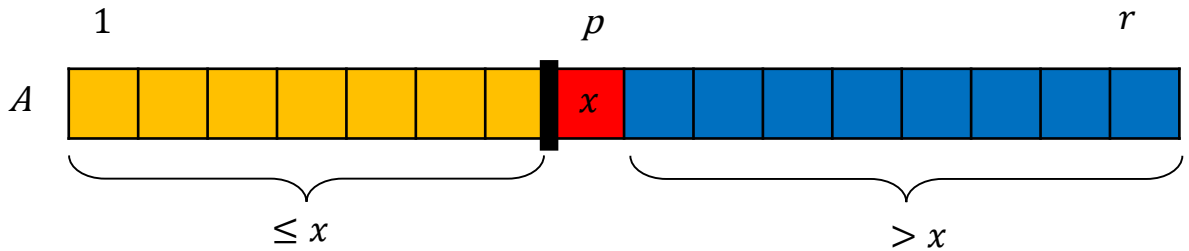
Introduction to Algorithms (Chapter7: Quick Sorting)

Kyuseok Shim

Electrical and Computer Engineering
Seoul National University

Quicksort

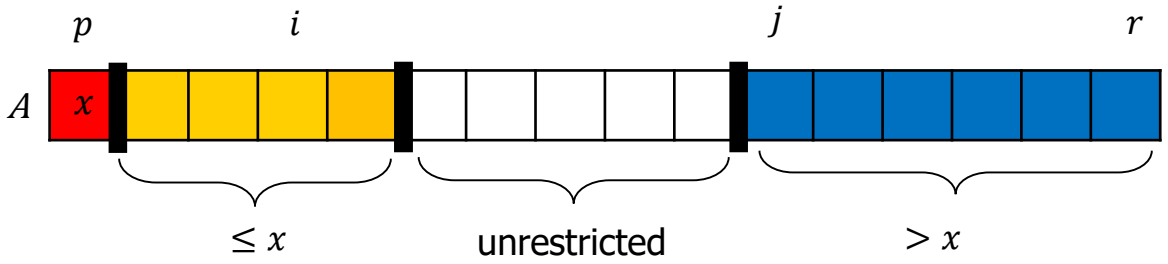
- Divide: Paritition the array $A[1..r]$ into two nonempty subarrays $A[1..p-1]$ and $A[p+1..r]$ around the pivot $A[p]$ such that
  - (every element in $A[1..p-1]$) $\leq A[p]$
  - $A[p] \leq$ (every element in $A[p+1..r]$)



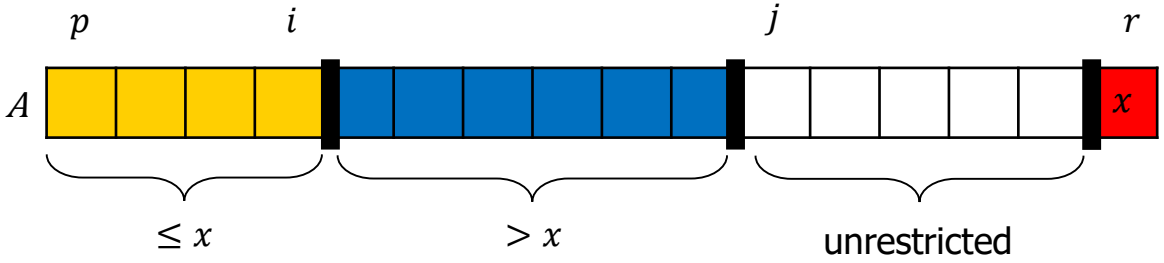
- Conquer: Sort each of $A[1..p]$ and $A[p+1..r]$ by recursive calls to Quick sort
- Combine: Do nothing

Quicksort

- We investigate two versions of quick sort based one different partitioning algorithms
  - One in the data structure text book



  - Another one in the algorithm text book



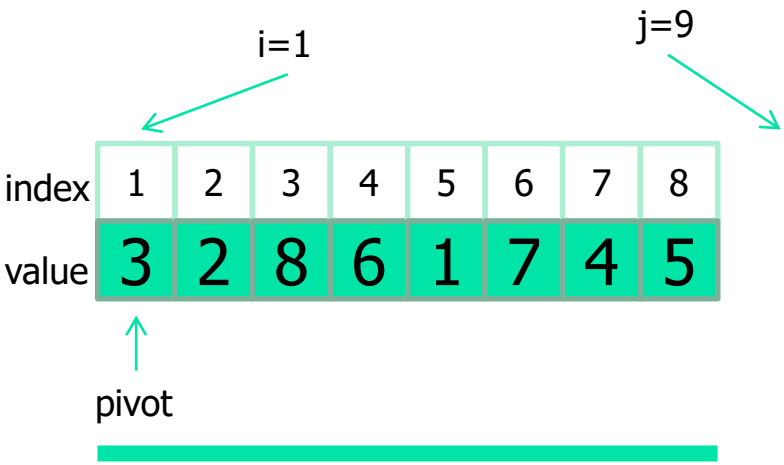
Quick Sort Code

```
template <class T>
void QuickSort(T *a, const int left, const int right)
{ // Sort a[left:right] into nondecreasing order.
  // a[left] is arbitrarily chosen as the pivot. Variables i and j
  // are used to partition the subarray so that at any time a[m] ≤ pivot, m < i
  // and a[m] ≥ pivot, m > j. It is assumed that a[left] ≤ a[right + 1]
     if (left < right) {
          int i=left,
          j = right + 1,
          pivot = a[left];
          do {
               do i++; while (a[i] < pivot);
               do j--; while (a[j] > pivot);
               if(i < j) swap(a[i], a[j]);
          } while (i<j);
          swap(a[left], a[j]);

          QuickSort(a, left, j-1);
          QuickSort(a, j+1, right);
     }
}
```

Quicksort

```
if (left < right) {
        int i=left,
        j = right + 1,
        pivot = a[left];
        do {
            do i++; while (a[i] < pivot);
            do j--; while (a[j] > pivot);
            if(i < j) swap(a[i], a[j]);
        } while (i<j);
        swap(a[left], a[j]);

        QuickSort(a, left, j-1);
        QuickSort(a, j+1, right);
    }
```

