Ejercicio 1:

Para demostrar que 6n^3 no es O(n^2), necesitamos mostrar que no hay una constante positiva c y un entero positivo n_0 tales que $6n^3 \leq c \cdot n^2$ para todo $n \geq n_0$.

(1) Vamos a suponer lo contrario, que 6n^3 es O(n^2). Esto significa que existe una constante positiva c tal que $6n^3 \leq c \cdot n^2$ para todo n mayor a n_0

Podemos dividir ambos lados de la desigualdad por n^2, resultando:

$6n^3 \leq c \cdot n^2 \rightarrow 6n \leq c$

Pero, sin importar que valor tome c o que tan grande sea, siempre vamos a poder encontrar un n que multiplicado por 6 sea mayor a c, por lo tanto, lo que se supuso en (1) es incorrecto y $6n^3 \neq O(n^2)$.

Ejercicio 2:

El mejor caso para la estrategia de QuickSort seria que el pivote que se elige sea siempre el elemento que, al momento de ordenarlo, iría en el medio, por ejemplo:

10	9	8	7	6	5	4	3	2	1

En este caso se elige de pivote al elemento del centro

Ejercicio 3:

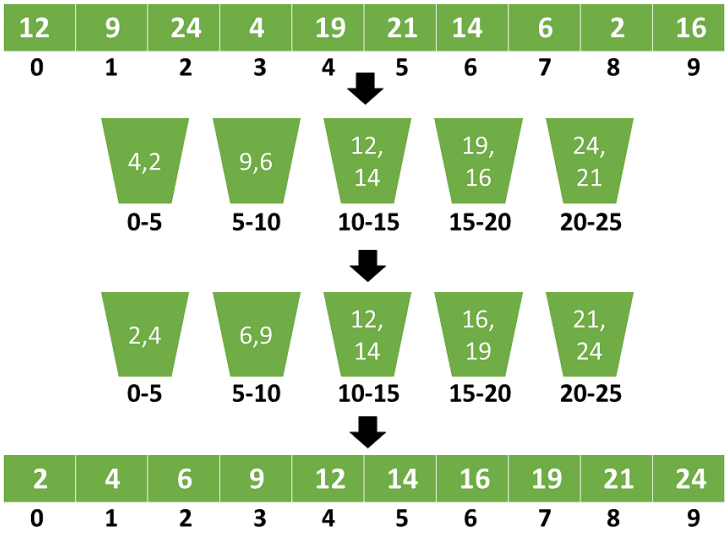
-EL tiempo de ejecución de QuickSort sería de O(n^2) dado que al estar el array ya ordenado va a tener que reordenar los valores que siempre van a quedar o de un lado o del otro del elemento que quedara fijo.

-Para la estrategia de InsertionSort el tiempo de ejecución es de O(n) debido a que hay que tomar todos los elementos del array para ver si es que es necesario moverlo de lugar, pero como todos sus elementos son iguales esta parte se omite.

-Para MergeSort el tiempo va a ser igual al peor caso O(nlogn) dado que sin importar que los elementos ya se encuentren ordenados la operación de dividir la tiene que realizar de todas formas al igual que la de combinar las distintas soluciones.

Ejercicio 6:

Bucket Sort: En este algoritmo se agrupan los distintos elementos que tenemos del array en distintas buckets o cubetas, donde luego los elementos que queden seleccionados se ordenaran dentro de cada grupo para después finalizar el algoritmo uniendo todos los resultados en un array que queda ordenado. Por ej:



-El orden de complejidad de este algoritmo es de O(n^2) ya que al depender de como queden agrupados los elementos y que método se utilice para ordenarlos una vez en la cubeta, podría pasar que todos los elementos se agrupen juntos y el algoritmo que se elija para el ordenamiento sea de orden O(n^2) (Peor caso)

- En cuanto al mejor caso, este sería el cual al dividir los elementos en cubetas, estos queden uniformemente distribuidos por lo que su orden de complejidad seria de O(n).

-En el caso promedio los elementos están distribuidos de forma equilibrada y se utiliza un algoritmo de ordenamiento eficiente, por lo que dependería del tamaño de cubetas (k) y del array (n), quedando el orden de complejidad de O(n+k)

<u>Ejercicio 7:</u>

a) $T(n) = 2T(n/2) + n^4$

$a = 2, b = 2, c = 4, f(n) = n^4, \log_2(2) = 1$

Por el caso 3 tenemos que $f(n) = \Omega(n^{1+\varepsilon})$, para $\varepsilon = 3$

Y además, $2 * n^4/16 = n^4/8 \leq n^4/8$, para $C = 1/8 < 1$

Por lo tanto, $T(n) = \theta(n^4)$

b) $T(n) = 2T(7n/10) + n$

$a = 2, b = 10/7, c = 1, f(n) = n, \log_{10/7}(2) \approx 1{,}94$

Por el primer caso tenemos que $f(n) = O(n^{1{,}94-\varepsilon})$, para $\varepsilon \approx 1{,}94$. Luego, tenemos que $T(n) = O(n^{1{,}94})$

c) $T(n) = 16T(n/4) + n^2$

$a = 16, b = 4, c = 2, f(n) = n^2, \log_4(16) = 2$

Como $f(n) = n^2 = \theta(n^2)$, por el caso numero 2, tenemos que $T(n) = (n^2 \log n)$

d) $T(n) = 7T(n/3) + n^2$

$a = 7, b = 3, c = 2$

Como $\log_3(7) \approx 1,77 < 2$, tenemos que $T(n) = \theta(n^2)$

e) $T(n) = 7T(n/2) + n^2$

$a = 7, b = 2, c = 2$

Como $\log_2(7) \approx 2,8 > 2$, tenemos que T(n) = θ(n^($\log_2(7) \approx 2,8$))

f) $T(n) = 2T(n/4) + \sqrt{n}$

$a = 2, b = 4, c = 1/2$

Como $\log_4(2) = 1/2 = 1/2$, tenemos que $T(n) = \theta(n^{1/2} \log n)$