El Despegue
Tarea 02

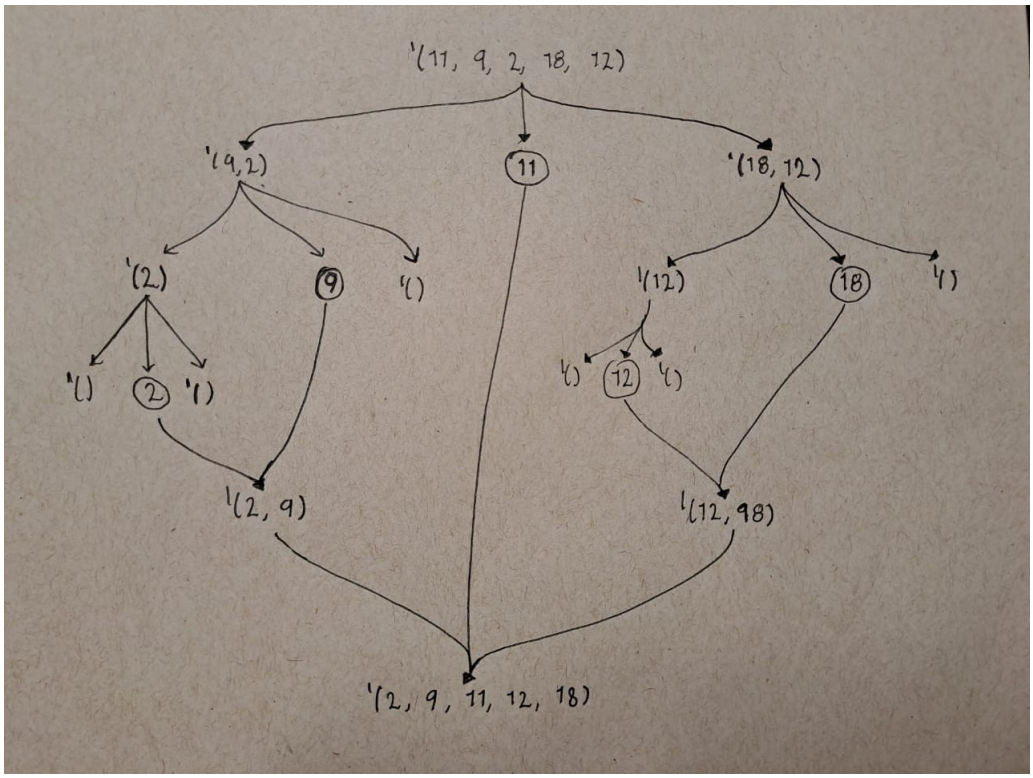
Durazo Duarte Chelsea
220216031

Lenguajes de programación 2022

Problema 5. La llamada (bundle '("a" "b" "c") 0) es un buen uso de bundle? ¿qué produce? ¿por qué?

Solicitar a la función que empaquete un bufer el pedazos de longitud "0" no tiene sentido. Cuando hablamos de longitud esperamos una cantidad positiva y hablar de una longitud "0" haría referencia a algo inexistente, así la función *bundle* se cicla. Esto sucede ya que la función drop no va reduciendo la lista y esta nunca llega al caso base cuando es vacía.
Sin embargo, al momento de programar es importante considerar todas las posibles entradas y asignar una salida por lo que consideré viable que en este caso la función devuelva una lista vacía.

Problema 9.



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.

Sucede que solo se están considerando los elementos estrictamente mayores o menores que el pivote, por lo que todos los elementos iguales al pivote quedan fuera de la lista ordenada. La solución sería considerar en uno de los casos a los elementos menores/mayores e iguales al pivote, pero solo en uno de los casos.

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

Implementé una función *testing-sort* que verifica el tiempo dado según la función *time* utilizando los dos algoritmos *quicksort* y *isort* para ordenar la misma lista.
La lista va creciendo multiplicando por 10 el anterior tamaño y en todos los casos el algoritmo *quicksort* presenta un mejor desempeño, incluso hubo casos en los que en listas de menos de 50 elementos, *quicksort* resultó ser más rápido.
Por lo anterior, el umbral del que habla el problema no existe.

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.

```
1 #lang racket
2 (define (smallers l n)
3   (cond
4     [(empty? l) '()]
5     [else (if (<= (first l) n)
6               (cons (first l) (smallers (rest l) n))
7               (smallers (rest l) n))]))
```

a

Problema 19. Describe con tus propias palabras cómo funciona find-largest-divisor de gcd-structural. Responde por qué comienza desde (min n m).

```
1 (define (gcd-structural n m)
2   (define (find-largest-divisor k)
3     (cond [(= i 1) 1]
4           [(= (remainder n i) (remainder m i) 0) i]
5           [else (find-largest-divisor (- k 1))]))
6   (find-largest-divisor (min n m)))
```

EL algoritmo recorre uno a uno los elementos menores que el mínimo de los dos números dados tratando de encontrar el común divisor mayor entre los 2. Comienza desde el mínimo porque el máximo común divisor es, como máximo, el menor de los dos números.

Problema 20. Describe con tus propias palabras cómo funciona find-largest-divisor de gcd-generative.

```
(define (gcd-generative n m)
  (define (find-largest-divisor max min)
    (if (= min 0)
        max
        (find-largest-divisor min (remainder max min))))
  (find-largest-divisor (max n m) (min n m)))
```

En este caso se toma como máximo posible común divisor el residuo que queda de dividir el mayor entre el menor si no lo es el menor, lo que resulta en una búsqueda mucho más rápida.

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".

Utilizé como valores pequeños, números menores a 1000 cuya diferencia con su máximo común divisor sea la máxima, para valores medianos utilicé la misma premisa con números menores de 1000000, y para números muy grandes, menores a 1000000000.
Después de las pruebas determiné que el algoritmo más eficiente sería el generativo. Los números fueron asignados de ese tamaño porque con números mayores ya tardaba mucho en terminar el algoritmo.

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".

En ocasiones es importante priorizar otros aspectos, tales como el uso de memoria, o en proyectos grandes es importante la legibilidad del procedimiento. Así el mejor procedimiento se designaría según la importancia que se le de a cada uno de los aspectos según el contexto.

```
> (gcd-generative 100000998 100000996)
2
> (gcd-generative 100998 100996)
2
> (gcd-generative 998 996)
2
> (testinggcd 100000998 100000996)
"Structural: "
cpu time: 13562 real time: 13963 gc time: 1468
"Generative: "
cpu time: 0 real time: 0 gc time: 0

> (testinggcd 100998 100996)
"Structural: "
cpu time: 0 real time: 9 gc time: 0
"Generative: "
cpu time: 0 real time: 0 gc time: 0

> (testinggcd 998 996)
"Structural: "
cpu time: 0 real time: 0 gc time: 0
"Generative: "
cpu time: 0 real time: 0 gc time: 0
```