1. Escribe una función recursiva para **invertir** una **lista enlazada**.
2. Escribe un programa de **POO** para resolver el siguiente problema:

   Tienes dos jarras, una de 4 litros y otra de 3 litros. Ninguna de las jarras tiene marcas en ella. Hay una bomba que se puede utilizar para llenar las jarras con agua. ¿Cómo se pueden obtener exactamente dos litros de agua en la jarra de 4 litros?

3. El **triángulo de Pascal** es un triángulo numérico con números dispuestos en filas escalonadas de manera que

$$a_{nr} = \frac{n!}{r!(n-r)!}$$

   Esta ecuación es la ecuación para un coeficiente binomial. Se puede construir el triángulo de Pascal agregando los dos números que están, en diagonal, encima de un número en el triángulo. A continuación, se muestra un ejemplo del triángulo de Pascal.

```
        1
      1   1
    1   2   1
  1   3   3   1
1   4   6   4   1
```

   Escribe, siguiendo técnicas descritas en los apuntes, distintas versiones de un programa que imprima el triángulo de Pascal. El programa debe aceptar un parámetro que indique cuántas filas se imprimirán del triángulo.

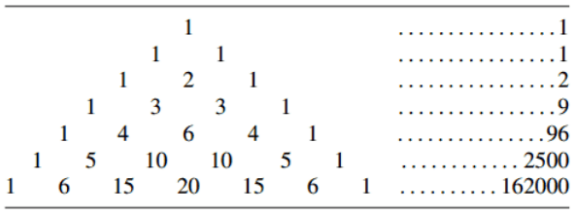
4. Usando el algoritmo de **programación dinámica** para dar las vueltas, encuentra el menor número de monedas que se podrían usar para completar unas vueltas de 33 céntimos. Además de las monedas usuales, supón que disponemos de una moneda de 8 céntimos. Realizar una simulación con distintas cantidades a devolver y distintos valores de monedas.
5. Supón que eres un científico de la computación/ladrón de arte que se ha colado en una galería de arte importante. Dispones sólo de una mochila para sacar las obras de arte robadas, que sólo puede contener W "kilos de arte"; no obstante, para cada pieza de arte conoces su valor y su peso. Escribe una función de **programación dinámica** para maximizar tus ganancias, con distintas capacidades de la mochila y distintos valores/pesos de los ítems. **He aquí un ejemplo del problema** que puedes usar para empezar: Supongamos que la mochila aguanta un peso total de 20. Tienes 5 ítems como sigue:

Ítem	peso	valor
1	2	3
2	3	4
3	4	8
4	5	8
5	9	10

   Aplicar alguna de las técnicas descritas en el tema…
6. Escribir un algoritmo que obtenga de **manera recursiva** el valor máximo del tramo [**izq**; **der**] de un vector **V**. Expresar el coste temporal del algoritmo.
7. Dado un vector **V** de longitud **n**, aplicando el esquema de diseño **Divide y vencerás y sin modificar V**, escribir una función que devuelva el valor que ocuparía la posición **k** (**p.e. posición de la mediana**) si el vector **V** estuviera ordenado. Calcular su complejidad temporal en función de **n** y expresarla en notación asintótica.
8. Generaliza el programa 2 de esta actividad:

   Tienes dos jarras, una de X litros y otra de Y litros, con X>Y. Ninguna de las jarras tiene marcas en ella. Hay una bomba que se puede utilizar para llenar las jarras con agua. ¿Cómo se pueden obtener exactamente X/2 litros de agua en la jarra de X litros?

9. A partir del programa del **triángulo de Pascal** desarrollado en el ejercicio 3, vamos a multiplicar los elementos de cada una de las filas:



Si ahora dividimos cada resultado obtenido al multiplicar entre el obtenido en la fila anterior obtenemos los siguientes valores:

{1, 2, 4.5, 10.666, …, 26.0417, 64.8}

Y ahora volvamos a dividir cada uno de los resultados de esa lista entre el anterior. Llegamos a los siguientes datos:

{2, 2.25, 2.370370, … , 2.44140625, 2.48832}

Parece que después de comenzar en 2 los números van subiendo poco a poco. Si avanzamos un poco, por ejemplo, por la zona del **n=1000**, el dato de la lista sería ya **2.71692**, que ya está más cerca del número ***e*** **=2.71818281**

Modifica el programa realizado en el ejercicio 3 para que calcule una aproximación del número ***e*** tal y como se explica en este enunciado. Aplicar alguna de las técnicas descritas en el tema…

---

10. Utilizando las fórmulas de desempeño de la tabla hash que se dan en el Tema 4, calcula el número promedio de comparaciones necesarias cuando la tabla está

    10% completa
    25% completa
    50% completa
    75% completa
    90% completa
    99% completa

    ¿En qué punto crees que la tabla hash es demasiado pequeña? Explícalo.

