3.3 Sorting

- Quick sort
  - Given a sequence of n elements { a_1, a_2, …, a_n }, split them into two set.
  - Rearrange the elements so that merging later is not necessary.
  - Rearrange such that $a_i <= a_j$ for all i between 1 and m and j between m+1 and n ➔ partitioning
  - Two steps
    - Divide
      - Split the list into two halves (with partitioning)
    - Conquer
      - Recursively sort each half

- partition

```
int partition ( int s, int e, int A[] )
{   int pivot, left, right;
    left = s+1;             right = e;          pivot = A[s];
    while (left <= right) {
        while ((A[right] >= pivot) && (left <= right))
            right--;

        while ((A[left] <= pivot) && (left <= right))
            left++;

        if ( left <= right )
            swap ( A[left], A[right] );
    }
    swap ( A[right], A[s] ); // A[s]: pivot;
    return right;
}
```

- Quick sort (Divide & Conquer)

```
void quick_sort( int s, int e, int A[] )
{
    if ( s >= e )
        return;
    int m = partition ( s, e, A );
    quick_sort ( s, m-1, A );
    quick_sort ( m+1, e, A );
}
```

- Quick sort (Example)

```
void quick_sort( int s, int e, int A[] )
{
    if ( s >= e )
        return;
    int m = partition ( s, e, A );
    quick_sort ( s, m-1, A );
    quick_sort ( m+1, e, A );
}
```

```
int partition ( int s, int e, int A[] )
{   int pivot, left, right;
    left = s+1;            right = e;          pivot = A[s];
    while (left <= right) {
        while ((A[right] >= pivot) && (left <= right))
            right--;

        while ((A[left] <= pivot) && (left <= right))
            left++;

        if ( left <= right )
            swap ( A[left], A[right] );
    }
    swap ( A[right], A[s] ); // A[s]: pivot;
    return right;
}
```

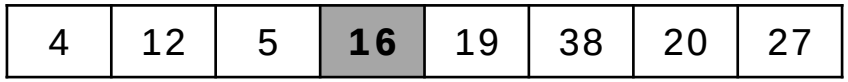
qs(0, 7)

```
void quick_sort( int s, int e, int A[] )
{
    if ( s >= e )
        return;
    int m = partition ( s, e, A );
    quick_sort ( s, m-1, A );
    quick_sort ( m+1, e, A );
}
```

```
int partition ( int s, int e, int A[] )
{   int pivot, left, right;
    left = s+1;            right = e;          pivot = A[s];
    while (left <= right) {
        while ((A[right] >= pivot) && (left <= right))
            right--;

        while ((A[left] <= pivot) && (left <= right))
            left++;

        if ( left <= right )
            swap ( A[left], A[right] );
    }
    swap ( A[right], A[s] ); // A[s]: pivot;
    return right;
}
```

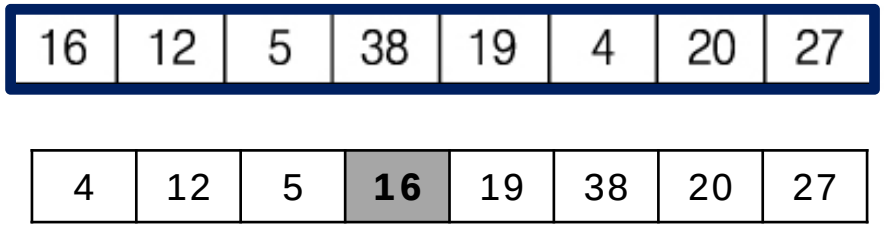
pivot

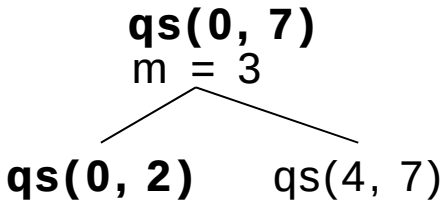
qs(0, 7)
m = 3



```
void quick_sort( int s, int e, int A[] )
{
    if ( s >= e )
        return;
    int m = partition ( s, e, A );
    quick_sort ( s, m-1, A );
    quick_sort ( m+1, e, A );
}
```

```
int partition ( int s, int e, int A[] )
{   int pivot, left, right;
    left = s+1;            right = e;          pivot = A[s];
    while (left <= right) {
        while ((A[right] >= pivot) && (left <= right))
            right--;

        while ((A[left] <= pivot) && (left <= right))
            left++;

        if ( left <= right )
            swap ( A[left], A[right] );
    }
    swap ( A[right], A[s] ); // A[s]: pivot;
    return right;
}
```





