**Punto A**

Escriba una función recursiva que ordene de menor a mayor una lista de enteros basándose en la siguiente idea: coloque el elemento más pequeño en la primera ubicación, y luego ordene el resto del arreglo con una llamada recursiva.

**Entrada:**

Un arreglo con L de longitud con números enteros en cada index.

**Salida:**

Un arreglo con L de longitud con números enteros en cada index ordenado de menor a mayor proporcional a su posición.

**Estrategia:**

Si el numero en la posición i+1 es menor al número en la posición i mientras que i sea menor a L, entonces se retorna el mismo arreglo con el orden de la secuencia modificada

**Invariante:**

* Iniciación: Antes de que comience cualquier llamada recursiva, la lista original “sequence” debe tener elementos para ordenar.
* Estabilidad: La lista “seq\_modified” se actualiza en cada iteración para reflejar la lista original excluyendo los elementos mínimos ya ordenados.
* Terminación: La función recursiva termina cuando la longitud de la lista “sequence” es menor o igual a 1. En este punto, la lista “sequence” ya está ordenada y la recursión se detiene, devolviendo la lista ordenada.

**Casos de Prueba:**

|  |  |  |
| --- | --- | --- |
| **Entrada** | **Justificación** | **Salida** |
| [-4, -1, 2, 8, 23] | Ordenado | [-4, -1, 2, 8, 23] |
| [18,12,4,0,-2,-3] | Reverso | [-3, -2, 0, 4, 12, 18] |
| [1,24,12,-5,-16,2,0] | Caso general | [-16, -5, 0, 1, 2, 12, 24] |

1. **def** sortRec2(sequence = []):
2. **if** len(sequence) <= 1:
3. **return** sequence
4. ele = [min(sequence)]
5. sequence.remove(ele[0])
6. **return** ele + sortRec(sequence)
7. **def** sortRec(sequence = []):
8. ele = []
9. seq\_modified = sequence[:]
10. **if** len(sequence) > 1:
11. ele = [min(sequence)]
12. seq\_modified.remove(ele[0])
13. **return** sequence **if** len(sequence) <= 1 **else** (ele + sortRec(seq\_modified))

**Punto B**

Encontrar una manera de sumar los elementos de un sub-arreglo a limitado por los índices i y j de un arreglo L de manera recursiva.

**Entrada:**

La entrada del problema es un arreglo L de longitud n, así como 2 enteros ‘i’ y ‘j’ que limitan al arreglo L en un sub-arreglo a tales que 0<=i<=j<=len(L).

**Salida:**

La salida es un entero que es la suma de los elementos del sub-arreglo dentro del rango mencionado.

**Estrategia:**

La estrategia para este problema fue implementar una función que tomara el sub-arreglo a para después ser llevada a otra función encargada de sumar sus elementos recurrentemente. La idea fue tomar el primer valor del sub arreglo y sumarlo con la función conquistar(a) en a[i:]

**Invariante:**

* Iniciación: la suma no se ve afectada por la determinación de i,j
* Estabilidad: la suma se mantiene tras cada iteración
* Terminación: no hay más elementos para sumar.

**Casos de prueba:**

|  |  |  |
| --- | --- | --- |
| Entrada | Justificación | Salida |
| L = [1,2,3,4,5,6,7,8,9,10] i =2 j = 7 | Rango convencional | 33 |
| L = [1,2,3,4,5,6,7,8,9,10] i=1 j=1 | Rango 1 | 2 |
| L = [1,2,3,4,5,6,7,8,9,10] i=2 j= 1 | i>j | 0 |
| L = [] i=n j=m | Rango vacío | 0 |

1. **def** dividir(lista, i, j):
2. **if** len(lista)==0:
3. **return** 0
4. **elif** i == j:
5. k = lista[i]+lista[i]
6. **return** k
7. **elif** j>i:
8. **return** 0
9. **else**:
10. mid = (i + j) // 2
11. left = lista[i:mid + 1]
12. right = lista[mid + 1:j + 1]
13. suma = conquistar(left) + conquistar(right)
14. **return** suma
15. **def** conquistar(a):
16. **if** len(a) == 1:
17. **return** a[0]
18. **else**:
19. **return** a[0] + conquistar(a[1:])

**Punto C**

Escribir una función y un programa que encuentre la suma de los enteros positivos pares desde N hasta 2.

**Entradas:**

Un numero entero positivo n cualesquiera

**Salidas:**

Un numero entero positivo par

**Estrategia:**

Función a trozos que toma como casos base si n = 0 o si n = 2 para luego con ayuda de un contador toma el caso que no cumple con las condiciones base y si este es par entonces al contador se le suma el número que el usuario introdujo y lo sume recursivamente quitándole 2 por iteración para luego, cuando el numero sea 2 entra en el caso base y devuelve el contador sumado 2, si el número es par.

