1. **Caja Negra**
   Supongamos que tenemos acceso a un algoritmo, denominado Caja Negra, del cual sólo conocemos sus resultados, contesta: **sí** o **no**. Si se le da una secuencia de **n** números enteros y un entero **k**, el algoritmo responde **sí** o **no**, dependiendo si existe un subconjunto de esos n umeros cuya suma sea exactamente k.

   Mostrar cómo usar esta **Caja Negra O(n)** veces en un proceso que encuentre el subcon-junto en cuestión, si es que existe, donde **n** es el tamaño de la secuencia.
   Respuesta
   Primero mostraremos como funciona la **Caja Negra**; el problema dice que necesitamos encontrar un conjunto A, tal que solución $=\{X_i|x_0 + ... + x_n = k\}$ y sea S la secuenciade números enteros, solución$\subseteq P(s)$.
   Sabemos que la caja negra dice que **si** en el caso de que haya al menos un subconjunto, **B** tal que $B \subseteq P(s)$, todo esto es posible por 2^n subconjuntos de una $'P(s)$ y alguno de estos conjuntos pueda generar K y de no ser el caso de que sea pueda generar K, se regresara como respuesta **No**.
   Ahora solo tenemos dos posibles casos, en el que el caso sea **no**, en caso de que la secuencia original sea no, significa que no hay una K y por ende finalizamos. En el caso de que la respuesta sea **si**, significa que existe un subconjunto que es capaz de crear una K, pero sin saber cual es la solución.
   Entonces supongamos que existe una solución para la secuencia S. Sea $A = S$ en la caja negra(A,K) la respuesta es **si**, si existen ciertos números enteros de A que sumados den **k**, entonces veamos si le quitamos el ultimo elemento de la caja negra(A=$\{X_n\}$,K), tenemos dos posibles respuestas **si** a la caja negra le quitamos el ultimo elemento (X_n), existen un subconjunto de A que es capaz de poder generar a K, por ende X_n no es necesario, pero en el caso de que **no** se pueda generar a K sin X_n, pero sabemos que si existe una solución, entonces X_n es parte del conjunto solución, además tenemos que actualizar el valor de k entonces tendríamos que: A = (A-$\{X_n\}$, K-X_n).
   Hipótesis de Inducción:
   Resolver el ejercicio para (n-1) elementos.
   Caso Base:
   n=1, entonces A=$\{X_0\}$, X_0=K sabemos que se puede resolver gracias a la caja negra.
   Paso Inductivo:
   Supongamos que nuestra secuencia es de n elementos, entonces usamos la Caja Negra para ver si hay solución, si no existe solución acabamos y si la respuesta es si entonces ∃B tal que B⊆'P(s), entonces B=$\{x|x_0 + ... + x_m = k\}$. Recordando que si la respuesta es **no**, sabemos que existe una solución por lo que X_n forma parte de la solución, pero aunque quitemos X_n no es necesario, entonces necesitamos actualizar A, puesto que ya hemos evaluado X_n por ende tendríamos A=A-$\{X_n\}$, entonces A tiene (n-1) elementos y por **H.I.** sabemos que se puede resolver el problema de (n-1).
   Entonces la caja negra se puede aplicar n veces, puesto que vamos recorriendo el conjunto de forma lineal, hasta que esta sea vacía y en cada iteración eliminamos el último elemento, sin repetir, todo esto usando la caja negra y sin importar su respuesta. En el peor de los caos todos los elementos eran parta de la caja negra, es decir parte de la solución y tuvimos que recorrer todos los elementos.

```
//Pre cond: Una secuencia, finita de números enteros y un enetero K
//Psot cond: Un conjunto solución, tal que solucion={x|x_0+...+x_n=k}
 class ConSolucion{
    conSolucion(S:secuencia,K:enetero)
    if cajaNegra(s,K)== No
        return // se termina la ejecución
    var string n
    var int real
    var A=S
    var solucion = conjunto
    while (A!= null || k > 0 || n > 0) {
        x= A.ultimoelemento()
        A= A.elemeultimoelemto()
        if(cajaNegra(A,K)== No){
            solucion.agregar(real)
            k = K-real
        }
    n--
    }
}
```

2. **Los Dos menores elementos de un Conjunto.**

   Problema μ: Dada una secuencia **S** = $[x_1, x_2, ..., x_n]$ encontrar a los dos menores elementos de S, usando la menor cantidad posible de comparaciones.

   - a) Diseñar, usando Inducción Matemática, un algoritmo que resuelva el problema μ.
   - b) Determinar el número de comparaciones que realiza el algoritmo.

