1. El Problema de Selección consiste en encontrar el $k - esimo$ elemento más pequeño de un conjunto de n datos, $k \leq n$.
   Proporcionar un algoritmo que solucione el Problema de Seleccin utilizando las estrategias de:
   (a) Merge Sort; (b) Heap Sort. (c) Quick Sort; (d) Tree Sort.
   Pregunta adicional: ¿Se puede hacer sin ordenar toda la secuencia? Justificar
   (a) Merge Sort **Respuesta**
   Aquí también tenemos un problema, ya que también debemos de ordenarlo para encontrar el elemento k, esto debido a que como ordenamos los elementos y la secuencia no esta completa, además de tener que ordenar la otra sublista para después volver a mezclarlo.
   (b) Heap Sort **Respuesta**
   En este caso tenemos un pequeño problema, ya que cuando agregamos algún elemento, este puede cambiar la noción del elemento k más pequeño, es decir en otras palabras, se desorganiza todo, además de que con Heap solo se sabe que los elementos son mayores o menores y en este caso se tendrá que ordenar toda la secuencia para encontrar el elemento k más pequeño.
   (c) Quick Sort **Respuesta**
   El algoritmo a seguir es parecido al que se meciona en el ejercicio dos, pero en vez de buscar el elemento a la mitad lo buscamos en la posicion k y con esto no es necesario ordenar toda la secuencia, debido a que solo debemos de hacer la busqueda el elemento k.
   (d) Tree Sort **Respuesta**
   Como en el caso de Heap Sort, si agregamos un nuevo elemento y este nuevo elemento es menor a los elementos a los que ya teníamos, es decir en este caso como en Heap, debemos de ordenarlo y después encontrar el elemento k.

2. **Afirmación.** El sub-algoritmo **Partition** del Algoritmo **Quick Sort** puede ser usado para encontrar la mediana de una lista de valores, de tamaño n, en tiempo $O(n)$.
   a) Mostrar que la afirmación es verdadera, diseñando y presentando el algoritmo correspondiente.
   b) Justificar detalladamente que se alcanza el tiempo dado.
   Respuesta
   Sabemos que el algoritmo de Partition es comparado con el pivote para ver si el elemento va a ver a las derecha o a la izquierda, además de que tenemos que ir dividiendo la lista en dos sublistas, ahora tenemos que checar las sublistas que son de $\frac{n}{2}$ y se va reduciendo, es decir : n, $\frac{n}{2}$, $\frac{n}{4}$, .. asi hasta acabar y nos da un resultado de $2n$, es decir $O(n)$ El algoritmo recibe una lista L y escogemos como pivote que es $\frac{n}{2}$ ya que dividimos la lista en dos, luego como Partition nos regresa el índice j que van del $[0, j]$ que es la primera parte de la sublista y luego tenemos la otra sublista que va de $[j, n]$ los cuales son elementos mayores o iguales. Ahora solo debemos de hacer recursión sobre los $\frac{n}{2}$ elementos, mientras que en la otra sublista que va de $[j, n]$ y tenemos más de $\frac{n}{2}$ buscando el elemento n - $\frac{n}{2}$ elemento y por ende ya la siguiente iteración ya no sera sobre $\frac{n}{2}$ si no que ahora sera $\frac{n}{4}$ debido a que la búsqueda se reduce y así hasta llegar a solo tener un elemento.

3. Sea L una secuencia de n números enteros diferentes. Suponga que los elementos x de L están en el intervalo [1, 2000].
   a) Diseñar un algoritmo de orden lineal que ordene los elementos de la secuencia L.
   b) Justificar detalladamente que se alcanza el tiempo solicitado.
   c) El algoritmo resultante, ¿viola la cota mínima de ordenamiento?
   ¿Por qué? Justifica con detalle tu respuesta.
   Respuesta
   Veamos que la condición que nos muestra el problema que es un arreglo de [1,200], entonces lo que debemos de hacer es asignar los números en colas basadas en el menor dígito significativo por ejemplo:
   Supongamos que tenemos esta secuencia de números $[35, 67, 58, 87, 22, 106, 226, 66]$
   y tomaremos como dígito menos importante a los centésimos ya que es el mayor de los elementos posibles y nos quedaría:
   $[35, 67, 58, 87, 22, 66, 106, 226]$
   luego nos fijamos en los decimos y nos queda:
   $[22, 35, 66, 67, 87, 58, 106, 226]$
   y por ultimo nos fijamos en la primer elemento de los números y obtendremos como resultado :
   $[22, 35, 58, 66, 67, 87, 106, 226]$
   Este algoritmos es llamado ordenamiento de Radix o Radix Sort y este tiene una complejidad de $O(n)$ y por ende no este no viola la cota mínima del ordenamiento debido a que este algoritmo no se basa en

comparaciones, si no en el ordenamiento en base a los elementos del arreglo, además de que la cota mínima de ordenamiento es encontrada en algoritmos basados en comparaciones y por lo mencionado anteriormente Radix Sort no lo es.

4. Considerar dos versiones de **Quick Sort**, aplicado a una secuencia A de datos n datos; A[l..r] **QuickSort 1** que toma como pivote al elemento A[n div 2]; **QuickSort 2** que toma como pivote al elemento que resulta ser la mediana de A[l], A[(l + r) div 2], A[r].
(a) Dar un ejemplo de una lista de al menos 35 valores donde el desempeño computacional de **QuickSort 2** sea mejor que el de **QuickSort 1**
(b) Dar un ejemplo de una lista de al menos 35 valores donde el desempeño computacional de **QuickSort 1** sea mejor que el de **QuickSort 2**
(c) Presentar la traza de ambas ejecuciones.
Respuesta
Tenemos la lista
$[36, 34, 32, 30, 28, 26, 24, 22, 20, 18, 16, 14, 12, 10, 8, 6, 4, 2, 40, 42, 44, 46, 48, 50, 52, 54, 56, 58, 60, 62, 64, 66, 68, 70, 72]$
entonces escogemos la posición $[17]$ del arreglo que vendría siendo el valor de 2 que es el elemento menor de toda la lista y por ende caer en el peor de los casos teniendo como resultado:
$[2, 36, 34, 32, 30, 28, 26, 24, 22, 20, 18, 16, 14, 12, 10, 8, 6, 4, 40, 42, 44, 46, 48, 50, 52, 54, 56, 58, 60, 62, 64, 66, 68, 70, 72]$
Que tiene como complejidad $O(n^2)$
Ahora consideremos la misma lista, es decir $[36, .., 2, ..72]$ ahora escogemos al número 36 que seria el pivote y abría 17 elementos del lado derecho de este y 17 elementos del lado izquierdo, como se muestra a continuación:
$[34, 32, 30, 28, 26, 24, 22, 20, 18, 16, 14, 12, 10, 8, 6, 4, 2, 36, 40, 42, 44, 46, 48, 50, 52, 54, 56, 58, 60, 62, 64, 66, 68, 70, 72]$
siendo este el mejor caso y dándonos una complejidad de $O(nlogn)$.

5. **Opcional** Consideremos k secuencias de elementos $S_1, S_2, ..., S_k$. Cada secuencia S_j está ordenada. Además, $|S_1| + |S_2| + ... + |S_k| = n$, el número total de elementos.
(a)**Diseñar** un algoritmo que genere una secuencia ordenada de estos n elementos. Su algoritmo debe tener complejidad $O(nlogk)$.
(b)**Verificar** que el algoritmo propuesto alcanza la cota deseada.
Respuesta