Quicksort

- Our next sorting algorithm is Quicksort.
- It is one of the fastest sorting algorithms known and is the method of choice in most sorting libraries.
- Quicksort is based on the divide and conquer strategy.

Quicksort

```
QUICKSORT( array A, int p, int r)
if (r > p)
   then
     i ← a random index from [p..r]
       swap A[i] with A[p]
     q ← PARTITION(A, p, r)
     QUICKSORT(A, p, q – 1)
     QUICKSORT(A, q + 1, r)
```

Partition Algorithm

Recall that the partition algorithm partitions the array A[p..r] into three sub arrays about a pivot element x.

- A[p..q – 1] whose elements are less than or equal to x,
- A[q] = x,
- A[q + 1..r] whose elements are greater than x

Choosing the Pivot

- We will choose the first element of the array as the pivot, i.e. x = A[p].
- If a different rule is used for selecting the pivot, we can swap the chosen element with the first element.
- We will choose the pivot randomly.

Partition Algorithm

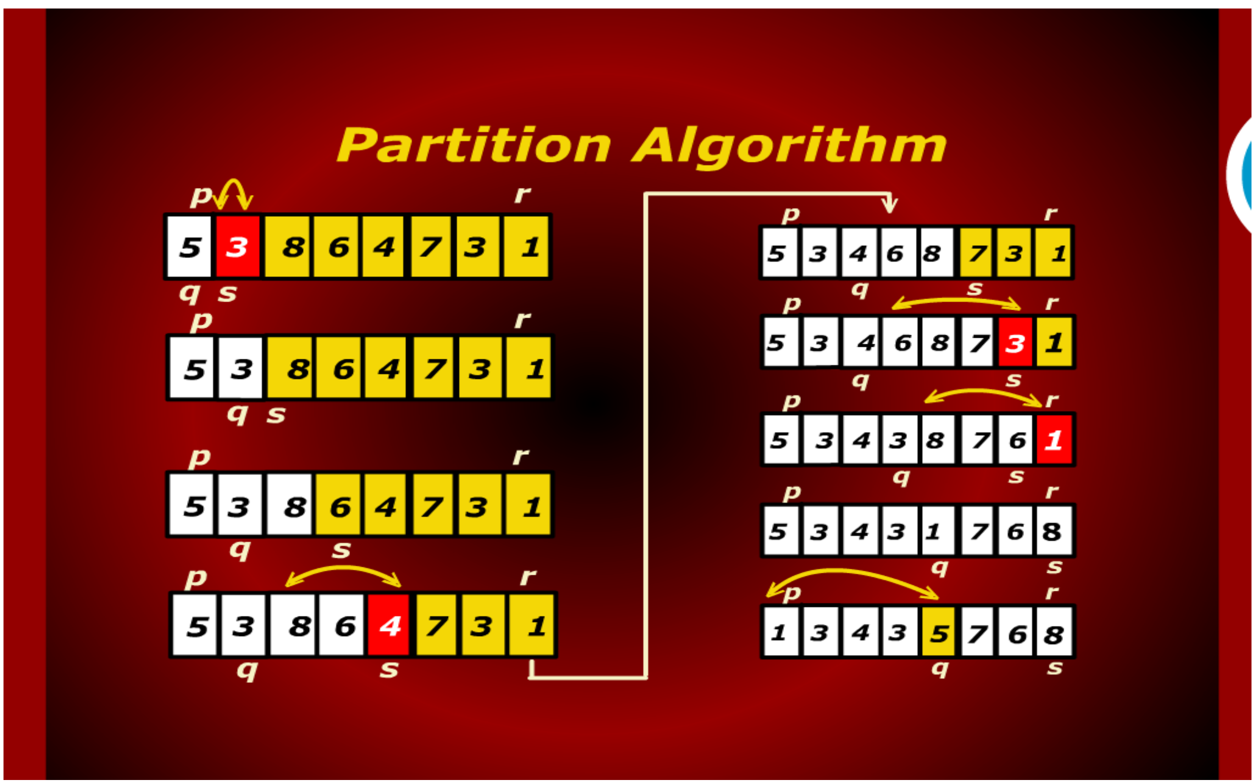
The algorithm works by maintaining the following *invariant condition.*

1. A[p] = x is the pivot value.
2. A[p..q – 1] contains elements that are less than x.
3. A[q + 1..s – 1] contains elements that are greater than or equal to x
4. A[s..r] contains elements whose values are currently unknown.