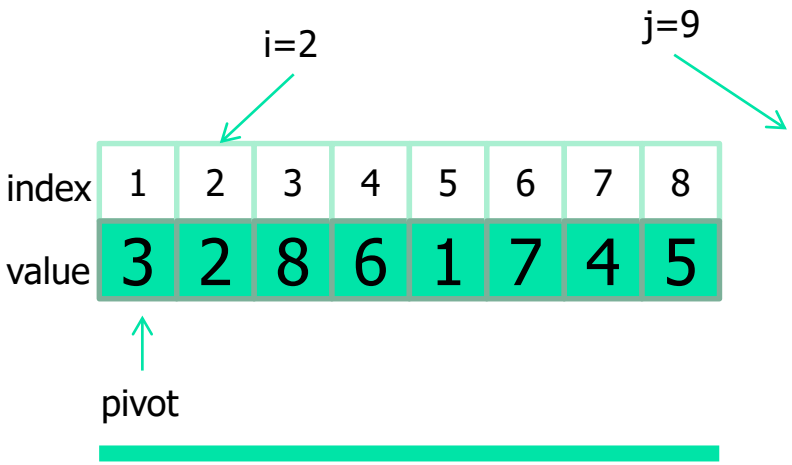
left = 1
right = 8



Quicksort

```
if (left < right) {
        int i=left,
        j = right + 1,
        pivot = a[left];
        do {
            do i++; while (a[i] < pivot);
            do j--; while (a[j] > pivot);
            if(i < j) swap(a[i], a[j]);
        } while (i<j);
        swap(a[left], a[j]);

        QuickSort(a, left, j-1);
        QuickSort(a, j+1, right);
    }
```

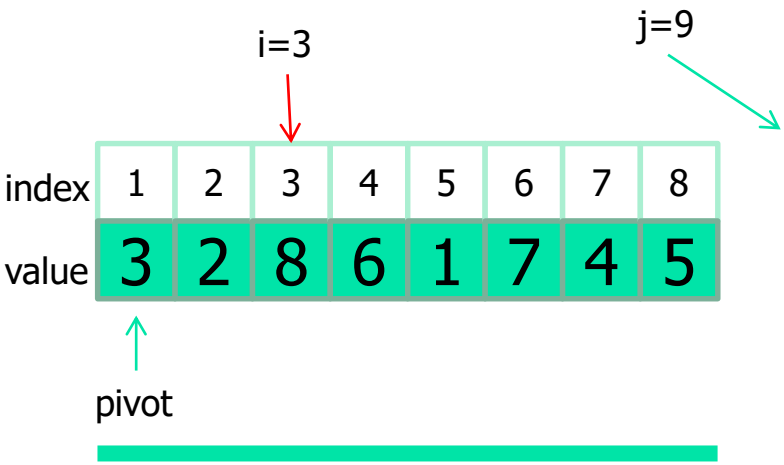
left = 1
right = 8



Quicksort

```
if (left < right) {
        int i=left,
        j = right + 1,
        pivot = a[left];
        do {
            do i++; while (a[i] < pivot);
            do j--; while (a[j] > pivot);
            if(i < j) swap(a[i], a[j]);
        } while (i<j);
        swap(a[left], a[j]);

        QuickSort(a, left, j-1);
        QuickSort(a, j+1, right);
    }
```

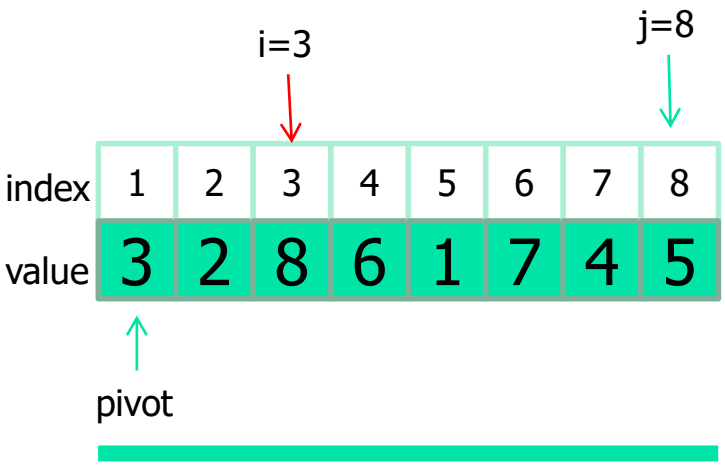
left = 1
right = 8



Quicksort

```
if (left < right) {
        int i=left,
        j = right + 1,
        pivot = a[left];
        do {
            do i++; while (a[i] < pivot);
            do j--; while (a[j] > pivot);
            if(i < j) swap(a[i], a[j]);
        } while (i<j);
        swap(a[left], a[j]);

        QuickSort(a, left, j-1);
        QuickSort(a, j+1, right);
    }
```

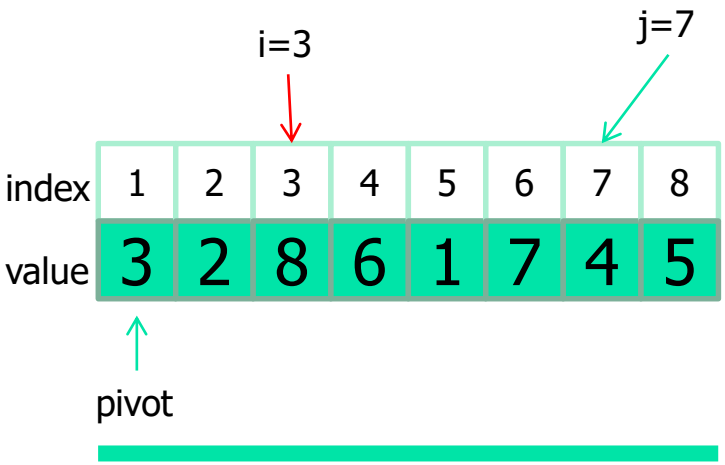
left = 1
right = 8



Quicksort

```
if (left < right) {
        int i=left,
        j = right + 1,
        pivot = a[left];
        do {
            do i++; while (a[i] < pivot);
            do j--; while (a[j] > pivot);
            if(i < j) swap(a[i], a[j]);
        } while (i<j);
        swap(a[left], a[j]);

        QuickSort(a, left, j-1);
        QuickSort(a, j+1, right);
    }
```

