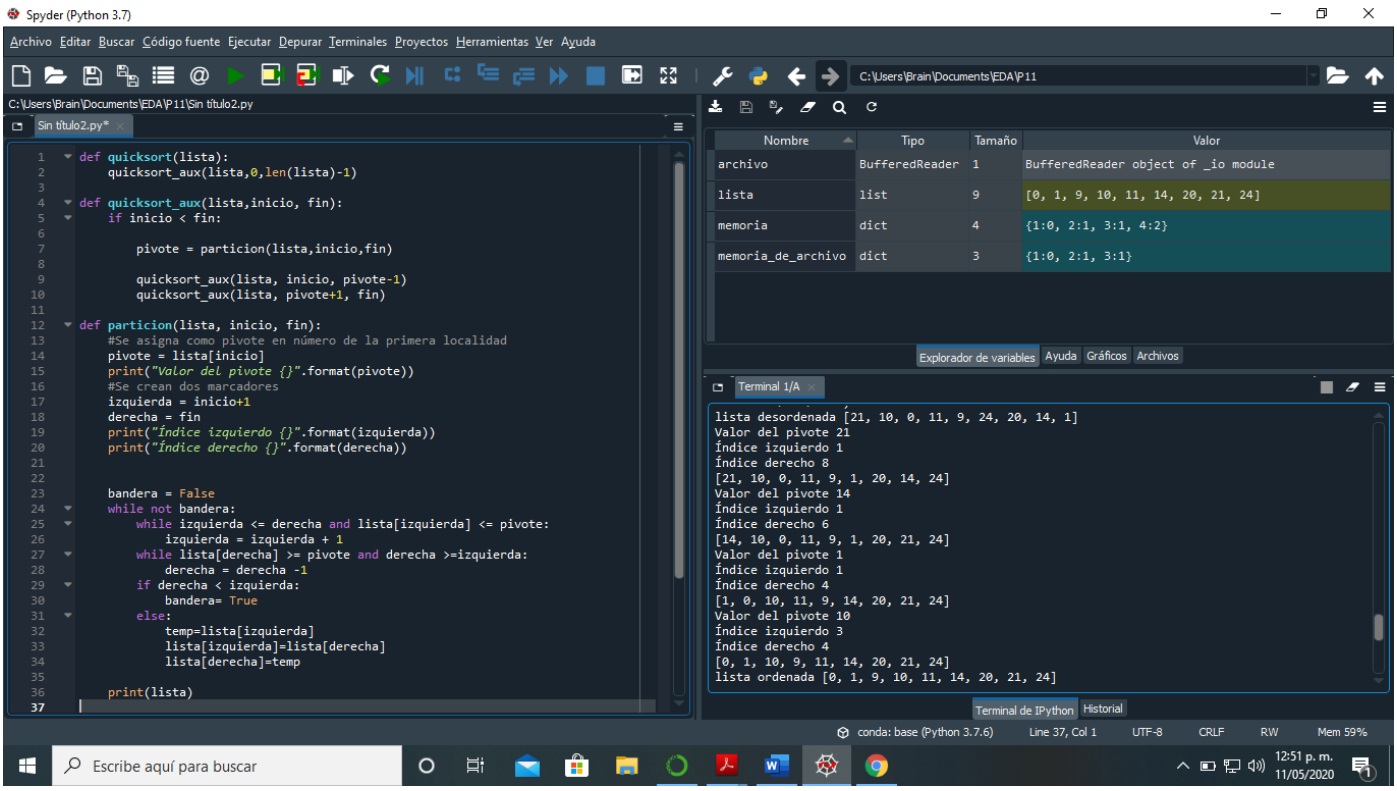
- Diagrama de funcionamiento:



El ejemplo utiliza funciones recursivas, además ciclos anidados, lo que significa que tiene una complejidad elevada, donde el principal proceso es el del primer análisis ya que a partir de este se comienza a segmentar la lista original, el proceso es relativamente simple, ya que una lista se parte en dos mas sencillas y a la vez cada una de estas se vuelve a partir, donde el parámetro para saber si se debe dividir es la función partición ya que el valor de pivote permite la recursividad.

