QuickSort

QuickSort algorithm sorts an array of items (could be integers, strings, or any other objects that can be compared to each other with operators less than, equal to, or greater than).

QuickSort follows divide-and-conquer technique:
1) It chooses an element of the given input array, usually called *pivot*, and partitions all items of the array into two subarrays: the first subarray has items less than or equal to *pivot* and the second subarray has items strictly greater than *pivot*.
2) The two subarrays then are sorted recursively.
3) Because QuickSort sorts items *in place*, there is no need to combine the solutions, the entire array is sorted.

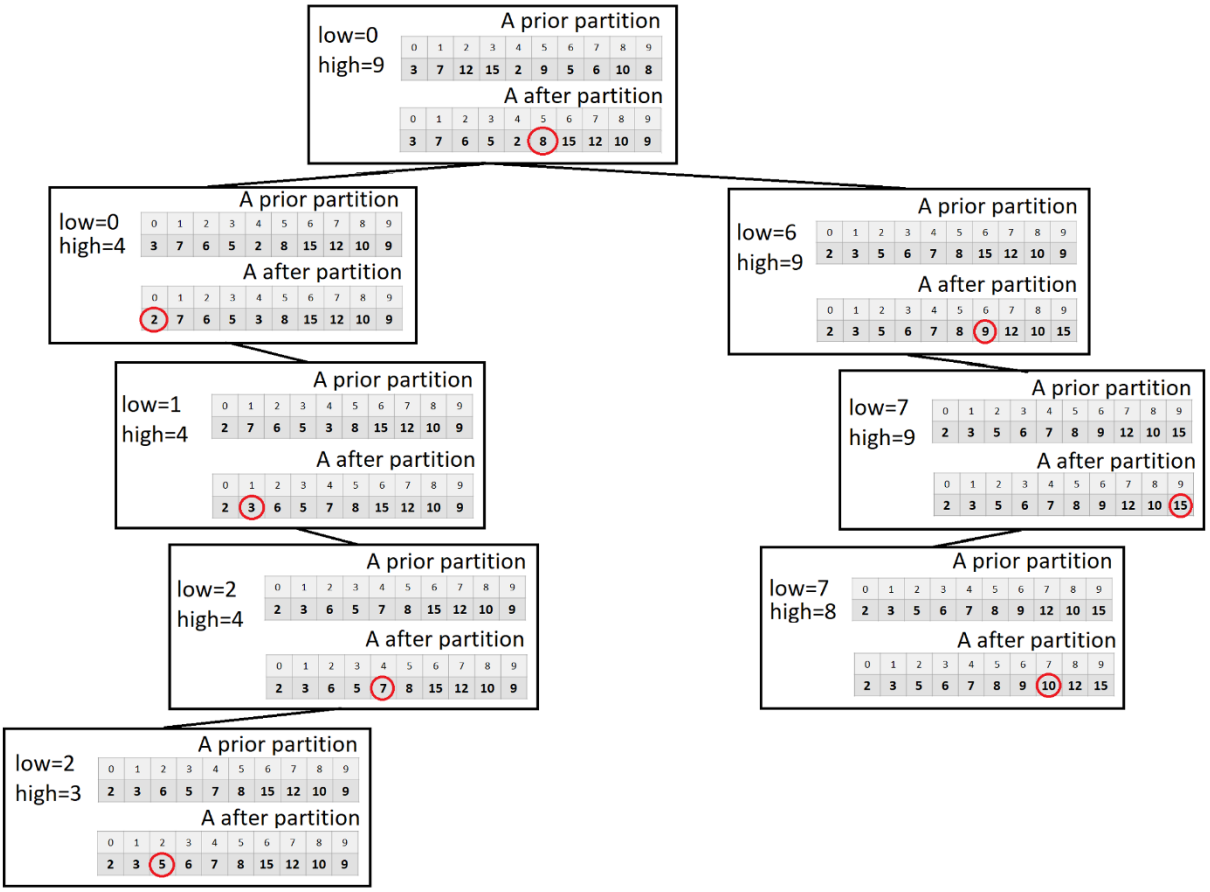
Here, we will show how QuickSort sorts an array of integers, but keep in mind that it can sort an array of any items that can be ordered.

Input: An vector A of integers, A[0,…,n-1], of size *n*; and two integer indices *low* and *high* that are indices to the first and last element of the current range. Initially *low* is 0 and *high* is *n*-1.
Output: vector A is sorted in non-decreasing order.

```
void quicksort(vector<int> &A, int low, int high){
	if(low < high){
		int pivotIndex = partition(A, low, high);
		quicksort(A, low, pivotIndex - 1);
		quicksort(A, pivotIndex + 1, high);
	}
}
```

```
int partition(vector<int> &A, int low, int high){
	int pivot = A[high];
	int i = low, j = high - 1;
	while(i <= j){
		while(i < high && A[i] <= pivot)
			i++;
		while(j >= low && A[j] > pivot)
			j--;
		if(i < j){//swap A[i] and A[j]
			int temp = A[i];
			A[i] = A[j];
			A[j] = temp;
			i++;
			j--;
		}//if
	}//while
	//swap A[i] and pivot
	A[high] = A[i];
	A[i] = pivot;
	return i;
}
```

Example of QuickSort



Loop Invariant:
At the beginning of each iteration of the outmost ***while*** loop of ***partition*** function, the subarray A[low…i-1] contains the items of A that are less than or equal to *pivot* and the subarray A[j+1…high-1] contain the items of A that are greater than *pivot*.
Try to prove this by yourself using two steps:
1. Initialization.
2. Maintenance (if true at k-th iteration, then this is also true at k+1-th iteration).
3. Termination: what happens when *while* loop finishes execution in terms of correctness of *partition*?