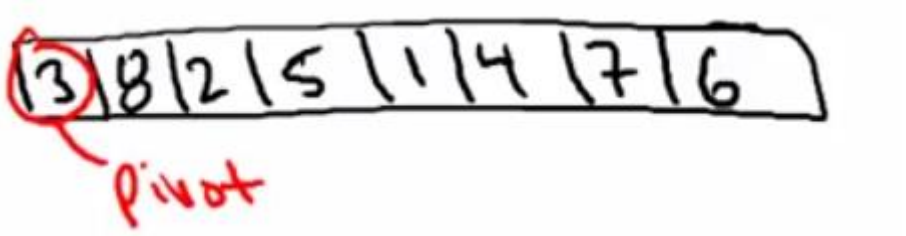
El problema de Ordenar

QuickSort

Algoritmo recursivo que ordena un arreglo desordenado (asume que no hay elementos repetidos) y devuelve uno ordenado en el orden O(n log n) en promedio

***Idea**: Particionar el vector alrededor de un PIVOTE*

1. ***Seleccionar un pivote (podría ser el primer elemento)***



2. ***PARTICIONAR el vector de manera que :***
   a. ***A la izquierda del pivote estén los menores al pivote***
   b. ***A la derecha del pivote estén los mayores al pivote***

3. ***Note que el pivote queda en la posición que le corresponde***
4. ***Llamar recursivamente al algoritmo con las dos partes que quedan***

A continuación el algoritmo a alto nivel del QuickSort

```
QuickSort (Arreglo A, posición inicio,  longitud N) -> A

    Si (inicio < N) entonces  //Longitud de la lista mayor a 1
        Pivote=inicio  //Primer OJO: Podría ser otro valor
        Particionar(A, inicio, N) -> pivote //Pivote cambia de posición
        QuickSort( A, inicio, pivote-1)      // llamada recursiva
        QuickSort( A, pivote+1,  N)          // llamada recursiva
    Fin Si

Terminar devolviendo A
```

Recurrencia que define el tiempo del algoritmo QuickSort

$T_{qs}(N) = N + 2T_{qs}(N/2)$
$T_{qs}(1) = 2$

Análisis de la recurrencia

$T_{qs}(N) = N + 2T_{qs}(N/2)$
$T_{qs}(N) = N + 2(N/2 + 2\,T_{qs}(N/4))$
$T_{qs}(N) = N + 2(N/2 + 2\,(N/4 + 2\,T_{qs}(N/8))$
$T_{qs}(N) = N + N + N + 8\,T_{qs}(N/8))$
$T_{qs}(N) = N + N + N + 2^3\,T_{qs}(N/2^3))$
....
$T_{qs}(N) = N + N + N + ... + 2^x\,T_{qs}(N/2^x))$

¿Cuantas veces hay que sumar N?

$T_{qs}(N) = N * x + 2^x\,T_{qs}(N/2^x))$

¿Cuál es el valor de x?
Para que $T_{qs}(N/2^x) = T(1)$ tendría que $2^x = N \Rightarrow x = lg_2 N$

$T_{qs}(N) = N\ lg_2 N + 2^{\,lg2N}\,T_{qs}(N/2^{\,lg2N}))$
$T_{qs}(N) = N\ lg_2 N + N\,T_{qs}(N/N)$
$T_{qs}(N) = N\ lg_2 N + N\,T_{qs}(1)$
$T_{qs}(N) = N\ lg_2 N + N\ 2$

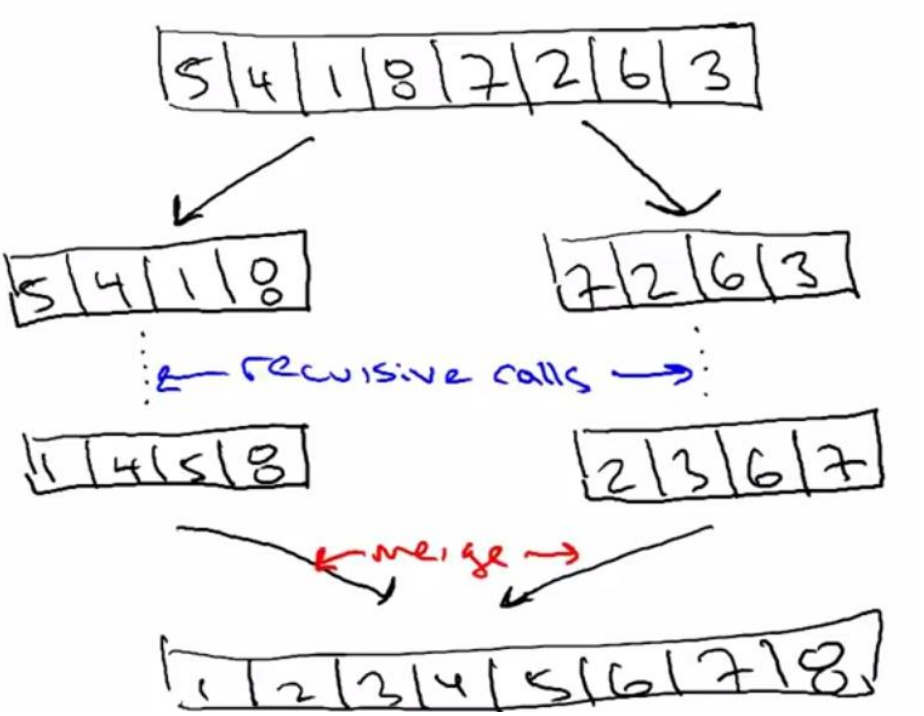
Creemos que $T_{qs}(N)$ pertenece al orden $O(N\ lg_2 N)$
Hay que demostrar usando inducción