Es Divide y Vencerás porque divide el ejercicio y simplifica cada vez que divide lo que, al momento de repetir este método, vuelve el problema mas sencillo y fácil de resolver.

- Evidencia de implementación.



Ejercicios de la práctica

- Ejercicio 1.

El primer ejercicio corresponde a la implementación de un algoritmo que permite organizar las actividades de una persona en un día, indicando cuando empieza y cuando termina, para implementar el código en python es necesario el colocar las respectivas funciones en termino del lenguaje, además el clasificar el código dentro de una de las categorías de construcción de algoritmos.

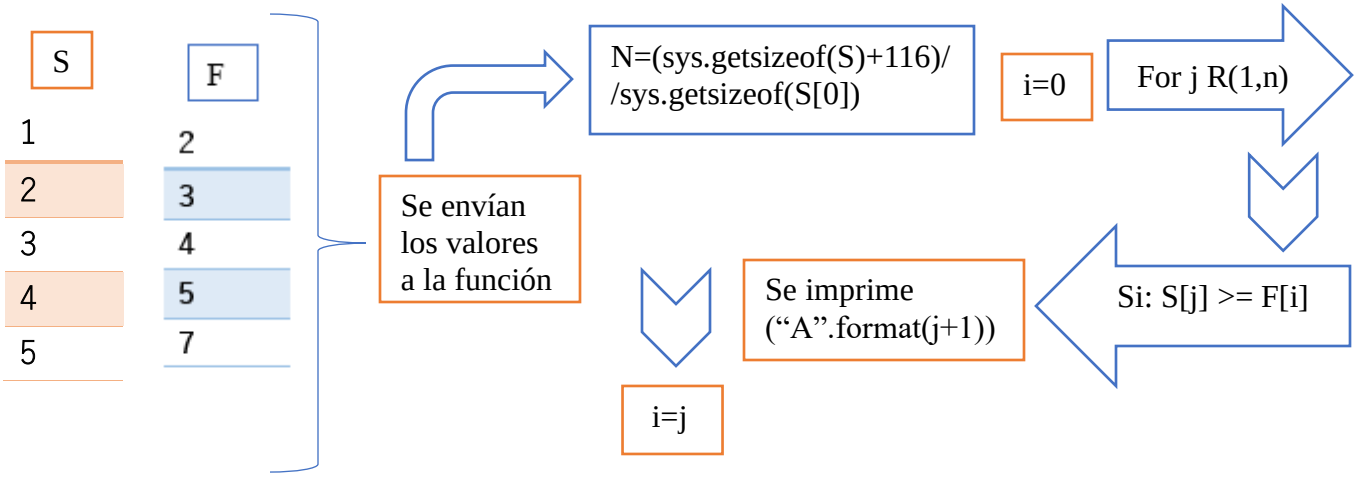
- Dificultades en el código.

El principal dilema que encontré al pasar el código a python fue el colocar el sizeof, debido a que en python no existe y para poder agregarlo es necesario el agregar una biblioteca. Por otro lado, el valor que tiene en python cada variable es distinto por ellos tuve que hacer ajustes en la cantidad de bytes.

```
a=sys.getsizeof(S)+116
b=sys.getsizeof(S[0])
n=a//b
```