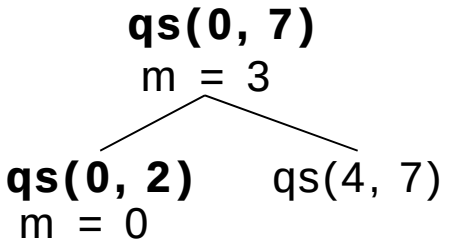
```
void quick_sort( int s, int e, int A[] )
{
    if ( s >= e )
        return;
    int m = partition ( s, e, A );
    quick_sort ( s, m-1, A );
    quick_sort ( m+1, e, A );
}
```

```
int partition ( int s, int e, int A[] )
{   int pivot, left, right;
    left = s+1;            right = e;          pivot = A[s];
    while (left <= right) {
        while ((A[right] >= pivot) && (left <= right))
            right--;

        while ((A[left] <= pivot) && (left <= right))
            left++;

        if ( left <= right )
            swap ( A[left], A[right] );
    }
    swap ( A[right], A[s] ); // A[s]: pivot;
    return right;
}
```

16	12	5	38	19	4	20	27

4	12	5	**16**	19	38	20	27

4	12	5	**16**	19	38	20	27



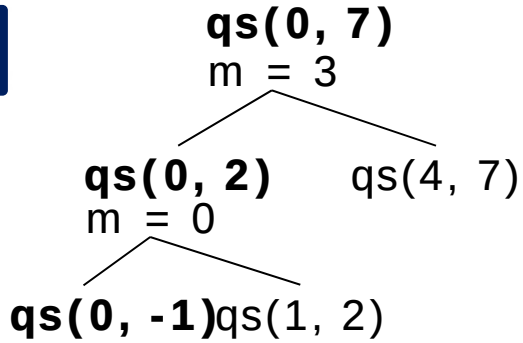
```
void quick_sort( int s, int e, int A[] )
{
    if ( s >= e )
        return;
    int m = partition ( s, e, A );
    quick_sort ( s, m-1, A );
    quick_sort ( m+1, e, A );
}
```

```
int partition ( int s, int e, int A[] )
{   int pivot, left, right;
    left = s+1;            right = e;          pivot = A[s];
    while (left <= right) {
        while ((A[right] >= pivot) && (left <= right))
            right--;

        while ((A[left] <= pivot) && (left <= right))
            left++;

        if ( left <= right )
            swap ( A[left], A[right] );
    }
    swap ( A[right], A[s] ); // A[s]: pivot;
    return right;
}
```

16	12	5	38	19	4	20	27

4	12	5	**16**	19	38	20	27

4	12	5	**16**	19	38	20	27



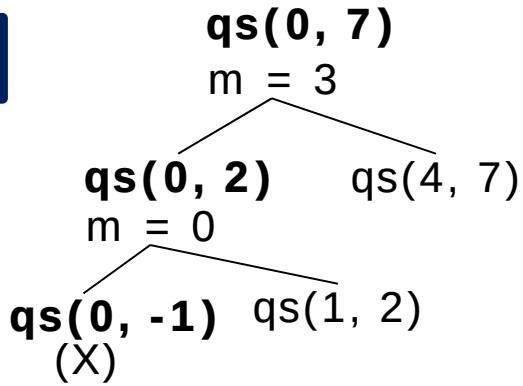
```
void quick_sort( int s, int e, int A[] )
{
    if ( s >= e )
        return;
    int m = partition ( s, e, A );
    quick_sort ( s, m-1, A );
    quick_sort ( m+1, e, A );
}
```

```
int partition ( int s, int e, int A[] )
{   int pivot, left, right;
    left = s+1;            right = e;          pivot = A[s];
    while (left <= right) {
        while ((A[right] >= pivot) && (left <= right))
            right--;

        while ((A[left] <= pivot) && (left <= right))
            left++;

        if ( left <= right )
            swap ( A[left], A[right] );
    }
    swap ( A[right], A[s] ); // A[s]: pivot;
    return right;
}
```

16	12	5	38	19	4	20	27

4	12	5	**16**	19	38	20	27

4	12	5	**16**	19	38	20	27



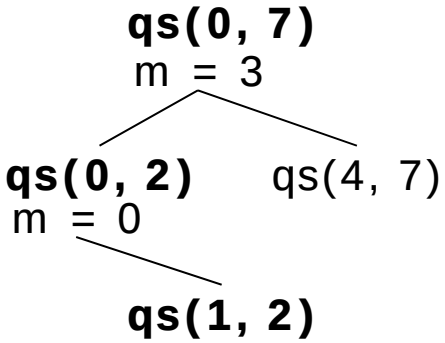
```
void quick_sort( int s, int e, int A[] )
{
    if ( s >= e )
        return;
    int m = partition ( s, e, A );
    quick_sort ( s, m-1, A );
    quick_sort ( m+1, e, A );
}
```

```
int partition ( int s, int e, int A[] )
{   int pivot, left, right;
    left = s+1;            right = e;          pivot = A[s];
    while (left <= right) {
        while ((A[right] >= pivot) && (left <= right))
            right--;

        while ((A[left] <= pivot) && (left <= right))
            left++;

        if ( left <= right )
            swap ( A[left], A[right] );
    }
    swap ( A[right], A[s] ); // A[s]: pivot;
    return right;
}
```