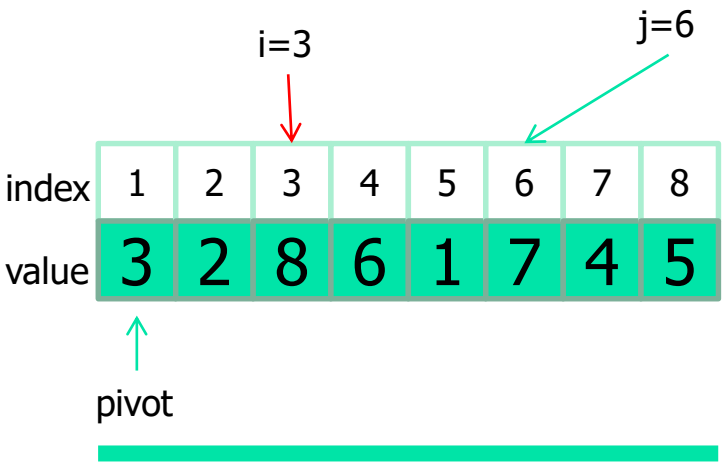
left = 1
right = 8



Quicksort

```
if (left < right) {
        int i=left,
        j = right + 1,
        pivot = a[left];
        do {
            do i++; while (a[i] < pivot);
            do j--; while (a[j] > pivot);
            if(i < j) swap(a[i], a[j]);
        } while (i<j);
        swap(a[left], a[j]);

        QuickSort(a, left, j-1);
        QuickSort(a, j+1, right);
    }
```

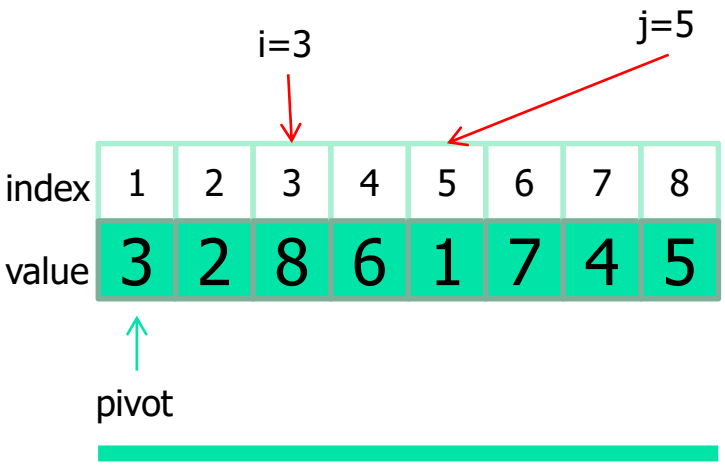
left = 1
right = 8



Quicksort

```
if (left < right) {
        int i=left,
        j = right + 1,
        pivot = a[left];
        do {
            do i++; while (a[i] < pivot);
            do j--; while (a[j] > pivot);
            if(i < j) swap(a[i], a[j]);
        } while (i<j);
        swap(a[left], a[j]);

        QuickSort(a, left, j-1);
        QuickSort(a, j+1, right);
    }
```

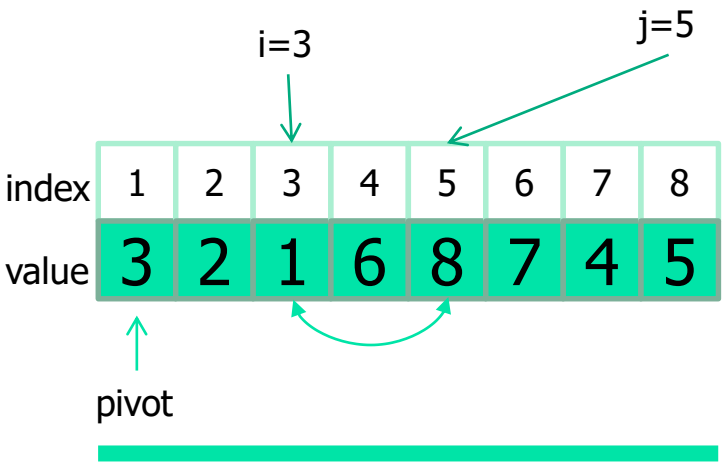
left = 1
right = 8



Quicksort

```
if (left < right) {
        int i=left,
        j = right + 1,
        pivot = a[left];
        do {
            do i++; while (a[i] < pivot);
            do j--; while (a[j] > pivot);
            if(i < j) swap(a[i], a[j]);
        } while (i<j);
        swap(a[left], a[j]);

        QuickSort(a, left, j-1);
        QuickSort(a, j+1, right);
    }
```

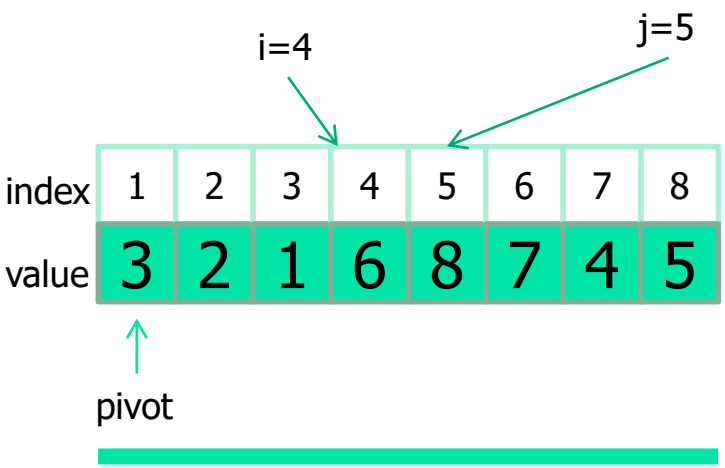
left = 1
right = 8



Quicksort

```
if (left < right) {
        int i=left,
        j = right + 1,
        pivot = a[left];
        do {
            do i++; while (a[i] < pivot);
            do j--; while (a[j] > pivot);
            if(i < j) swap(a[i], a[j]);
        } while (i<j);
        swap(a[left], a[j]);

        QuickSort(a, left, j-1);
        QuickSort(a, j+1, right);
    }
```