Imagen 1

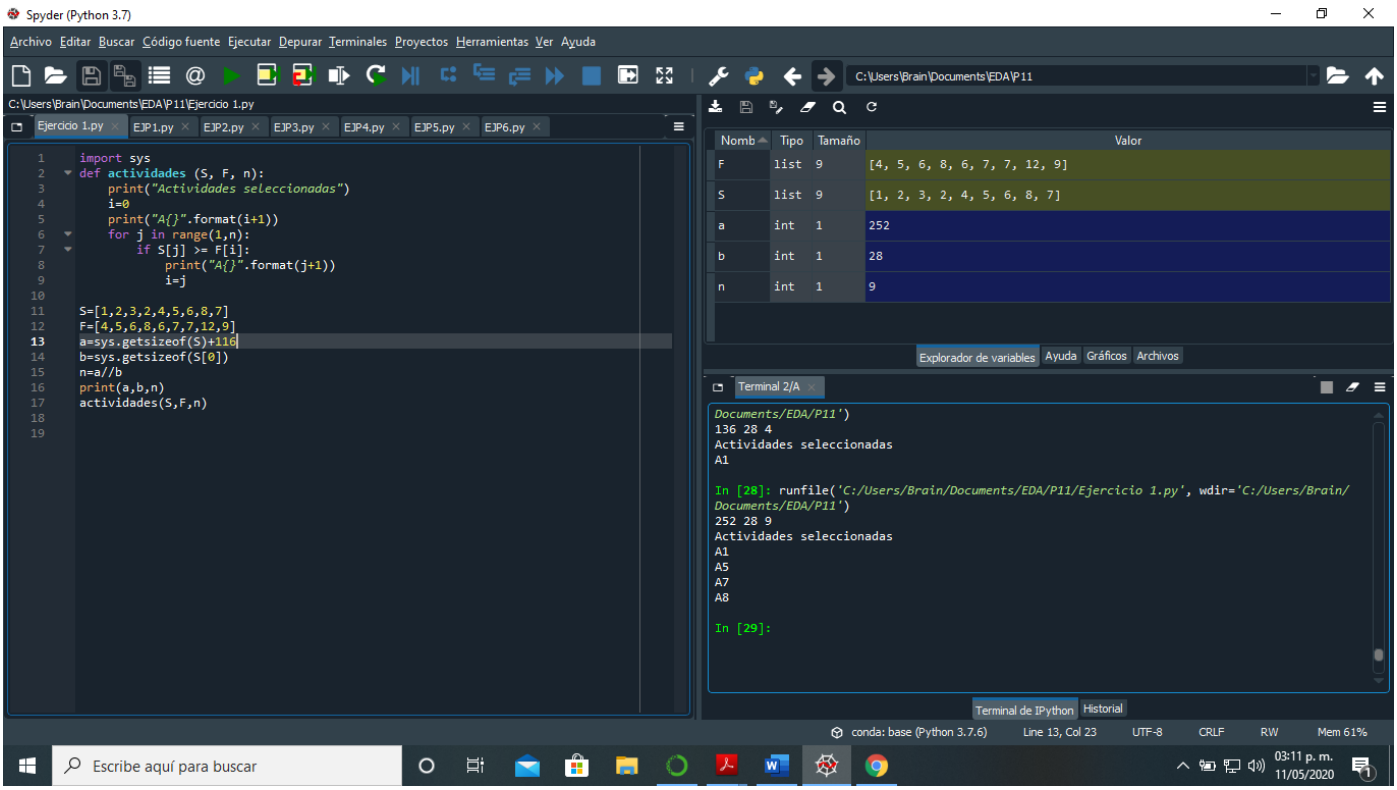
- Diagrama de funcionamiento.



- Relación con teoría.

En este caso el tipo de estrategia es Top – Down, debido a que se plantea de la generalidad de actividades que se pudieran tener en un día y comienza a relacionar las actividades con la hora de termino de las mismas, dando como resultado el horario donde idealmente tendrían que terminar cada una de las actividades, o incluso podría ser Bottom – Up, si se tuvieran actividades que no se pudieran omitir, estas serían las actividades base.

