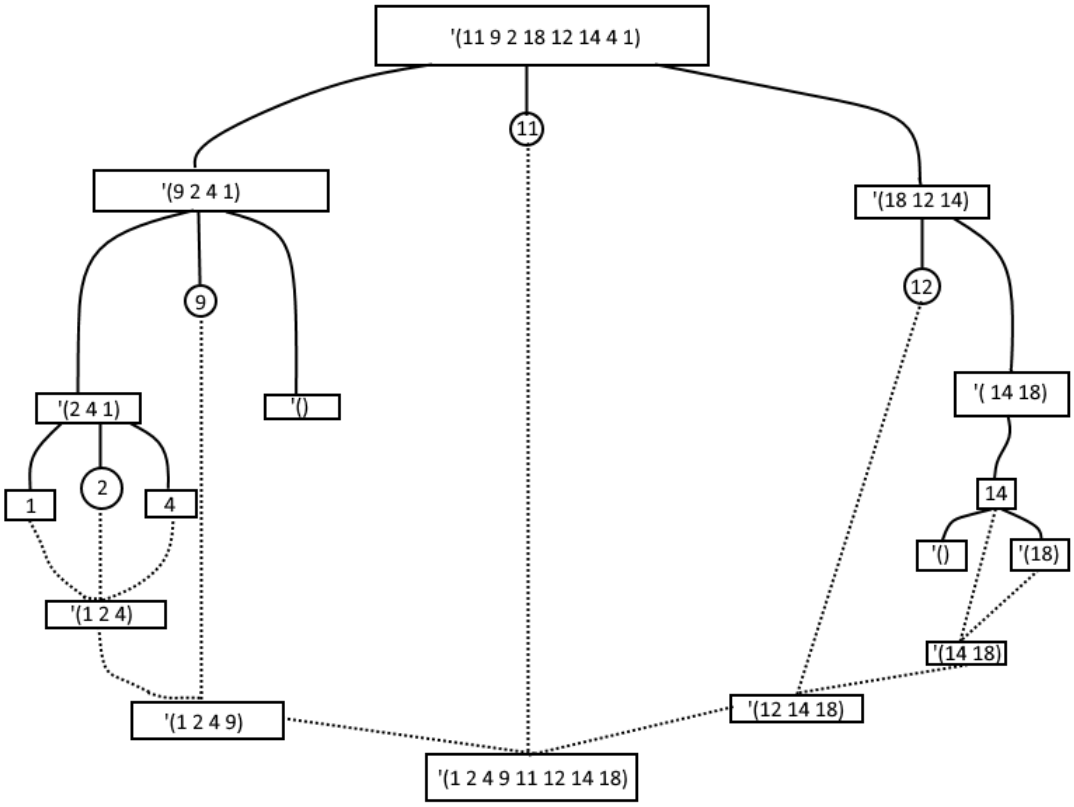
Tarea 2
Miguel Sebastian Navarro Islas

Problema 5
La llamada (bundle '("a" "b" "c") 0) es un buen uso de bundle? ¿qué produce? ¿por qué?
No es un buen uso de bundle ya que produce un ciclo infinito. Esto se debe a que se llama *take* y *drop* con $n = 0$ lo cual hace que no se consuman elementos de la lista y no se llegue a ls = '().
Problema 9
Dibuja un diagrama como el de la figura anterior pero para la lista '(11 9 2 18 12 14 4 1).



Problema 11
Si la entrada a quicksort contiene varias repeticiones de un número, va a regresar una lista estrictamente más corta que la entrada. Responde el por qué y arregla el problema.
Quicksort no toma en cuenta los duplicados ya que *smallers* y *largers* son estrictos, es decir solo toman en cuenta si son menores o mayores respectivamente. Este problema se puede solucionar formando 3 grupos de datos en vez de 2, uno para los mayores, uno para los menores y uno para los numeros iguales, de esta forma se puede mantener la lista ordenada y con la misma cantidad de datos.

Problema 13

Implementa una versión de quicksort que utilice isort si la longitud de la entrada está por debajo de un umbral. Determina este umbral utilizando la función time, escribe el procedimiento que seguiste para encontrar este umbral.