16	12	5	38	19	4	20	27

4	12	5	**16**	19	38	20	27

4	12	5	**16**	19	38	20	27



```
void quick_sort( int s, int e, int A[] )
{
    if ( s >= e )
        return;
    int m = partition ( s, e, A );
    quick_sort ( s, m-1, A );
    quick_sort ( m+1, e, A );
}
```

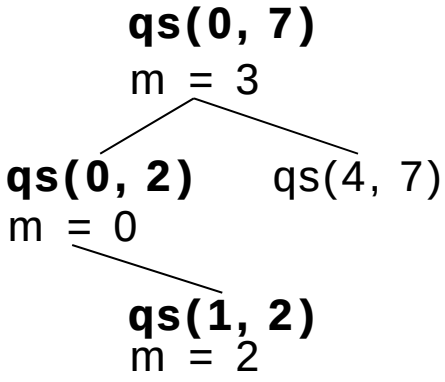
```
int partition ( int s, int e, int A[] )
{   int pivot, left, right;
    left = s+1;            right = e;          pivot = A[s];
    while (left <= right) {
        while ((A[right] >= pivot) && (left <= right))
            right--;

        while ((A[left] <= pivot) && (left <= right))
            left++;

        if ( left <= right )
            swap ( A[left], A[right] );
    }
    swap ( A[right], A[s] ); // A[s]: pivot;
    return right;
}
```

16	12	5	38	19	4	20	27

4	12	5	**16**	19	38	20	27

4	12	5	**16**	19	38	20	27

4	5	**12**	**16**	19	38	20	27



```
void quick_sort( int s, int e, int A[] )
{
    if ( s >= e )
        return;
    int m = partition ( s, e, A );
    quick_sort ( s, m-1, A );
    quick_sort ( m+1, e, A );
}
```

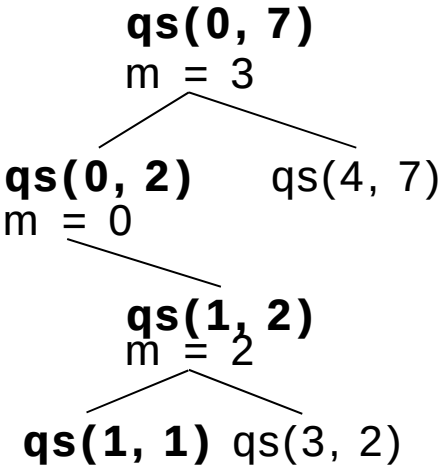
```
int partition ( int s, int e, int A[] )
{   int pivot, left, right;
    left = s+1;            right = e;          pivot = A[s];
    while (left <= right) {
        while ((A[right] >= pivot) && (left <= right))
            right--;

        while ((A[left] <= pivot) && (left <= right))
            left++;

        if ( left <= right )
            swap ( A[left], A[right] );
    }
    swap ( A[right], A[s] ); // A[s]: pivot;
    return right;
}
```

16	12	5	38	19	4	20	27

4	12	5	**16**	19	38	20	27

4	12	5	**16**	19	38	20	27

4	5	**12**	**16**	19	38	20	27



```
void quick_sort( int s, int e, int A[] )
{
    if ( s >= e )
        return;
    int m = partition ( s, e, A );
    quick_sort ( s, m-1, A );
    quick_sort ( m+1, e, A );
}
```

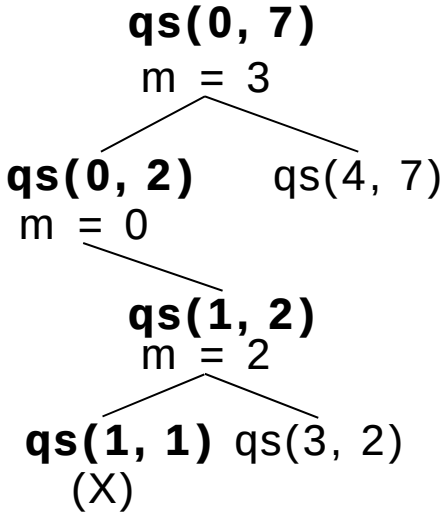
```
int partition ( int s, int e, int A[] )
{   int pivot, left, right;
    left = s+1;            right = e;          pivot = A[s];
    while (left <= right) {
        while ((A[right] >= pivot) && (left <= right))
            right--;

        while ((A[left] <= pivot) && (left <= right))
            left++;

        if ( left <= right )
            swap ( A[left], A[right] );
    }
    swap ( A[right], A[s] ); // A[s]: pivot;
    return right;
}
```