**Invariante:**

* Iniciación: la suma de los números pares no se ve afectada si la entrada es impar
* Estabilidad: sumar todos los números pares tales que sean diferentes a 2
* Terminación: al ser 2 se suma solo esta cantidad y finaliza el programa

**Casos de Prueba:**

|  |  |  |
| --- | --- | --- |
| **Entrada** | **Justificación** | **Salida** |
| n = 0 | Caso vacío | 0 |
| n = 1 | Caso único | 0 |
| n = 2 | Caso único | 2 |
| n = 7 | Caso regular | 12 |

1. n = int(input("n="))
2. **def** suma(n):
3. elite = 0
4. **if** n == 2:
5. elite = 2 + elite
6. **return** elite
7. **elif** n == 0:
8. **return** elite
9. **elif** n % 2 != 0:
10. elite = suma(n - 1)
11. **return** elite
13. **elif** n%2 == 0:
14. elite= n + suma(n - 2)
15. **return** elite
16. suma(n)

**Punto D**

Dada una función recursiva para MCD, escriba un programa que le permita al usuario ingresar los valores para M y N desde la consola e imprima el valor para el MCD.

**Entradas:**

Dado dos números enteros positivos M y N cualesquiera

**Salidas:**

Un numero entero positivo dado igual a N cuando el residuo entre M y N es igual a cero.

**Estrategia:**

Implementar un programa que a partir de las entradas M y N calcule el Máximo Común Divisor usando el algoritmo de Euclides.

**Invariante:**

Dado un arreglo [M,N] retorna [N, M%N] hasta que [M%N]==0

**Casos de Prueba:**

|  |  |  |
| --- | --- | --- |
| **Entrada** | **Justificación** | **Salida** |
| 17  23 | Por lo menos algún número primo | 1 |
| 28  16 | Caso general | 4 |

1. **def** mcd(m,n):
2. **if** n == 0:
3. **return** m
4. **else**:
5. **return** mcd(n, m % n)
6. num1=int(input("Digite su primer numero:\n"))
7. **while** num1<0:
8. num1=int(input("Diga un numero entero positivo. Digite su primer numero:\n"))
9. num2=int(input("Digite su segundo numero:\n"))
10. **while** num2<0:
11. num2=int(input("Diga un numero entero positivo. Digite su segundo numero:\n"))
12. **print**(f"El maximo comun divisor entre los numeros {num1} y {num2} es {mcd(num1,num2)}")

**Punto E**

Programe un método recursivo que transforme un número entero positivo a notación binaria.

**Entradas:**

Un numero entero positivo

**Salidas:**

Una cadena de números compuestos por unos y ceros equivalentes al número de entrada en binario

**Estrategia:**

Se procede a dividir el número de entrada sucesivamente entre dos hasta que su cociente sea igual a uno. En cada iteración de esta operación, se registra el residuo obtenido y se almacena en una lista. Posteriormente, se imprime el reverso de esta lista.

**Invariante:**

El cociente del número de entrada siempre será dividido entre dos

**Casos de Prueba:**

|  |  |  |
| --- | --- | --- |
| **Entrada** | **Justificación** | **Salida** |
| 15 | Siempre es divisible entre dos | 1111 |
| 16 | Nunca es divisible entre dos | 10000 |
| 37 | Caso general | 100101 |

1. n = int(input("n="))
2. **def** binario(n):
3. **if** n//2==0:
4. lista.append("1")
5. **else**:
6. **if** n%2==0:
7. lista.append("0")
8. **else**:
9. lista.append("1")
10. **return** binario(n//2)
11. lista=[]
12. binario(n)
13. **print**(''.join(lista[::-1]))

**Punto F**

Dado una lista, retornar el reverso de esa lista de manera recurrente

**Entradas:**

La única entrada en este caso es una lista a con elementos i\*n tales q i>0

**Salidas:**

La salida en este caso es una lista b de mismo tamaño que a

**Estrategia**

La estrategia para este problema fue, crear una lista vacía b, donde si la longitud de a es = 1 entonces retorne b; de lo contrario, entonces vaya añadiendo los elementos de a hacia b, para que se invierta la lista.

**Invariante**

* Iniciación: No hay elementos en b
* Estabilidad: se añaden elementos en b
* Terminación: b tiene la misma longitud de a

**Casos de prueba**

|  |  |  |
| --- | --- | --- |
| **Entrada** | **Justificación** | **Salida** |
| A = [] | Caso vacío | [] |
| A = [2] | Caso único | [2] |
| A = [1,2,3] | Caso regular | [3,2,1] |

1. **def** reverso(a):
2. b = []
3. **if** len(a) == 0:
4. **return** b
5. **else**:
6. **return** reverso(a[1:]) + [a[0]]