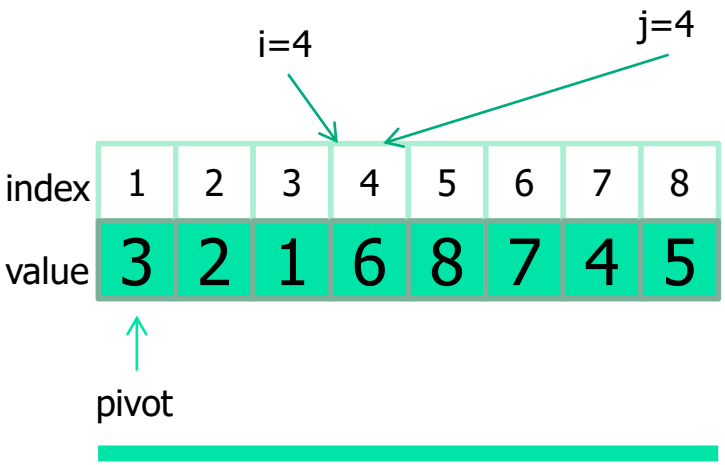
left = 1
right = 8



Quicksort

```
if (left < right) {
        int i=left,
        j = right + 1,
        pivot = a[left];
        do {
            do i++; while (a[i] < pivot);
            do j--; while (a[j] > pivot);
            if(i < j) swap(a[i], a[j]);
        } while (i<j);
        swap(a[left], a[j]);

        QuickSort(a, left, j-1);
        QuickSort(a, j+1, right);
    }
```

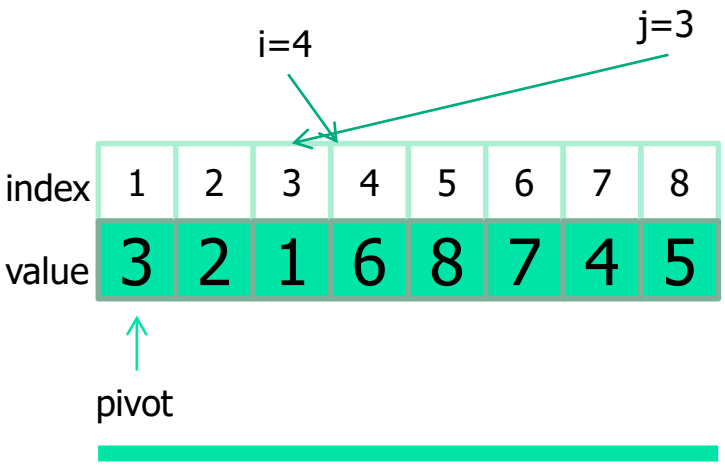
left = 1
right = 8



Quicksort

```
if (left < right) {
        int i=left,
        j = right + 1,
        pivot = a[left];
        do {
            do i++; while (a[i] < pivot);
            do j--; while (a[j] > pivot);
            if(i < j) swap(a[i], a[j]);
        } while (i<j);
        swap(a[left], a[j]);

        QuickSort(a, left, j-1);
        QuickSort(a, j+1, right);
    }
```

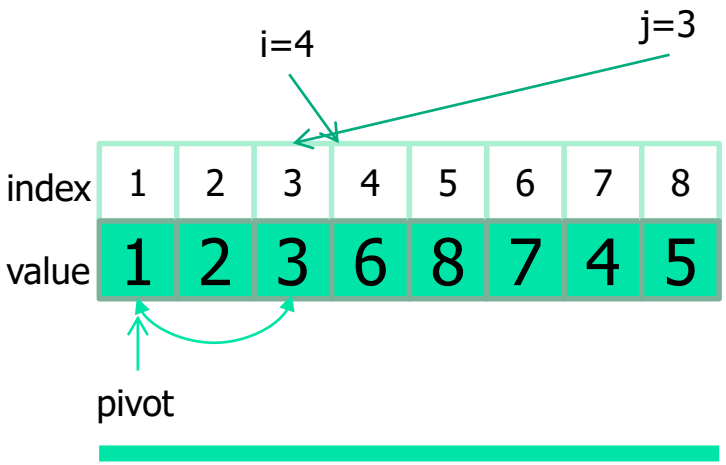
left = 1
right = 8



Quicksort

```
if (left < right) {
        int i=left,
        j = right + 1,
        pivot = a[left];
        do {
            do i++; while (a[i] < pivot);
            do j--; while (a[j] > pivot);
            if(i < j) swap(a[i], a[j]);
        } while (i<j);
        swap(a[left], a[j]);

        QuickSort(a, left, j-1);
        QuickSort(a, j+1, right);
    }
```

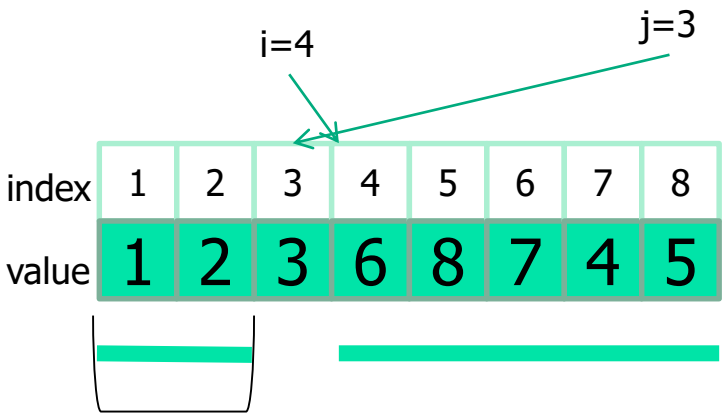
left = 1
right = 8



Quicksort

```
if (left < right) {
        int i=left,
        j = right + 1,
        pivot = a[left];
        do {
            do i++; while (a[i] < pivot);
            do j--; while (a[j] > pivot);
            if(i < j) swap(a[i], a[j]);
        } while (i<j);
        swap(a[left], a[j]);

        QuickSort(a, left, j-1);
        QuickSort(a, j+1, right);
    }
```

left = 1
right = 8



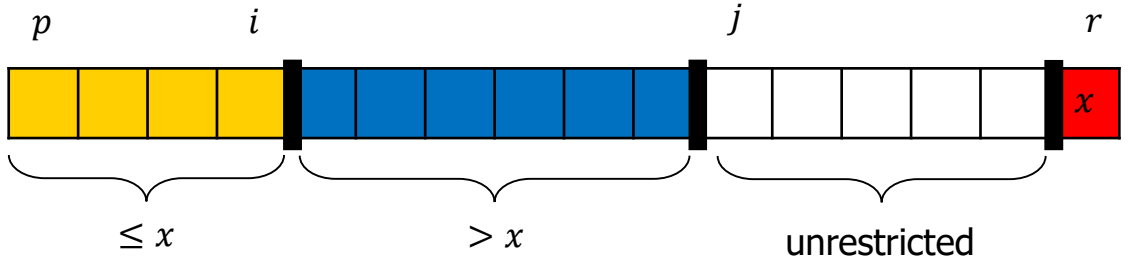
Quicksort

```
PARTITION(A, p, r)
1.    x = A[r]
2.    i = p – 1
3.    for j = p to r – 1
4.        if A[j] ≤ x
5.            i = i + 1
6.            exchange A[i] with A[j]
7.    exchange A[i +1] with A[r]
8.    return i + 1
```

```
QUICKSORT(A, p, r)
1.    if p < r
2.       q = PARTITION(A, p, r)
3.       QUICKSORT(A, p, q-1)
4.       QUICKSORT(A, q + 1, r)
```