16	12	5	38	19	4	20	27

4	12	5	**16**	19	38	20	27

4	12	5	**16**	19	38	20	27

4	5	**12**	**16**	19	38	20	27



```
void quick_sort( int s, int e, int A[] )
{
    if ( s >= e )
        return;
    int m = partition ( s, e, A );
    quick_sort ( s, m-1, A );
    quick_sort ( m+1, e, A );
}
```

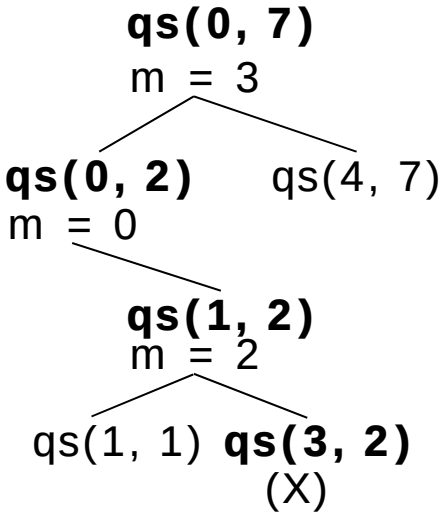
```
int partition ( int s, int e, int A[] )
{   int pivot, left, right;
    left = s+1;            right = e;          pivot = A[s];
    while (left <= right) {
        while ((A[right] >= pivot) && (left <= right))
            right--;

        while ((A[left] <= pivot) && (left <= right))
            left++;

        if ( left <= right )
            swap ( A[left], A[right] );
    }
    swap ( A[right], A[s] ); // A[s]: pivot;
    return right;
}
```

16	12	5	38	19	4	20	27

4	12	5	**16**	19	38	20	27

4	12	5	**16**	19	38	20	27

4	5	**12**	**16**	19	38	20	27

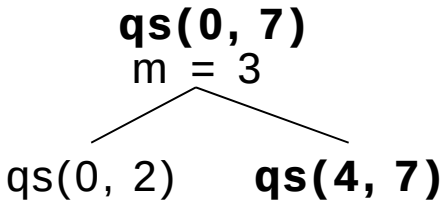


```
void quick_sort( int s, int e, int A[] )
{
    if ( s >= e )
        return;
    int m = partition ( s, e, A );
    quick_sort ( s, m-1, A );
    quick_sort ( m+1, e, A );
}
```

```
int partition ( int s, int e, int A[] )
{   int pivot, left, right;
    left = s+1;            right = e;          pivot = A[s];
    while (left <= right) {
        while ((A[right] >= pivot) && (left <= right))
            right--;

        while ((A[left] <= pivot) && (left <= right))
            left++;

        if ( left <= right )
            swap ( A[left], A[right] );
    }
    swap ( A[right], A[s] ); // A[s]: pivot;
    return right;
}
```

16	12	5	38	19	4	20	27

4	12	5	**16**	19	38	20	27
4	12	5	**16**	19	38	20	27
4	5	**12**	**16**	19	38	20	27
4	**5**	**12**	**16**	19	38	20	27

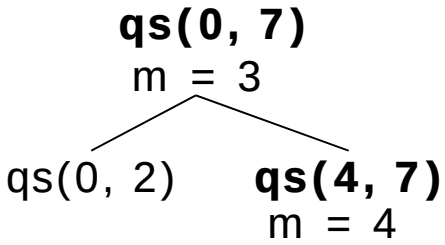


```
void quick_sort( int s, int e, int A[] )
{
    if ( s >= e )
        return;
    int m = partition ( s, e, A );
    quick_sort ( s, m-1, A );
    quick_sort ( m+1, e, A );
}
```

```
int partition ( int s, int e, int A[] )
{   int pivot, left, right;
    left = s+1;            right = e;          pivot = A[s];
    while (left <= right) {
        while ((A[right] >= pivot) && (left <= right))
            right--;

        while ((A[left] <= pivot) && (left <= right))
            left++;

        if ( left <= right )
            swap ( A[left], A[right] );
    }
    swap ( A[right], A[s] ); // A[s]: pivot;
    return right;
}
```

16	12	5	38	19	4	20	27

4	12	5	**16**	19	38	20	27
4	12	5	**16**	19	38	20	27
4	5	**12**	**16**	19	38	20	27
4	**5**	**12**	**16**	19	38	20	27
4	**5**	**12**	**16**	**19**	38	20	27