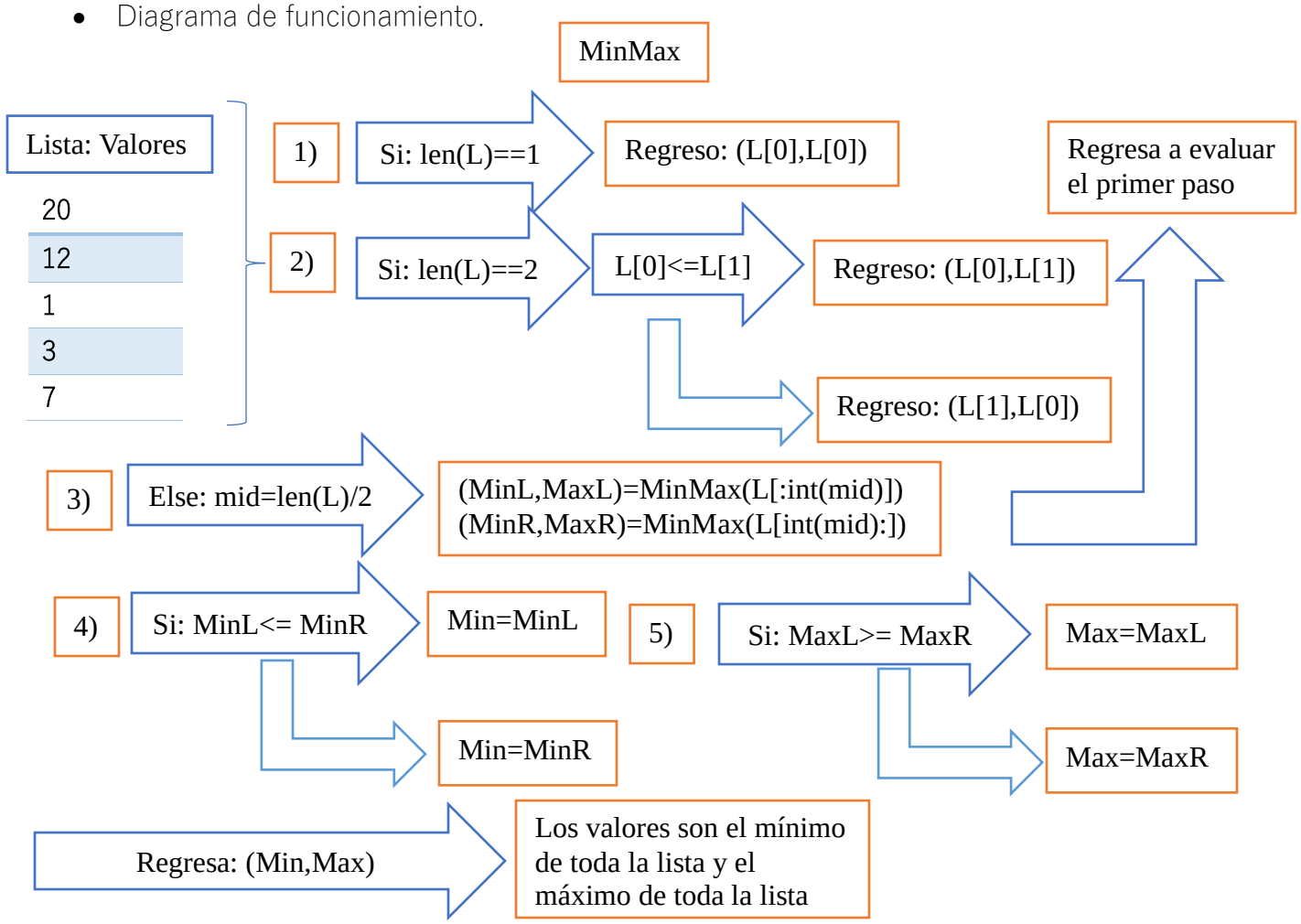
- Evidencia de implementación



- Ejercicio 2.

El segundo ejercicio consistía en pasar un código y analizar su funcionamiento, además de describir porque pertenece a la categoría de Divide y Vencerás.

- Diagrama de funcionamiento.