MergeSort

Algoritmo recursivo que ordena un arreglo desordenado (asume que no hay elementos repetidos) y devuelve uno ordenado en el orden O (n log n)

***Idea**: Utilizar la técnica de Divide y Veceras*

1. ***Dividir el vector en dos partes***
2. ***Llamar recursivamente con las dos mitades***



3. ***Fusionar las dos listas ordenadas de manera que la lista final quede ordenada***

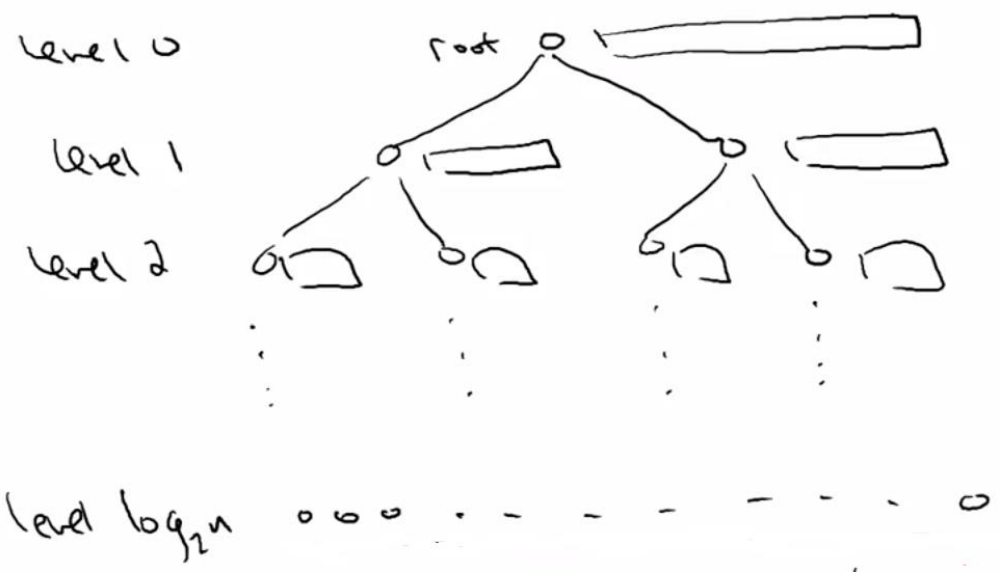
A continuación el algoritmo a alto nivel del MergeSort

```
Sort (Arreglo A, posición inicio,  longitud N) -> A

    Si (inicio < N) entonces  //Longitud de la lista mayor a 1
        mid=(inicio+N)/2;
        Sort (A, inicio, mid)      // llamada recursiva
        Sort (A, mid+1,  N)        // llamada recursiva
        Merge(A, inicio, mid, N) -> A  // Fusiona dos listas ordenadas
    Fin Si
Terminar devolviendo A
```

Análisis de eficiencia en tiempo Merge Sort

Si se tiene el siguiente árbol de recursión que representa el número de llamadas recursivas del algoritmo



Podemos ver que el número de niveles del árbol es de $\log_2 N$ donde N es la longitud de Vector, para ser más precisos **$(\log_2 N+1)$**

Para cada nivel j=0, 1, 2 … $\log_2 N$ podemos observar que hay **2^j** sub problemas cada uno de longitud **$N/2^j$**

Es interesante observar que el número total de operaciones en cada nivel j es <= $2^j * N/2^j$ que resulta ser = N

Si se tiene un total ($\log_2 N+1$) niveles, tenemos que el total de operaciones es <= N* ($\log_2 N+1$)
Por lo tanto sabemos que MergeSort para ordenar un vector usa al menos **$N \log_2 N+N$** operaciones

Por lo tanto queda por demostrar que el $T_{ms}(N)$ pertenece al orden $O(N \lg_2 N)$