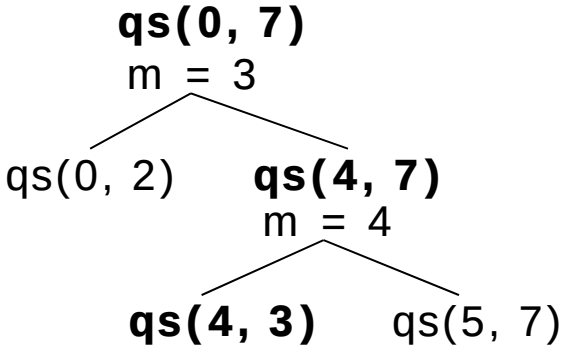


```
void quick_sort( int s, int e, int A[] )
{
    if ( s >= e )
        return;
    int m = partition ( s, e, A );
    quick_sort ( s, m-1, A );
    quick_sort ( m+1, e, A );
}
```

```
int partition ( int s, int e, int A[] )
{   int pivot, left, right;
    left = s+1;            right = e;          pivot = A[s];
    while (left <= right) {
        while ((A[right] >= pivot) && (left <= right))
            right--;

        while ((A[left] <= pivot) && (left <= right))
            left++;

        if ( left <= right )
            swap ( A[left], A[right] );
    }
    swap ( A[right], A[s] ); // A[s]: pivot;
    return right;
}
```

16	12	5	38	19	4	20	27

4	12	5	**16**	19	38	20	27
4	12	5	**16**	19	38	20	27
4	5	**12**	**16**	19	38	20	27
4	**5**	**12**	**16**	19	38	20	27
4	**5**	**12**	**16**	**19**	38	20	27

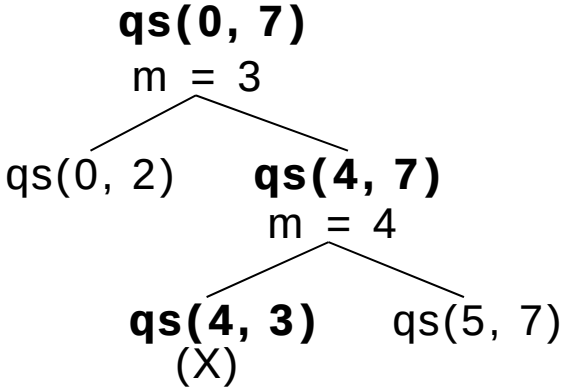


```
void quick_sort( int s, int e, int A[] )
{
    if ( s >= e )
        return;
    int m = partition ( s, e, A );
    quick_sort ( s, m-1, A );
    quick_sort ( m+1, e, A );
}
```

```
int partition ( int s, int e, int A[] )
{   int pivot, left, right;
    left = s+1;            right = e;          pivot = A[s];
    while (left <= right) {
        while ((A[right] >= pivot) && (left <= right))
            right--;

        while ((A[left] <= pivot) && (left <= right))
            left++;

        if ( left <= right )
            swap ( A[left], A[right] );
    }
    swap ( A[right], A[s] ); // A[s]: pivot;
    return right;
}
```

16	12	5	38	19	4	20	27

4	12	5	**16**	19	38	20	27
4	12	5	**16**	19	38	20	27
4	5	**12**	**16**	19	38	20	27
4	**5**	**12**	**16**	19	38	20	27
4	**5**	**12**	**16**	**19**	38	20	27

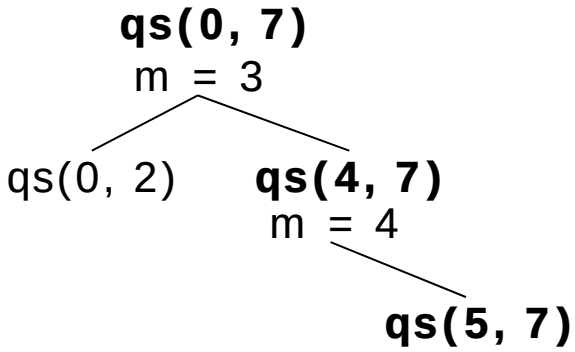


```
void quick_sort( int s, int e, int A[] )
{
    if ( s >= e )
        return;
    int m = partition ( s, e, A );
    quick_sort ( s, m-1, A );
    quick_sort ( m+1, e, A );
}
```

```
int partition ( int s, int e, int A[] )
{   int pivot, left, right;
    left = s+1;            right = e;          pivot = A[s];
    while (left <= right) {
        while ((A[right] >= pivot) && (left <= right))
            right--;

        while ((A[left] <= pivot) && (left <= right))
            left++;

        if ( left <= right )
            swap ( A[left], A[right] );
    }
    swap ( A[right], A[s] ); // A[s]: pivot;
    return right;
}
```