11. Elabora **estrategias alternativas** para elegir el valor pivote en el ordenamiento rápido. Reimplementa el algoritmo y haz un **Benchmark** de las mismas en conjuntos de datos aleatorios. ¿Bajo qué criterios estas estrategias funcionan mejor o peor que la estrategia de los apuntes?
12. Genera una pequeña lista aleatoria de números enteros. Muestra paso a paso cómo es ordenada dicha lista por los siguientes algoritmos:

    ordenamiento burbuja, en todas sus variantes
    ordenamiento por selección
    ordenamiento por inserción
    ordenamiento de Shell (con distintos valores de los incrementos)
    ordenamiento por mezcla
    ordenamiento rápido (decide sobre el valor pivote)

13. Diseña un experimento aleatorio (**Benchmark**) para probar la diferencia entre una búsqueda secuencial y una búsqueda binaria, en todas sus variantes, en una lista de 200 enteros.
14. Utiliza **todas** las funciones de búsqueda binaria (recursiva e iterativa). Genera una lista ordenada aleatoria de números enteros y realiza una prueba de referencia (**Benchmark**) para cada función. ¿Cuáles son sus resultados? ¿Puedes explicarlos?
15. Implementa el método tamano **(__len__**) para la implementación del TAD Vector Asociativo o mapa de las tablas hash, de manera que sea $\boldsymbol{O(1)}$**.**
16. ¿Cómo puedes eliminar ítems de una tabla hash que utiliza encadenamiento para la solución de colisiones? ¿Qué tal si se usa direccionamiento abierto? ¿Cuáles son las circunstancias especiales que deben manejarse? Implementa el método **eliminar** para la clase TablaHash que utiliza encadenamiento.

17. En la implementación de Vector Asociativo de las tablas hash, se eligió que el tamaño de la tabla hash fuera **11**. Si la tabla se llena, ésta debe agrandarse. Implementa el método agregar para que la tabla se redimensione automáticamente cuando el factor de carga alcance un valor predeterminado (puedes decidir el valor con base en su apreciación de la carga en función del desempeño).
18. Implementa la **prueba cuadrática** como una técnica de **rehash**.
19. Utilizando un generador de números aleatorios, crea una lista de 500 enteros. Realiza una prueba de referencia usando al menos 3 de los algoritmos de ordenamiento de los apuntes. ¿Cuál es la diferencia en la velocidad de ejecución? ¿Cuáles son tus conclusiones?
20. Un ordenamiento burbuja puede modificarse para que "burbujee" en ambas direcciones. La primera pasada mueve la lista hacia "arriba", y la segunda pasada la mueve hacia "abajo". Este patrón alternante continúa hasta que no son necesarias más pasadas. Implementa esta variación y describe en qué circunstancias podría ser apropiada.
21. Realiza una prueba de referencia (**Benchmark**) para un ordenamiento de Shell, utilizando diferentes conjuntos de incrementos en la misma lista (**los expresados en los apuntes y dos más…**).

Exámenes años anteriores

Problema 1.- Utilizando códigos vistos en el **Tema 1** e #include <chrono> para obtener los tiempos de ejecución en nanosegundos, implementa **en C++** una prueba de referencia (Benchmark) con las búsquedas **Binaria iterativa**, **por Salto**, **Fibonacci**, **Exponencial** e **Interpolación**.

Os recuerdo que para hacer un Benchmark (una prueba de referencia) con métodos de búsqueda, no me puedo limitar a buscar un elemento, sino que tengo que hacer varios casos: **Buscar al principio de la lista, buscar un valor hacia el medio y buscar un valor del final** y sacar los tiempos promedio de los tres.

Explicar los resultados obtenidos

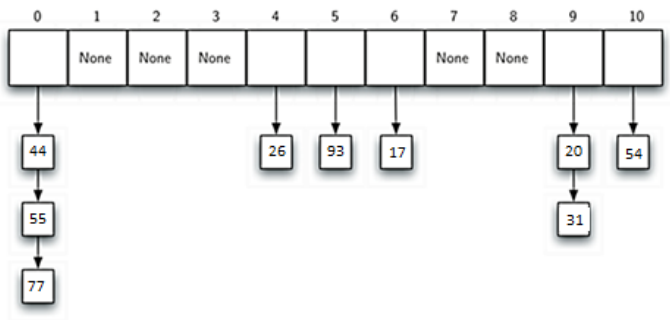
Problema 2.- Escribe un programa en Python que, utilizando las clases desarrolladas en los apartados a) y b), demuestren el funcionamiento de cada una de las implementaciones.

a). - Implementa **TAD Vector Asociativo o TAD mapa** con direccionamiento abierto, de manera que:

- El método **tamano** (__len__) sea O(1).
- Disponga de un método para **eliminar** elementos.
- Disponga de un método para **agrandar** la tabla al siguiente primo al doble del tamaño, cuando esté al 90% de ocupación.

b). - Implementa **TAD Vector Asociativo o TAD mapa** con encadenamiento, de manera que los elementos de la Tabla hash sean listas enlazadas ordenadas (**Class_ListaOrdenada** de la actividad 2) y disponga de todos los métodos necesarios del **TAD mapa** con direccionamiento abierto.

- El método **tamano** de la Tabla Hash (__len__) será O(1).
- **ListaOrdenada** tendrá implementados los métodos **tamano** (O(1)), **borrar**, **indice**, **extraer** y **extraer(pos)**



Problema 3.- La implementación del **TAD lista enlazada no ordenada** se ha desarrollado en los apuntes de la asignatura.

Una implementación alternativa se conoce como una **lista doblemente enlazada**. En esta implementación, cada nodo tiene una referencia al siguiente nodo (comúnmente llamado **siguiente**) así como una referencia al nodo precedente (comúnmente llamado **anterior**). La Cabecera de la lista también contiene dos referencias, una al primer nodo en la lista enlazada y una al último.

Implementar el **TAD Lista Doblemente enlazada** en Python (**especificación informal incluida**) y codificar un ejemplo que use dicha implementación.