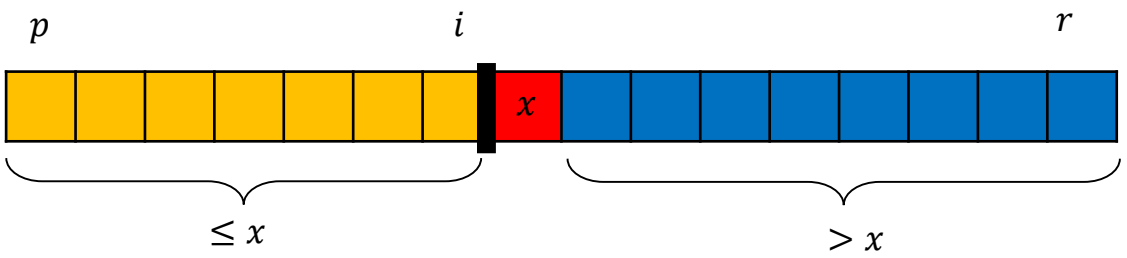
Quicksort

```
PARTITION(A, p, r)
1.    x = A[r]
2.    i = p – 1
3.    for j = p to r – 1
4.        if A[j] ≤ x
5.            i = i + 1
6.            exchange A[i] with A[j]
7.    exchange A[i +1] with A[r]
8.    return i + 1
```



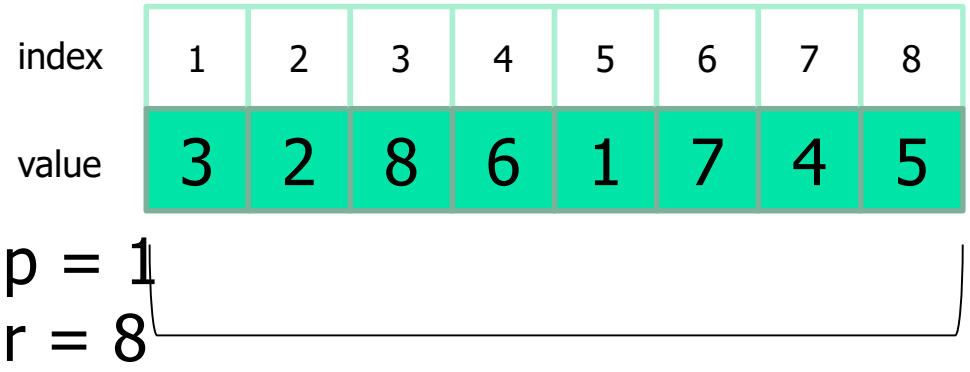
Quicksort

```
PARTITION(A, p, r)
1.    x = A[r]
2.    i = p – 1
3.    for j = p to r – 1
4.        if A[j] ≤ x
5.            i = i + 1
6.            exchange A[i] with A[j]
7.    exchange A[i +1] with A[r]
8.    return i + 1
```



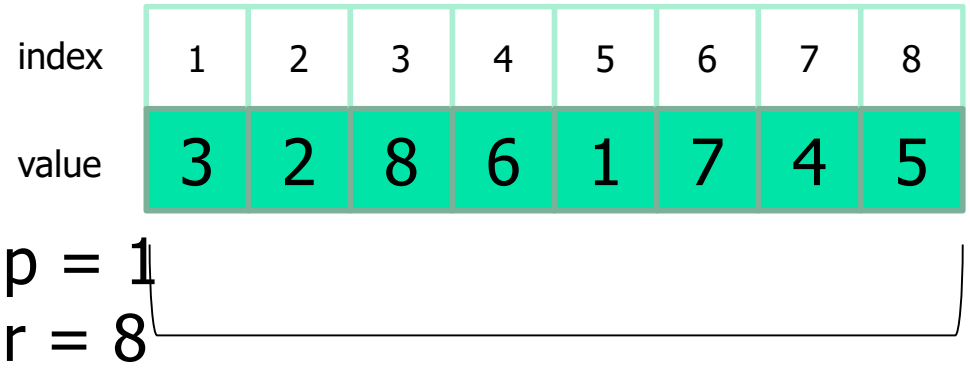
Quicksort

```
QUICKSORT(A, p, r)
1.   if p < r
2.      q = PARTITION(A, p, r)
3.      QUICKSORT(A, p, q-1)
4.      QUICKSORT(A, q + 1, r)
```



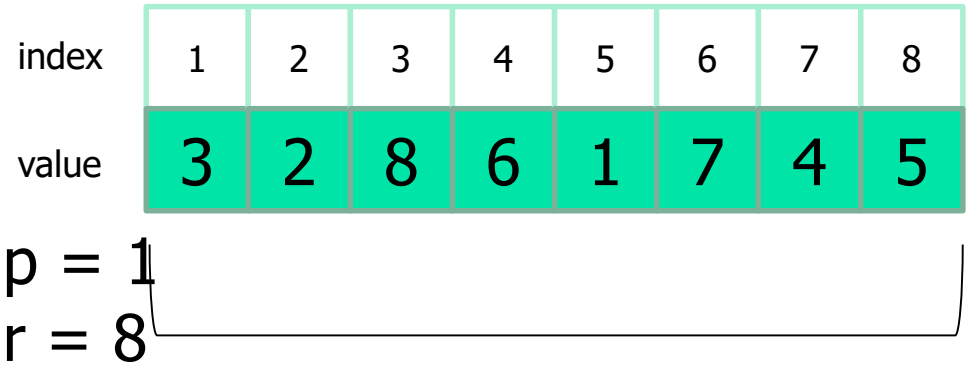
Quicksort

```
QUICKSORT(A, p, r)
1.    if p < r
2.       q = PARTITION(A, p, r)
3.       QUICKSORT(A, p, q-1)
4.       QUICKSORT(A, q + 1, r)
```



