

mergeSort(0,v.length,v)

p = 0 , n = 6, q = 3

mergeSort(p,q,v)

p = 0 , n = 3, q = 1

mergeSort(p,q,v)

p = 0 , n = 1, q = 1

mergeSort(p,q,v)

p = 0 , n = 1  
return

p = 0 , n = 1, q = 1

mergeSort(q,n,v)

p = 1 , n = 1  
return

v[] = {5, 8, 2, 1, 7, 4};

```
public static void mergeSort(int p, int n, int []
v){
    if(p < n - 1) {
        int q = (p+ n) / 2;
        mergeSort(p,q,v);
        mergeSort(q,n,v);
        intercala(p,q,n,v);
    }
}
```

mergeSort(q,n,v)

p = 3 , n = 6 , q = 4

mergeSort(p,q,v)

p = 3 , n = 4

mergeSort(p,q,v)

p = 3 , n = 4  
return

mergeSort(q,n,v)

p = 4 , n = 6 , q = 5

p = 5 , n = 6  
return

quickSort) v, 0, v.length - 1

quickSort(v , p, pivo - 1)  
p = 0 , r = 5

quickSort(v , p, pivo - 1)  
p = 0 , r = 2

return p = 0 , r = 0

quickSort(v , pivo + 1, r)  
p = 4 , r = 5

quickSort(v , p, pivo - 1)  
p = 4 , r = 4

return p = 4 , r = 4

quickSort(v , pivo + 1, r)  
p = 4 , r = 5

return p = 5 , r = 5

v[] = {5, 8, 2, 1, 7, 4};

```
static void quickSort(int v[], int p, int r) {
    if (p < r) {
        int pivo = particao(v, p, r);
        quickSort(v, p, pivo - 1);
        quickSort(v, pivo + 1, r);
    }
}
```