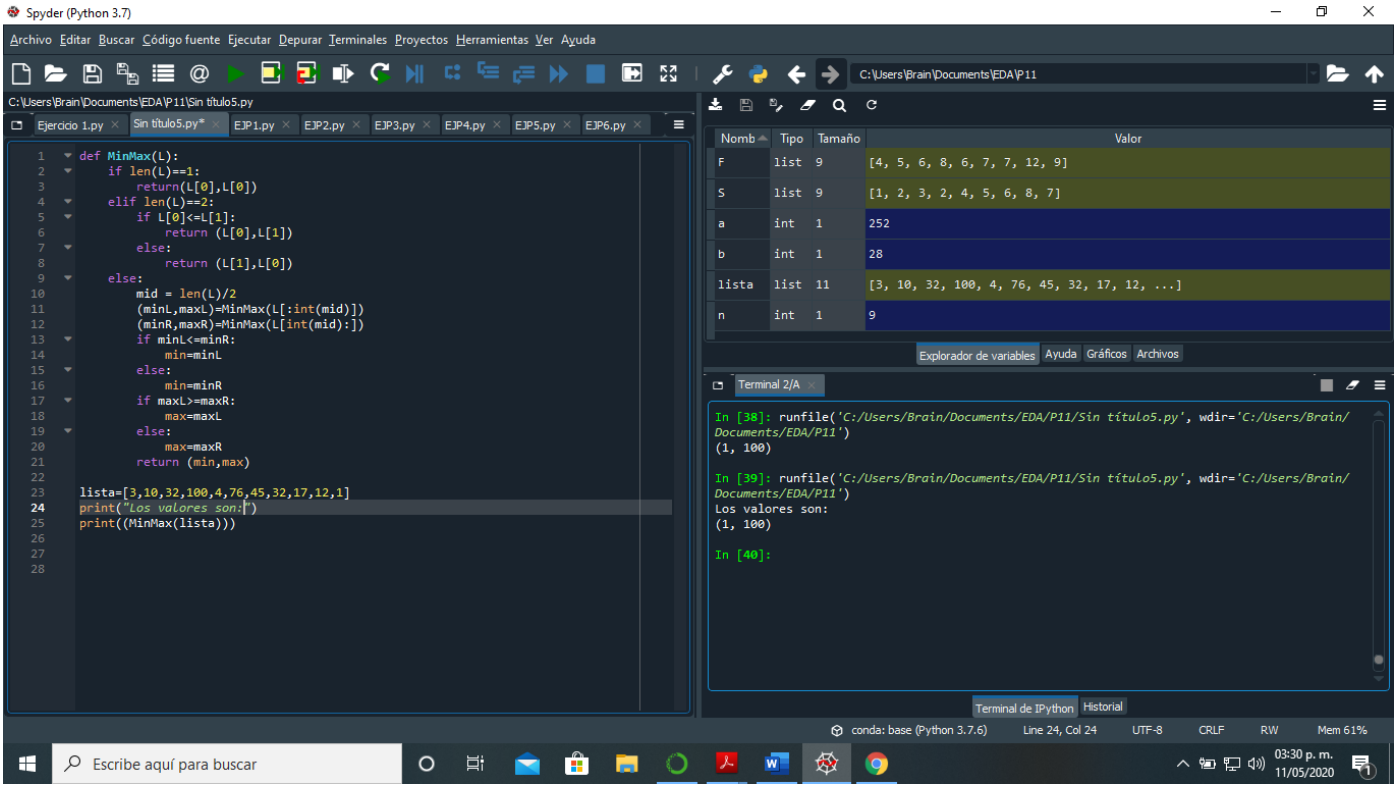


Los valores que se obtienen de la función son por medio de la simplificación de la lista de números, de hecho, se parece al Quicksort, debido a que se divide cada vez la lista al punto de que las soluciones de cada una de esas pequeñas partes en conjunto puedan dar solución al sistema, cuando estas se juntan se obtiene la solución final.

- Relación con teoría.

Pertenece a la categoría Divide y Vencerás, porque cada lista que es usada como entrada termina siendo separada de en dos partes, de forma que la división se vuelve a hacer hasta que se tenga una lista con 2 o 1 elemento, de esta forma se simplifican las operaciones por tener los casos mas simples al principio y obtener operaciones sencillas que resuelvan para cualquier caso.