Quicksort

```
QUICKSORT(A, p, r)
1.    if p < r
2.       q = PARTITION(A, p, r)
3.       QUICKSORT(A, p, q-1)
4.       QUICKSORT(A, q + 1, r)
```



Quicksort

```
PARTITION(A, p, r)
1.      x = A[r]
2.      i = p – 1
3.      for j = p to r – 1
4.          if A[j] ≤ x
5.              i = i + 1
6.              exchange A[i] with A[j]
7.      exchange A[i +1] with A[r]
8.      return i + 1
```

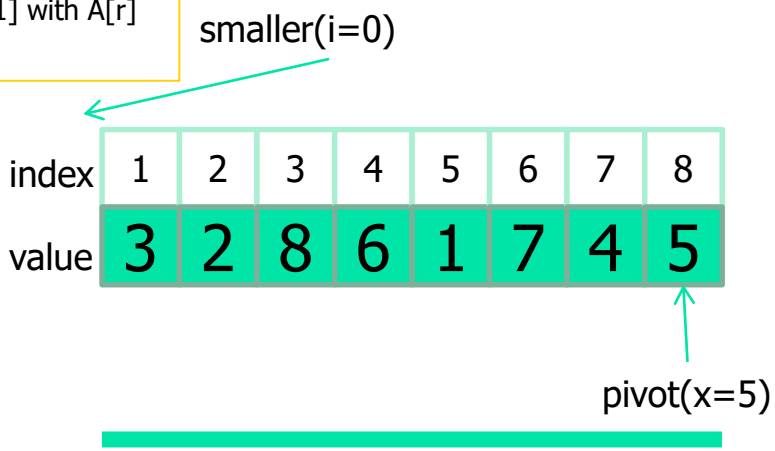
p = 1
r = 8

index	1	2	3	4	5	6	7	8
value	3	2	8	6	1	7	4	5

Quicksort

```
PARTITION(A, p, r)
1.    x = A[r]
2.    i = p – 1
3.    for j = p to r – 1
4.       if A[j] ≤ x
5.          i = i + 1
6.          exchange A[i] with A[j]
7.    exchange A[i +1] with A[r]
8.    return i + 1
```

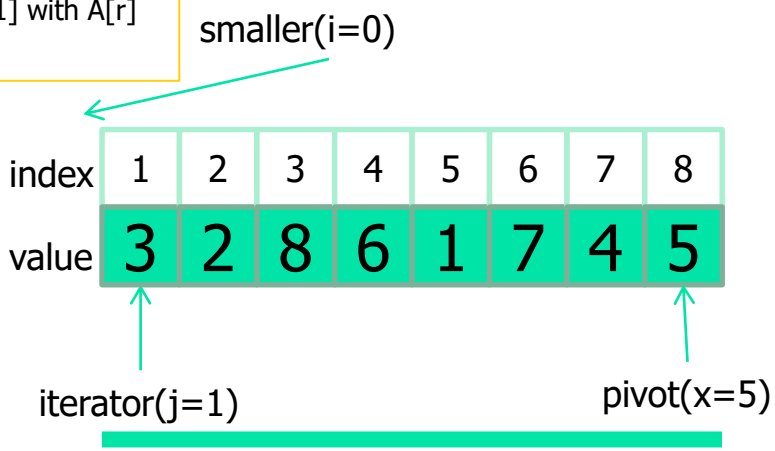
p = 1
r = 8



Quicksort

```
PARTITION(A, p, r)
1.    x = A[r]
2.    i = p – 1
3.    for j = p to r – 1
4.       if A[j] ≤ x
5.          i = i + 1
6.          exchange A[i] with A[j]
7.    exchange A[i +1] with A[r]
8.    return i + 1
```

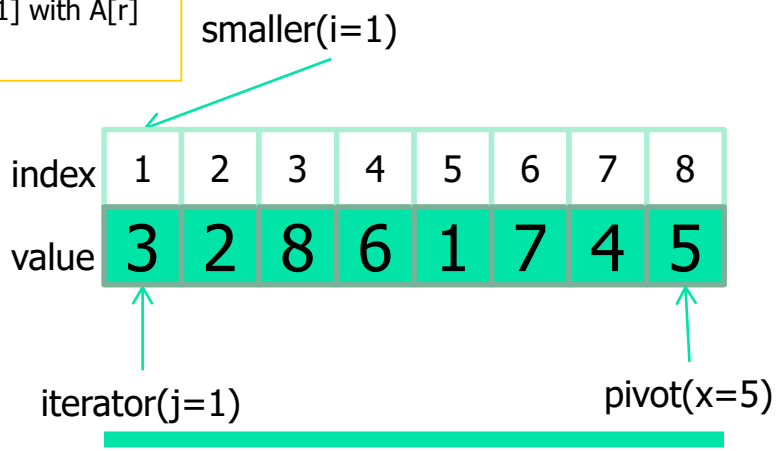
p = 1
r = 8



Quicksort

```
PARTITION(A, p, r)
1.      x = A[r]
2.      i = p – 1
3.      for j = p to r – 1
4.          if A[j] ≤ x
5.              i = i + 1
6.              exchange A[i] with A[j]
7.      exchange A[i +1] with A[r]
8.      return i + 1
```

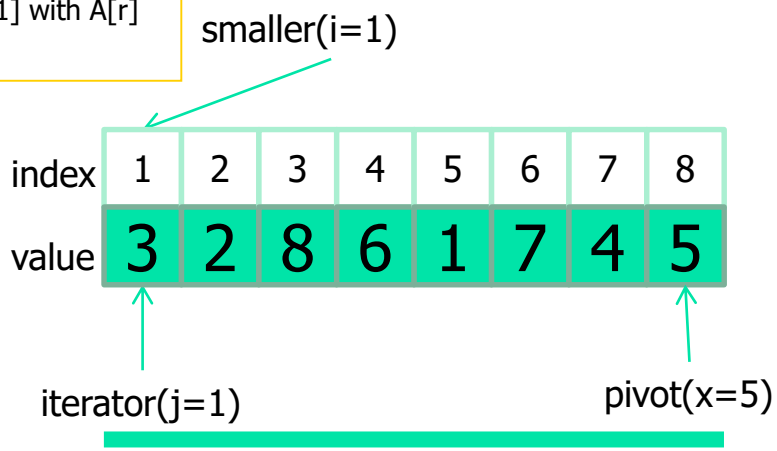
p = 1
r = 8



Quicksort

```
PARTITION(A, p, r)
1.      x = A[r]
2.      i = p – 1
3.      for j = p to r – 1
4.          if A[j] ≤ x
5.              i = i + 1
6.              exchange A[i] with A[j]
7.      exchange A[i +1] with A[r]
8.      return i + 1
```