16	12	5	38	19	4	20	27

4	12	5	**16**	19	38	20	27
4	12	5	**16**	19	38	20	27
4	5	**12**	**16**	19	38	20	27
4	**5**	**12**	**16**	19	38	20	27
4	**5**	**12**	**16**	**19**	38	20	27

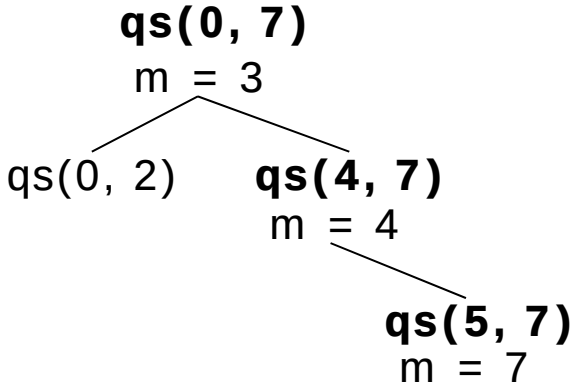


```
void quick_sort( int s, int e, int A[] )
{
    if ( s >= e )
        return;
    int m = partition ( s, e, A );
    quick_sort ( s, m-1, A );
    quick_sort ( m+1, e, A );
}
```

```
int partition ( int s, int e, int A[] )
{   int pivot, left, right;
    left = s+1;            right = e;          pivot = A[s];
    while (left <= right) {
        while ((A[right] >= pivot) && (left <= right))
            right--;

        while ((A[left] <= pivot) && (left <= right))
            left++;

        if ( left <= right )
            swap ( A[left], A[right] );
    }
    swap ( A[right], A[s] ); // A[s]: pivot;
    return right;
}
```

16	12	5	38	19	4	20	27

4	12	5	**16**	19	38	20	27
4	12	5	**16**	19	38	20	27
4	5	**12**	**16**	19	38	20	27
4	**5**	**12**	**16**	19	38	20	27
4	**5**	**12**	**16**	**19**	38	20	27
4	**5**	**12**	**16**	**19**	27	20	**38**

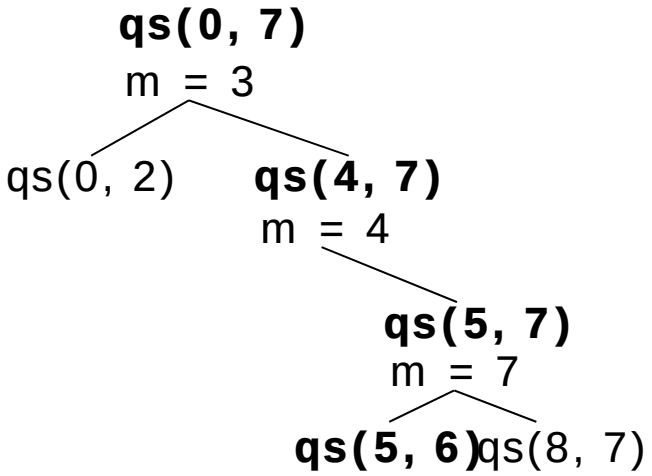


```
void quick_sort( int s, int e, int A[] )
{
    if ( s >= e )
        return;
    int m = partition ( s, e, A );
    quick_sort ( s, m-1, A );
    quick_sort ( m+1, e, A );
}
```

```
int partition ( int s, int e, int A[] )
{   int pivot, left, right;
    left = s+1;            right = e;          pivot = A[s];
    while (left <= right) {
        while ((A[right] >= pivot) && (left <= right))
            right--;

        while ((A[left] <= pivot) && (left <= right))
            left++;

        if ( left <= right )
            swap ( A[left], A[right] );
    }
    swap ( A[right], A[s] ); // A[s]: pivot;
    return right;
}
```

16	12	5	38	19	4	20	27

4	12	5	**16**	19	38	20	27
4	12	5	**16**	19	38	20	27
4	5	**12**	**16**	19	38	20	27
4	**5**	**12**	**16**	19	38	20	27
4	**5**	**12**	**16**	**19**	27	20	38
4	**5**	**12**	**16**	**19**	27	20	**38**

