Reporte de segunda tarea de Lenguajes de Programación

Diego Torres G.

September 5, 2022

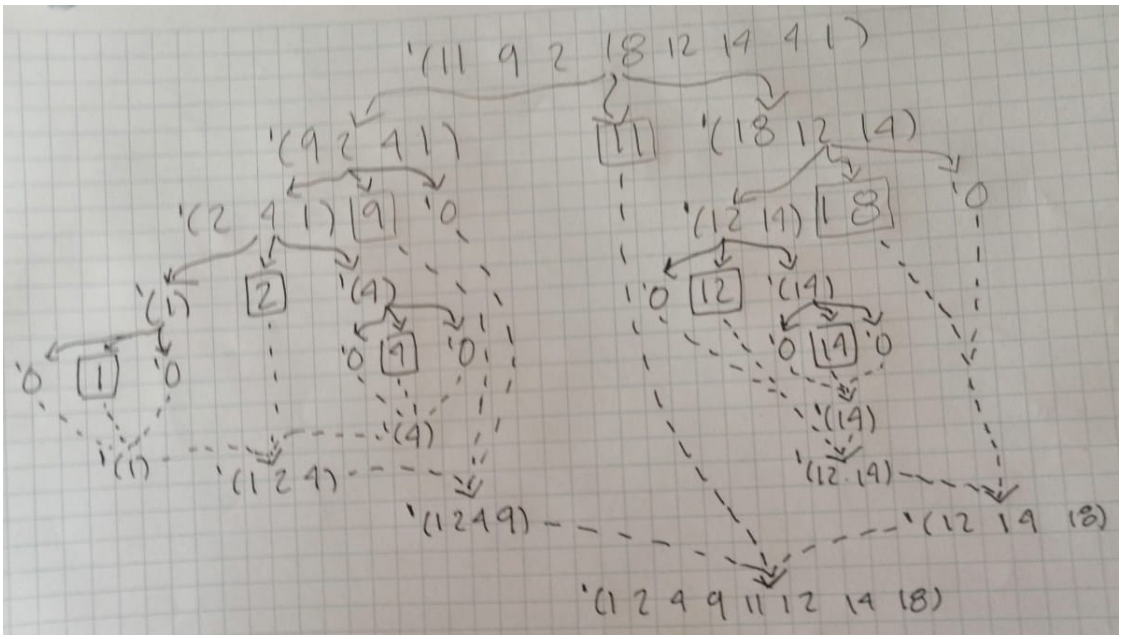
5. La llamada (bundle '("a" "b" "c") 0) es un buen uso de bundle? ¿qué produce? ¿por qué?

   De acuerdo a la definición que usa los procedimientos "take" y "drop", usar $n = 0$, haría que el procedimiento se cicle ya que en cada ocasión se haría una llamada recursiva a bundle con los mismos argumentos.

   Estarías creando una lista infinita con todos sus elementos siendo listas vacías.

   En la implementación que hay en esta tarea, bundle tira un error con $n = 0$.

9. Dibuja un diagrama como el de la figura anterior pero para la lista '(11 9 2 18 12 14 4 1).



10. Implementa los procedimientos smallers y largers.
    De acuerdo a la definición que se pedía, cualquier elemento repetido se quedaba afuera.
    Decidí cambiar el procedimiento largers por larger-or-eq para admitir elementos repetidos.
    En este caso y con esta implementación, en donde el pivote es el primer elemento, se mantiene la estabilidad del algoritmo.

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.
    Como mecioné en el problema anterior, ya había decidido modificar quicksort para aceptar repeticiones en la lista. Esto era porque para cada elección de pivote, se eliminaban las repeticiones de ese elemento al escoger solo los estrictamente mayores o menores.

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.

    Primero, mis supuestos; Tomé como si los números generados por la función estándar 'random fueran variables independientes e identicamente distribuidas, además, que el tiempo medido por 'time-apply para cada ordenación no está alterado por ningún factor externo de mi computadora o de cualquier otro tipo.
    Todo esto para decir que el tiempo promedio que tomaría a un mismo algoritmo ordenar listas del mismo tamaño generadas de la misma forma, de acuerdo al teorema de Limite Central, es una variable aleatoria normal.

    Teniendo esto en cuenta, hice un procedimiento que mide el tiempo que toma a cada algoritmo ordenar una misma lista y regresa el tamaño de entrada para el cuál quicksort es más rápido que insertion sort.

    Para obtener un tamaño de lista promedio, hice otro procedimiento que hace n experimentos y calcula el valor esperado para esa muestra de n experimentos.

    Finalmente, saqué el valor esperado con 100 experimentos y obtuve como resultado que quicksort es más rápido para un tamaño de entrada de 658 elementos. Empíricamente, observe que esta variable aleatoria no tenía mucha desviación y me conformé con este número de experiementos. Además el tiempo que le tomó a mi computadora hacer este cálculo ya era bastante grande.

17. Modifica bundle para verificar que su entrada permita la terminación y que señale un error en caso contrario.
    Bundle lanza un error en caso de que n sea negativo o cero. Esto nos asegura que en cada paso estamos resolviendo bundle con una lista más pequeña ya que estamos eliminando al menos un elemento en cada paso.

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.

    Si la lista tiene elementos y el primero de ellos es mayor que el pivote, el procedimiento nunca termina porque se llama recursivamente a la función con los mismos argumentos.

19. Describe con tus propias palabras cómo funciona find-largest-divisor de gcd-structural. Responde por qué comienza desde (min n m).
    Empieza con (min n m) porque todos los divisores de un número son menores o iguales que el número, dado que buscamos un divisor de n y m, el divisor no puede ser mayor que ninguno de los dos.

    Funciona checando si k es divisor de n y m y el siguiente caso es con el siguiente menor a k.

20. Describe con tus propias palabras cómo funciona find-largest-divisor de gcd-generative.
    Se usa la propiedad de que (gcd l s) := (gcd s (remainder l s)). En cada caso (remainder l s) es menor que s por definición. Reducimos los argumentos hasta llegar al caso base (= min 0), donde en el paso anterior el min, que ahora es el max, era divisor del max anterior.

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".

    De acuerdo a la función time, el procedimiento 'gcd-generative siempre es más rápido, sin importar el tamaño de las entradas. Probé entradas de dos digitos y la función aún no hacía una medición suficientemente precisa. Luego en los decenas de miles y 'gcd-generative fue más rápido. Luego en las decenas de cientos de millones y 'gcd-generative terminó

mientras que el estructural estaba tardando más de lo que estaba dispuesto a esperar.

También, se puede ver que 'gcd-structural tiene complejidad lineal en tiempo respecto a (min n m) y la de 'gcd-generative es logaritmica en tiempo respecto a (min n m).

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".

    - Legibilidad: Uno de los objetivos principales del código es ser leído por otras personas. Un pedazo de código es mejor si se puede entender su modo de operar más facilmente por un lector humano.

    - Propiedades del algoritmo que implementa. Por ejemplo, hay ciertos algoritmos de ordenación que tienen la propiedad de estabilidad y, dependiendo del problema, puede ser más importante que tiempo de ordenación.