   Respuesta
   Debemos de resolver con la menor cantidad posible de comparaciones, resolviendo por cada iteración del algoritmo. Haremos inducción sobre $n \in N$ que es nuestro tamaño de nuestra secuencia.
   Caso Base:
   Sabemos que $n < 2$
   Hipótesis de Inducción
   Debemos de encontrar a los dos menores elementos en la secuencia que tenemos de tamaño n, que deben de ser a,b tal que $a < b$
   Paso Inductivo
   Supongamos que tenemos una secuencia S de tamaño n+1 y por HI sabemos que $a, b \in S[1, 2, 3, ..., n]$, entonces $C = S[n+1]$, entonces podemos decir que a y b son los menores elementos posibles. El número de comparaciones que hacemos por cada llamada se hacen a los más 3 comparaciones, pero en el caso base es solo uno, además de que no se toman en cuenta si la lista es vacía o de un solo elemento, entonces si n es del tamaño de la lista tenemos que: $comparaciones(n) = 3(n-2)+1$, donde $n \geq 2$ y este es el peor de nuestros casos.

```
//Pre cond: |s|>= 2
//Post cond: 2> elementos de n
 class Menores{
    (n:int, S.Array)
    if (n==2)
        {
            a->S[1]
            b->S[2]
            return a,b si a<=b, si no b,a
        }
    else
        {
            c->S[n]
            a,b-> Menores(S, n-1)
            if( a<=b<=c)
            {
                return a,b
            }
            else if(b<=c)
            {
                return b,a
            }
            else
            return b,c
        }
}
```

3. Resolver **uno** de los siguientes problemas:
   A. **Partición**
   Dada una lista **L** de **n** enteros positivos y distintos, particionar (dividir) la lista en dos sublistas L1 y L2, cada una de tama no n/2 tal que: **L**= L1 L2; L1 L2 = ; se satisface, adem as, que la diferencia entre las sumas de los enteros en las dos listas sea m ınima.

   - a) Diseñar, usando inducción matemática, un algoritmo para este problema.
   - b) Determinar la complejidad del algoritmo propuesto.

   Pueden suponer que n es múltiplo de dos.
   Respuesta
   Primero empezamos comparando dos elementos de la lista principal, luego metemos el menor elemento a la lista de los pequeños y el mayor de los mayores elementos.Verificando a su vez que el menor sea el menor elemento de la lista de menores y el mismo caso para la lista de mayores.
   Caso Base: n=2
   Tenemos dos listas una Min, donde se compararan los dos elementos y meteremos el menor de estos a Min y otra Max. Cabe aclarar que estas listas están vacías y supongamos que la lista original tiene los elementos [7,2,3,5], además de que la listas Min y Max tienen la diferencia entre estas es la menor elemento.
   Segundo Caso Base: n=4
   Tenemos que recordar que debemos de hacer las comparaciones del mínimo elemento de Min y el máximo elemento de Max para que el resultado sea correcto. Ahora recordando la lista original que era [7,2,3,5] y esta la dividimos en dos aleatoriamente.

   Uno=[7,2] y Dos=[3,5]

   Ahora comparamos los dos primeros elementos de las listas que serían **7** y **3**, entonces comparamos y vemos que el 3 es el menor y 7 es el mayor elemento, los agregamos a sus respectivas listas **Min=[3]** y

Max=[7]. Luego hacemos la segunda comparación con los siguientes elementos de la lista Uno y Dos que son **2** y **5**, los comparamos y vemos que el 2 es menor y el 5 es mayor, antes de agregarlos a Min y Max debemos de volver a comparar estos elementos con los elementos de Min y Max, es decir vemos quien es **menor** entre **2** y **3**, obteniendo como resultado **Min=[2,3]**. El mismo caso se aplica para Mac, comparamos **5** y **7**, obteniendo **Max=[7,5]**

Hipótesis de Inducción

Supongamos que para las listas **n**, con n par, se puede obtener las listas Min y Max que cumplen esta propiedad.

Paso Inductivo

Sabemos que nuestra lista es de **n+2** enteros positivos y distintos. La cuál esta propiedas se cumple por la HI, tambien por la HI sabemos que tenemos las listas Min y Max, las cuales tamien cumplen la propiedad de ser n+2, es decir tenemos la longitud y diferencia menor de estas. Por último tenemos que recordar las comparaciones que hicimos, el primer elemento de Uno y el primer elemento de Dos, luego repetimos el mismo paso pero ahora con el segundo elemento de Uno y Dos, los comparamos para ver cual elemento es menor y cual es mayor, después hacemos una comparación para ver quien es el menor elemento de Min y cual es el mayor elemento de Max, por ende podemos decir que la lista n+2 cumple con la propiedad. Falta mostrar cual es la complejidad de este algoritmo, primero debemos tomar en cuenta las comparaciones como operaciones elementales y también agregar elementos a las lisas, recorremos la lista original una vez, agarrando así dos elementos aleatorios de esta y haciendo comparaciones entre estas, este paso por cada comparación de elementos que hagamos y dos más por cada elemento mínimo y máximo que encontremos en cada lista. Por lo tanto obtenemos 4 comparaciones por cada comparación o iteración del algoritmo, es decir tenemos $\frac{n}{2}$ iteraciones, obteniendo un total de 2n comparaciones, que nos da como resultado de $O(n)$ creando dos listas(Min y Max) que tienen $\frac{n}{2}$ elementos, con lo que el espacio es $O(n)$.

C. **Distancias**

Sea **T=(V,A)** un árbol binario con **n** vértices. Suponga que el árbol **T** esta representado por su lista de adyacencias.

Construir una matriz M de n $\times$ n tal que el elemento M[i, j] sea igual a la distancia entre los vértices vi y vj.

- a) Diseñar, usando inducción matemática, un algoritmo que construya tal matriz.
- b) Determinar la complejidad del algoritmo propuesto.