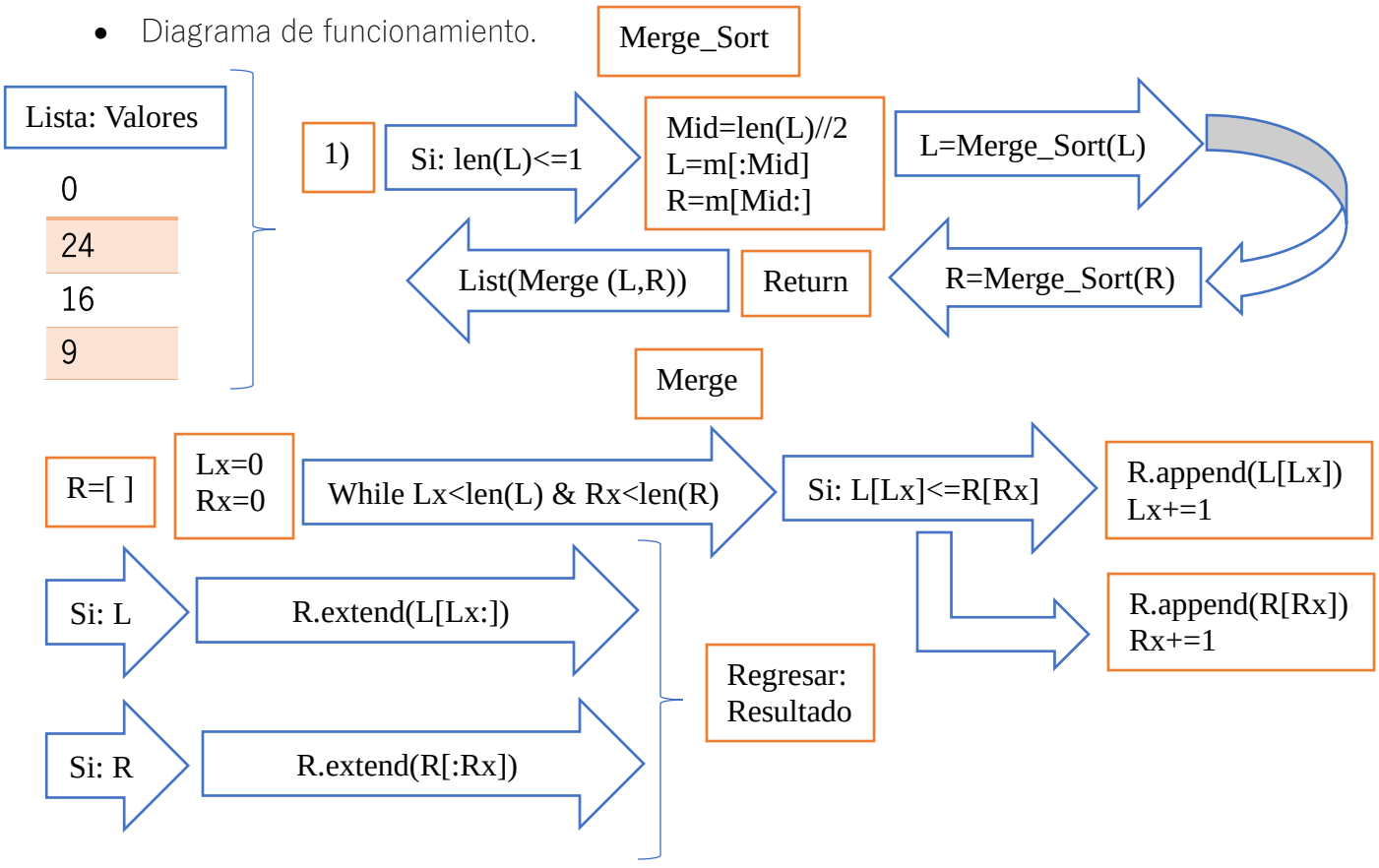
- Evidencia de implementación



- Ejercicio 3.

El último ejercicio consistía en pasar un código y analizar su funcionamiento, además de describir porque pertenece a la categoría de Divide y Vencerás.