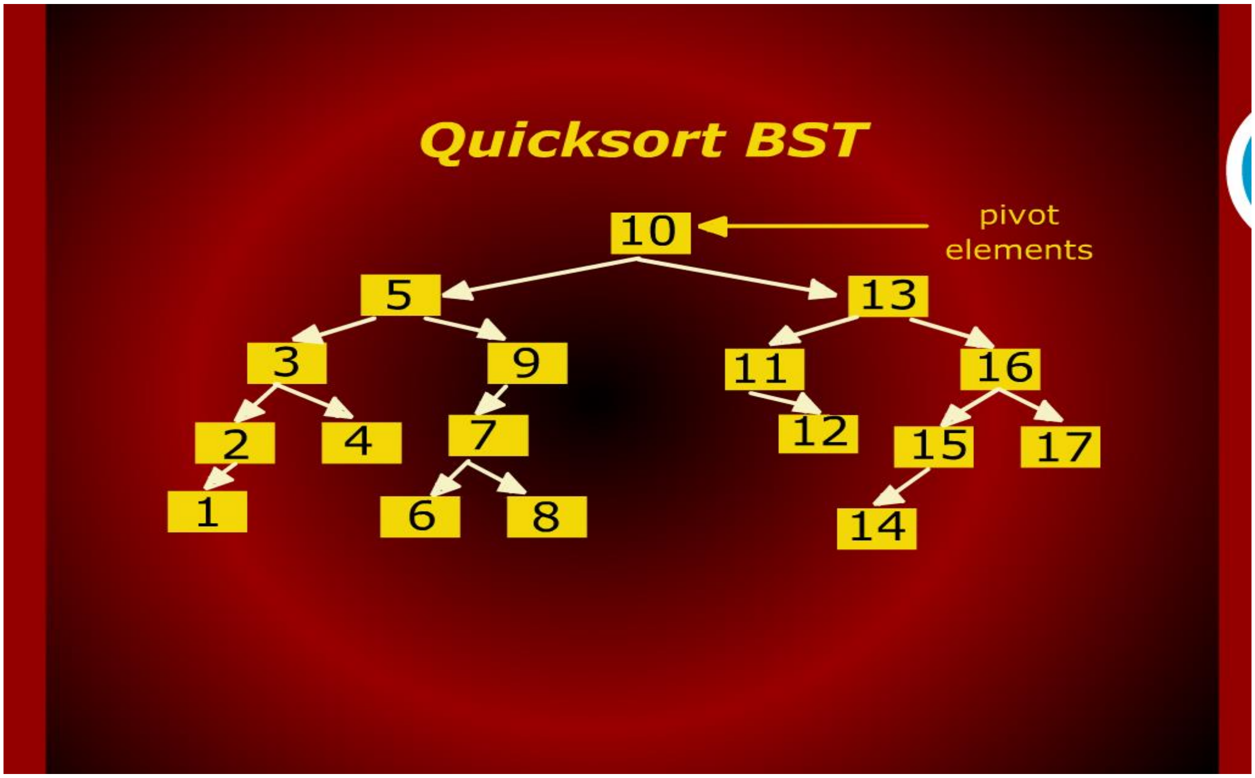
Partition Algorithm

```
PARTITION( array A, int p, int r)
  x ← A[p]
  q ← p
  for s ← p + 1 to r
  do if (A[s] < x)
        then q ← q + 1
             swap A[q] with A[s]

   swap A[p] with A[q]
  return q
```



It is interesting to note (but not surprising) that the pivots form a binary search tree


Quicksort BST
pivot
elements
10
5
13
3
9
11
16
2
4
7
12
15
17
1
6
8
14