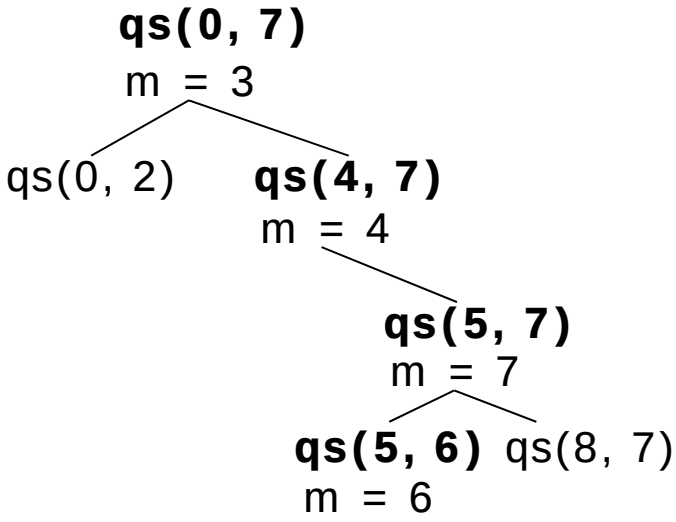


```
void quick_sort( int s, int e, int A[] )
{
    if ( s >= e )
        return;
    int m = partition ( s, e, A );
    quick_sort ( s, m-1, A );
    quick_sort ( m+1, e, A );
}
```

```
int partition ( int s, int e, int A[] )
{   int pivot, left, right;
    left = s+1;            right = e;          pivot = A[s];
    while (left <= right) {
        while ((A[right] >= pivot) && (left <= right))
            right--;

        while ((A[left] <= pivot) && (left <= right))
            left++;

        if ( left <= right )
            swap ( A[left], A[right] );
    }
    swap ( A[right], A[s] ); // A[s]: pivot;
    return right;
}
```

16	12	5	38	19	4	20	27

4	12	5	**16**	19	38	20	27
4	12	5	**16**	19	38	20	27
4	5	**12**	**16**	19	38	20	27
4	**5**	**12**	**16**	19	38	20	27
4	**5**	**12**	**16**	**19**	27	20	38
4	**5**	**12**	**16**	**19**	20	**27**	**38**

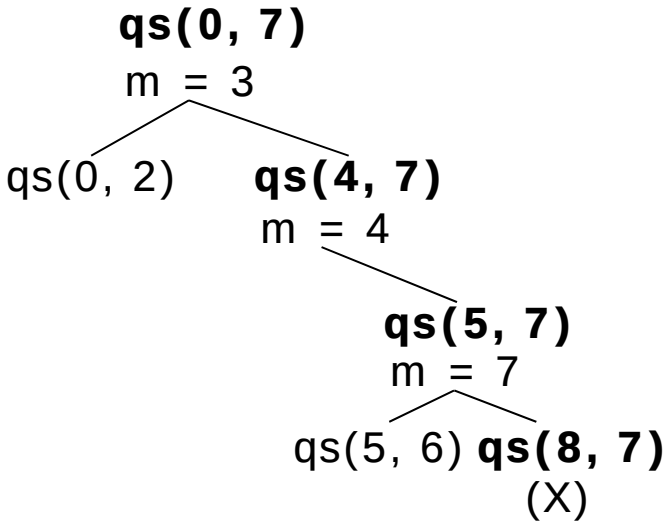


```
void quick_sort( int s, int e, int A[] )
{
    if ( s >= e )
        return;
    int m = partition ( s, e, A );
    quick_sort ( s, m-1, A );
    quick_sort ( m+1, e, A );
}
```

```
int partition ( int s, int e, int A[] )
{   int pivot, left, right;
    left = s+1;            right = e;          pivot = A[s];
    while (left <= right) {
        while ((A[right] >= pivot) && (left <= right))
            right--;

        while ((A[left] <= pivot) && (left <= right))
            left++;

        if ( left <= right )
            swap ( A[left], A[right] );
    }
    swap ( A[right], A[s] ); // A[s]: pivot;
    return right;
}
```

16	12	5	38	19	4	20	27

4	12	5	**16**	19	38	20	27
4	12	5	**16**	19	38	20	27
4	5	**12**	**16**	19	38	20	27
4	**5**	**12**	**16**	19	38	20	27
4	**5**	**12**	**16**	**19**	27	20	38
4	**5**	**12**	**16**	**19**	20	**27**	**38**



```
void quick_sort( int s, int e, int A[] )
{
    if ( s >= e )
        return;
    int m = partition ( s, e, A );
    quick_sort ( s, m-1, A );
    quick_sort ( m+1, e, A );
}
```

```
int partition ( int s, int e, int A[] )
{   int pivot, left, right;
    left = s+1;            right = e;          pivot = A[s];
    while (left <= right) {
        while ((A[right] >= pivot) && (left <= right))
            right--;

        while ((A[left] <= pivot) && (left <= right))
            left++;

        if ( left <= right )
            swap ( A[left], A[right] );
    }
    swap ( A[right], A[s] ); // A[s]: pivot;
    return right;
}
```

qs(0, 7)

