Gabriel Lopez Romero – Leg. 13996

1-

Usando el método maestro:

$T(n) = T(n/2) + 6n^3$

Donde $a = 1$, $b = 2$ y $f(n) = 6n^3$.

La complejidad de $f(n)$ es $f(n) = \Theta(n^3)$, ya que el término de mayor orden en $f(n)$ es n^3.

Para aplicar el método maestro, debemos comparar el valor de $\log_b(a) = \log_2(1) = 0$ con la complejidad de $f(n) = \Theta(n^3)$.

Si $\log_b(a) < c$, donde c es la complejidad de $f(n)$, entonces la solución de la recurrencia es $T(n) = \Theta(n^c)$.

En este caso, $\log_2(1) < 3$, por lo que se cumple que $\log_b(a) < c$, y la solución de la recurrencia es $T(n) = \Theta(n^3)$.

Por lo tanto, la complejidad temporal del algoritmo es $\Theta(n^3)$, lo cual significa que no es $O(n^2)$.

2-

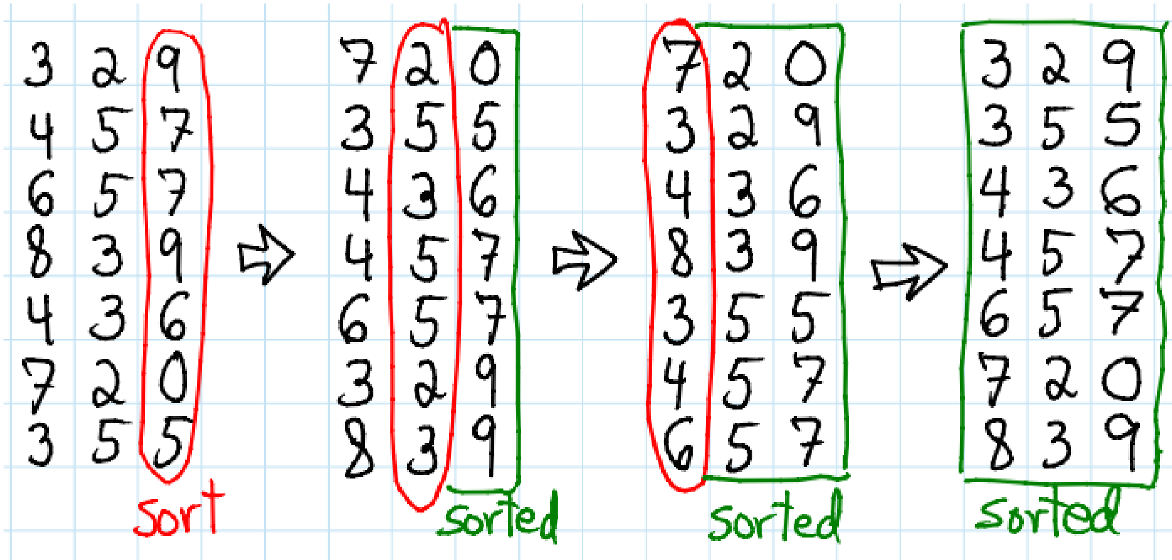
Un array ideal sería [1,2,3,4,5,6,7,8,9,10], ya que si se selecciona el primer elemento como pivote, cada llamada recursiva dividiría el array en dos partes iguales con 5 elementos en cada subarray, lo que resultaría en una complejidad de tiempo $O(n \log n)$.

3-

Para quicksort, este es uno de sus peores casos, porque al ser todos los elementos iguales, eligiría siempre el primer elemento como pivot y al comparar con el resto de elementos no encontraría elementos mayores o menores, por lo que todos deberían estar en la misma partición y crearía arrays vacíos en cada llamada recursiva, por lo que terminaría en $O(n^2)$. Para insertion sort, será $O(n)$, ya que tiene que recorrer todo el array si o si. Para merge sort, el tiempo siempre será $O(n \log n)$

6-

El funcionamiento de RadixSort es sencillo y se ve gráficamente muy fácil con esta imagen:



Al ser un algoritmo de ordenación lineal, su complejidad será lineal sin importar el input. Más específicamente, O(n*k) donde n es la longitud del array y k es la cantidad de dígitos del número mas grande.