Si asumimos que isort siempre sera mas rapido que quicksort bajo una cierta cantidad de datos podemos establecer el umbral cuando isort sea mas *lento* que quicksort, ya que este siempre sera mas rapido bajo este umbral.

Para encontrar el umbral realize pruebas desde los 10 datos hasta los 200 y con estas pruebas pude colocar el umbral en 150 datos. En promedio, despues de 150 datos isort era mas lento que quicksort, mientras que en 150 datos los algoritmos mostraban 1 o 0 ms de ejecucion en quicksort por lo que isort era mas rapido.

Problema 18

Considera la siguiente definición de smallers, uno de los procedimientos utilizados en quicksort, responde en qué puede fallar al utilizar esta versión modificada en el procedimiento de ordenamiento.

Va a llegar un punto en el que smallers tenga al pivote repetidas veces, lo cual va a ocasionar que la lista de *smallers* nunca llegue a *null*, es decir, nunca va a terminar.

Problema 19

Describe con tus propias palabras cómo funciona find-largest-divisor de gcd-structural. Responde por qué comienza desde (min n m).

La función itera desde el minimo entre ambos numeros para encontrar el numero que divida a los dos sin algun residuo. Si un numero no divide a ambos se le resta uno al argumento recibido y se continua con las iteraciones hasta encontrar el GCD o hasta llegar al 1.

Comienza desde el minimo entre ambos numeros para evitar iterar en el rango entre el mayor y menor, ya que es imposible que este rango tenga al GCD.

Problema 20

Describe con tus propias palabras cómo funciona find-largest-divisor de gcd-generative.

Partiendo de la observacion de que para dos enteros positivos, el maximo comun divisor es igual al maximo comun divisor del menor y el residuo del mayor entre el menor, la funcion calcula la primera iteracion de la formula, la cual requiere el GCD de otros dos numeros, y esos otros dos numeros requieren el GCD de otros dos numeros, y asi hasta llegar a un GCD el cual permite calcular el resto de GCDs hasta llegar a la llamada inicial de la funcion.

Problema 21

Utiliza la función time para determinar cuál de las dos implementaciones es más eficiente, escribiendo tu respuesta con los tiempos de ejecución obtenidos con ambos procedimientos para valores "pequeños", "medianos" y "grandes". Justifica qué valores usaste en cada una de estas mediciones y por qué los consideraste de ese "tamaño".

Comenzando con los datos 10 y 17, no existe una diferencia, lo mismo sucede con (100, 170) , (1000, 1700) y (10000, 17000). La diferencia comienza al entrar a datos grandes mayores a 100,000 ya que el gdc estructural comienza a ser mas lento que el generativo. Los datos utilizados para probar los datos grandes fueron (10700, 17000) donde gdc-structural siempre fue mas lento que el generativo por 2 o 3 ms.

Problema 22

Piensa y describe por qué no siempre es la mejor opción elegir el procedimiento más eficiente en tiempo de ejecución. Utiliza criterios que no sean el de "eficiencia".

La primera razon para no elegir un mejor tiempo de ejecucion es lo comprensible que es una solucion. Si un algoritmo es mas complicado de entender que otro y la diferencia entre eficiencia es muy pequeña, es mejor tener un algoritmo entendible que esa diferencia de eficiencia.

Otro motivo podria ser que un algoritmo sea mas rapido pero que utilize demasiada memoria lo cual puede ocasionar muchos problemas con una cantidad grande de datos, por lo que es mejor utilizar un algoritmo que tarde mas pero evite estos problemas.

Problema 23

Implementa un procedimiento que genere otro fractal, toma en consideración la discusión de esta tarea, caracteriza el tipo de recursividad que utilizaste y justifica la terminación de tu procedimiento.

Se genero el fractal utilizando recursividad generativa y va a terminar siempre y cuando reciba un numero positivo esto se debe a que la funcion auxiliar del fractal tiene un contador donde va decreciendo en 1 a 1 hasta llegar a 0. Sin embargo, un valor negativo causaria un ciclo infinito por lo que hay un filtro para que no existan las iteraciones negativas.

El hecho de que se asegure la terminacion no significa que se puedan correr n iteraciones, ya que existe un limite dado por la memoria fisica del sistema (o la memoria que se le permita tomar a racket) por lo que numeros muy grandes no van a funcionar.