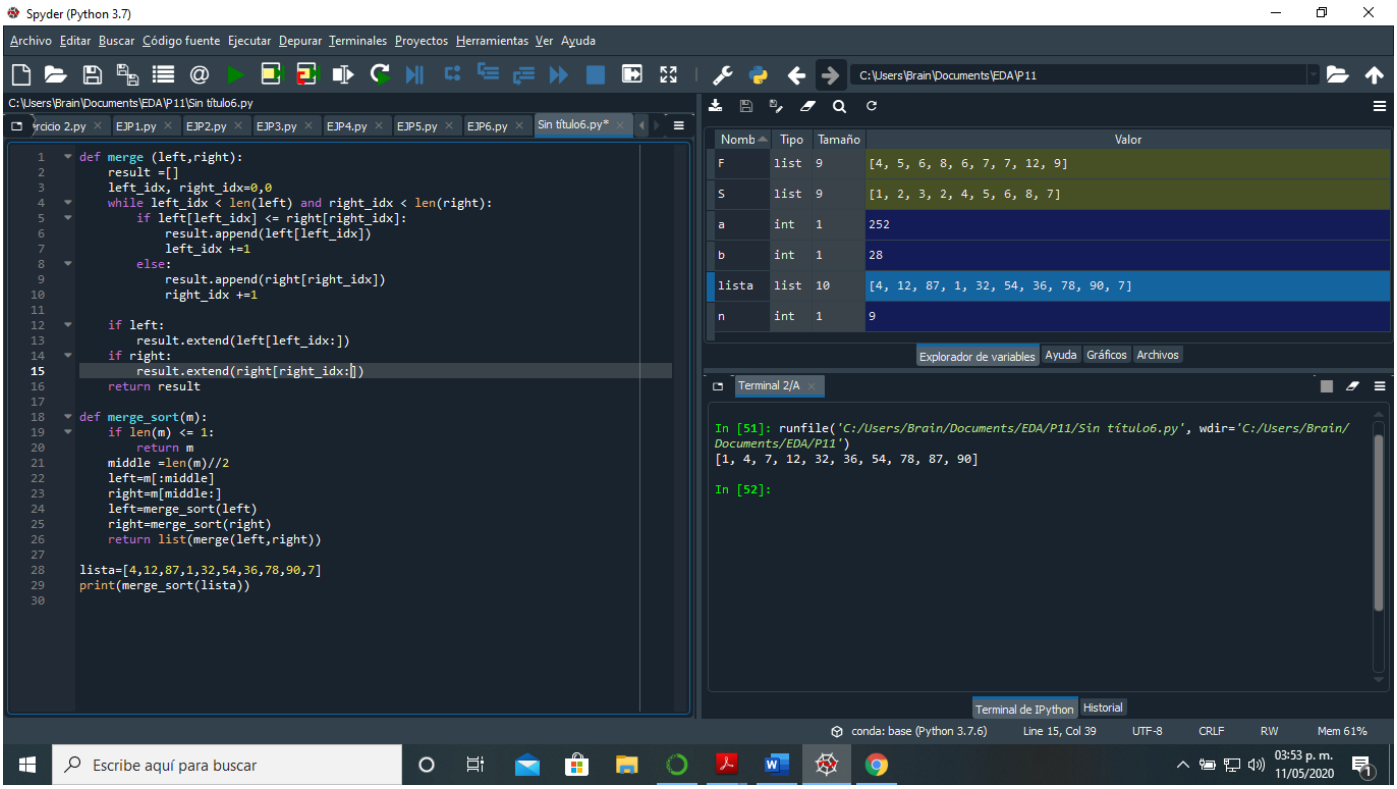
- Diagrama de funcionamiento.



- Relación con teoría.

El código pertenece a Divide y vencerás porque a partir de una lista de números, se comienza a dividir de forma secuencial, porque al momento que se divide la lista se generan dos nuevas y después se vuelven a generar dos nuevas de una de esas dos, lo que sucedes es una división sucesiva entre dos, pero la clave es como se mantiene pausada la operación right y left, al punto de tener una lista de len(L)==1, con ello se llega a un punto base y se obtiene un ordenamiento de los elementos.

- Evidencia de implementación



Conclusiones.

Lo que aprendí de esta práctica fue la forma en código que pueden tener algunas estrategias para construir algoritmos, además de poder comprender su funcionamiento, y como aplicarlas a los códigos para poder resolver problemas que conlleven un adecuado uso de los algoritmos, y por ultimo entendí reforcé el conocimiento que obtuve en las clases virtuales.