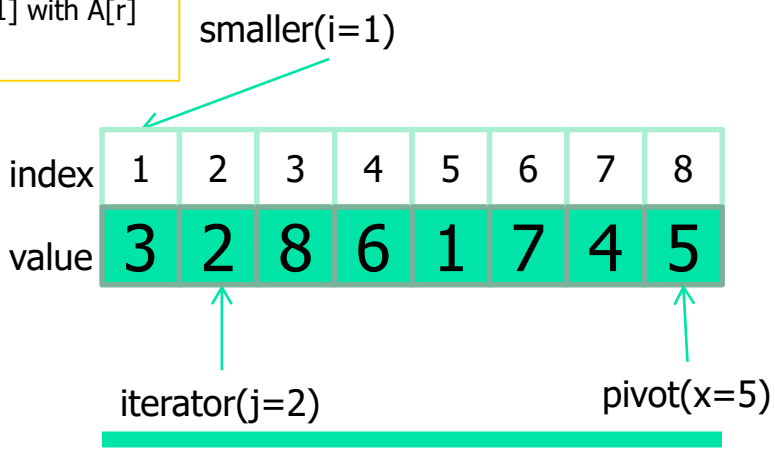
p = 1
r = 8



Quicksort

```
PARTITION(A, p, r)
1.      x = A[r]
2.      i = p – 1
3.      for j = p to r – 1
4.          if A[j] ≤ x
5.              i = i + 1
6.              exchange A[i] with A[j]
7.      exchange A[i +1] with A[r]
8.      return i + 1
```

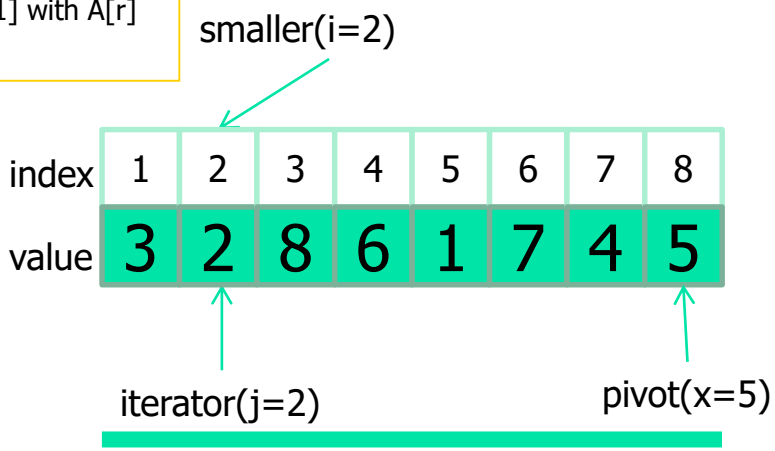
p = 1
r = 8



Quicksort

```
PARTITION(A, p, r)
1.    x = A[r]
2.    i = p – 1
3.    for j = p to r – 1
4.       if A[j] ≤ x
5.          i = i + 1
6.          exchange A[i] with A[j]
7.    exchange A[i +1] with A[r]
8.    return i + 1
```

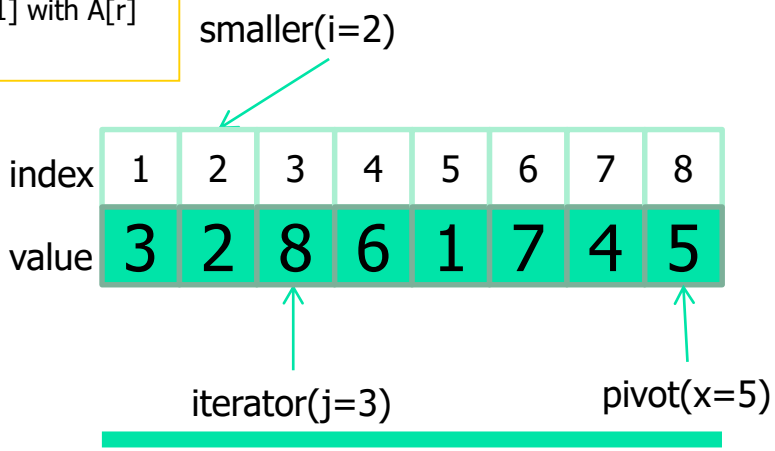
p = 1
r = 8



Quicksort

```
PARTITION(A, p, r)
1.    x = A[r]
2.    i = p – 1
3.    for j = p to r – 1
4.       if A[j] ≤ x
5.          i = i + 1
6.          exchange A[i] with A[j]
7.    exchange A[i +1] with A[r]
8.    return i + 1
```

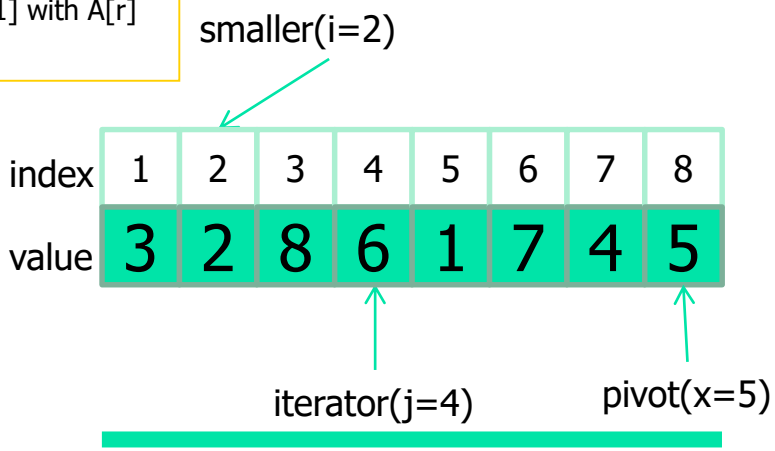
p = 1
r = 8



Quicksort

```
PARTITION(A, p, r)
1.    x = A[r]
2.    i = p – 1
3.    for j = p to r – 1
4.       if A[j] ≤ x
5.          i = i + 1
6.          exchange A[i] with A[j]
7.    exchange A[i +1] with A[r]
8.    return i + 1
```