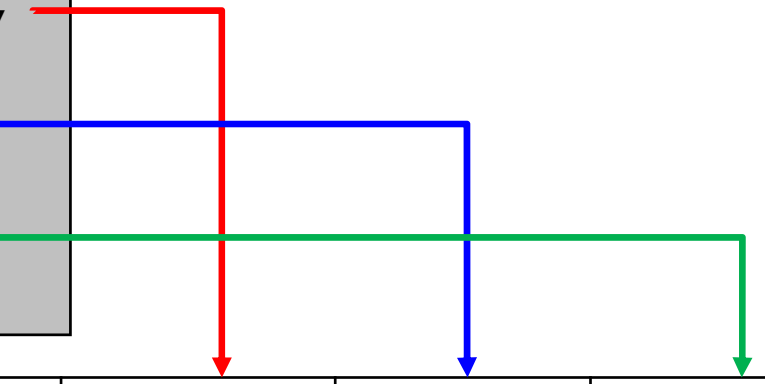
16	12	5	38	19	4	20	27

4	12	5	**16**	19	38	20	27
4	12	5	**16**	19	38	20	27
4	5	**12**	**16**	19	38	20	27
4	**5**	**12**	**16**	19	38	20	27
4	**5**	**12**	**16**	**19**	27	20	38
4	**5**	**12**	**16**	**19**	20	**27**	38
4	**5**	**12**	**16**	**19**	**20**	**27**	**38**

Done!

- Quick sort (Performance analysis)

```
quick_sort( )
{
    int m = partition ( s, e, A );

    quick_sort ( s, m-1, A );

    quick_sort ( m+1, e, A );
}
```



case	example	partition	q_sort ( )	q_sort ( )
best	{5, 3, 4, 8, 7, 2, 9, 1 }	O(n)	T(n/2)	T(n/2)
worst	{9, 8, 7, 5, 4, 3, 2, 1 }	O(n)	T(0)	T(n – 1)
average	{1, 3, 4, 8, 7, 2, 9, 5 }	O(n)	T(k – 1)	T(n – k)

- Quick sort (Performance analysis)
  - Recurrence relation
  - Best case

$$T(n) = 2\,T\left(\frac{n}{2}\right) + O(n)$$

  - Average case

$$T(n) = \frac{1}{n}\sum_{k=1}^{n}(T(k-1) + T(n-k)) + O(n)$$

  - Worst case

$$T(n) = T(n-1) + O(n)$$

- Quick sort (Performance analysis)
  - Comparison to merge sort

	Example	Quick sort	Merge sort
Best case	5 1 6 3 4 8 7 2	O(n log n)	O(n log n)
Worst case	1 2 3 4 5 6 7 8 8 7 6 5 4 3 2 1	$O(n^2)$	O(n log n)
Average case	4 1 3 6 5 2 9 8	O(n log n)	O(n log n)

- Comparison

	degenerate case	divide	conquer	combine	performance
tournament	n = 1 (s = e)	m = (s+e)/2	champ (s,m); champ (m+1,e);	win (LW, RW);	2T(n/2) + O(1) = O(n)
binary search	n = 1 (s = e)	m = (s+e)/2	bs (s, m-1); or bs (m+1, e);	-	T(n/2) + O(1) = O(log n)

integer multiplication	n = 1	s = n/2	mult (w+x, y+z);
			mult (w, y);
			mult (x, z);
				p 2^n +
				(r – p – q) 2^s +
				q;
					3T(n/2) + O(n)
					=O($n^{\log\_2 3}$)

merge sort	n = 1 (s = e)	m = (s+e)/2	ms (s, m); ms (m+1, e);	merge (s, m, e);	2T(n/2) + O(n) = O(n log n)
quick sort	n = 1 (s>=e)	m = partition ( );	qs (s, m-1); qs (m+1, e);	-	O(n log n)
median					
matrix multiplication					

퀴즈 4

- Divide & conquer에 기반한 정렬 알고리즘에 대한 설명이다 옳지 않은 것을 모두 고르시오.

  (a) Quick sort와 merge sort는 worst case에 대한 시간 복잡도가 같다.

  (b) Merge sort는 best case의 시간 복잡도가 worst case의 시간 복잡도보다 더 좋다.

  (c) Quick sort는 average case의 시간 복잡도가 worst case의 시간 복잡도보다 더 좋다.

  (d) Quick sort는 binary search와 같이 combine 과정을 요구하지 않는다.

  (e) Merge sort의 divide 과정과 quick sort의 divide 과정의 시간 복잡도는 같다.