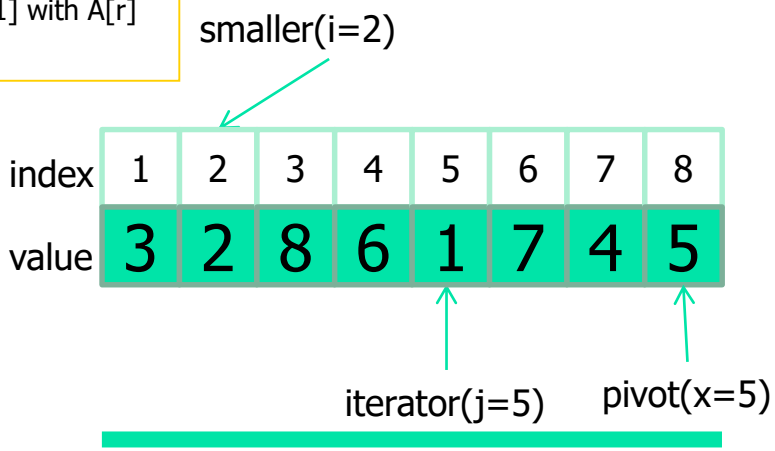
p = 1
r = 8



Quicksort

```
PARTITION(A, p, r)
1.    x = A[r]
2.    i = p – 1
3.    for j = p to r – 1
4.       if A[j] ≤ x
5.          i = i + 1
6.          exchange A[i] with A[j]
7.    exchange A[i +1] with A[r]
8.    return i + 1
```

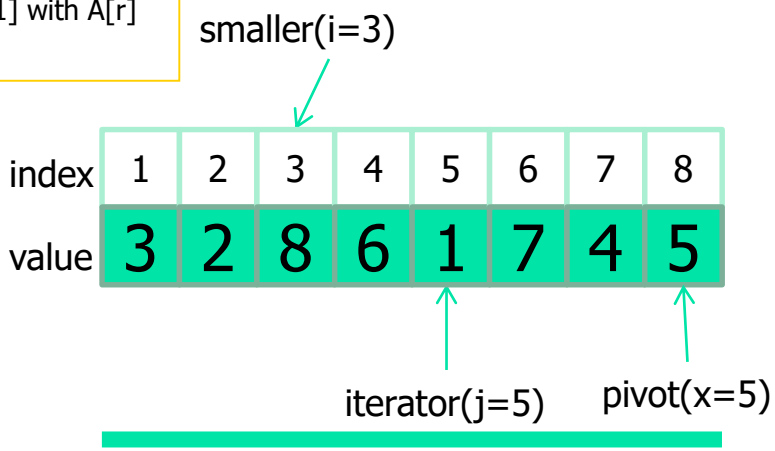
p = 1
r = 8



Quicksort

```
PARTITION(A, p, r)
1.      x = A[r]
2.      i = p – 1
3.      for j = p to r – 1
4.          if A[j] ≤ x
5.              i = i + 1
6.              exchange A[i] with A[j]
7.      exchange A[i +1] with A[r]
8.      return i + 1
```

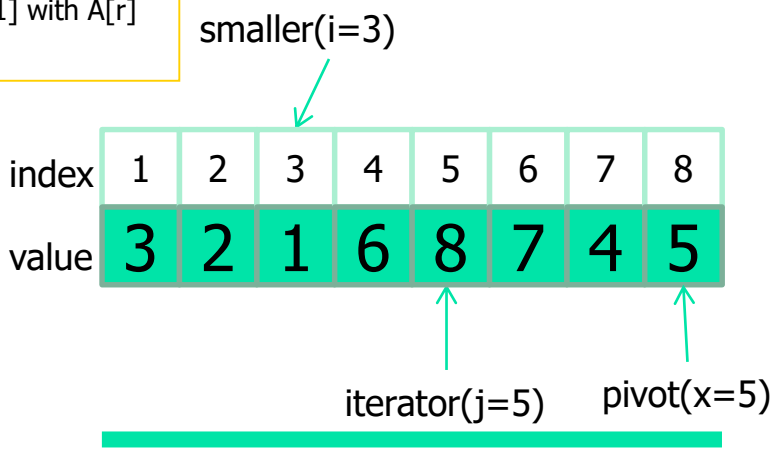
p = 1
r = 8



Quicksort

```
PARTITION(A, p, r)
1.    x = A[r]
2.    i = p – 1
3.    for j = p to r – 1
4.       if A[j] ≤ x
5.          i = i + 1
6.          exchange A[i] with A[j]
7.    exchange A[i +1] with A[r]
8.    return i + 1
```

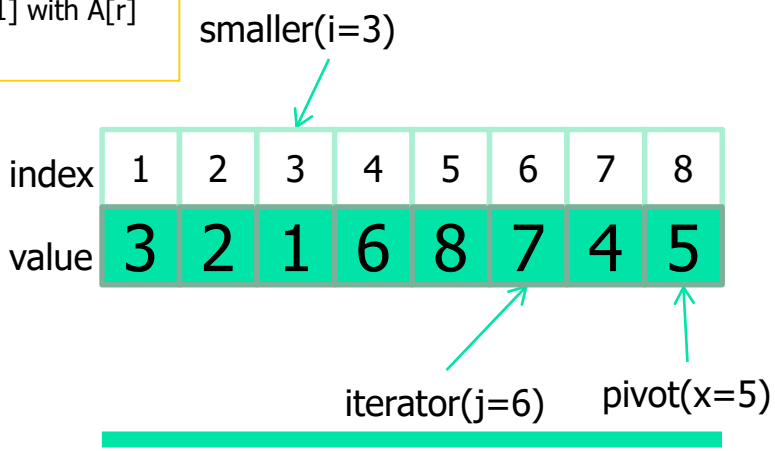
p = 1
r = 8



Quicksort

```
PARTITION(A, p, r)
1.    x = A[r]
2.    i = p – 1
3.    for j = p to r – 1
4.       if A[j] ≤ x
5.          i = i + 1
6.          exchange A[i] with A[j]
7.    exchange A[i +1] with A[r]
8.    return i + 1
```

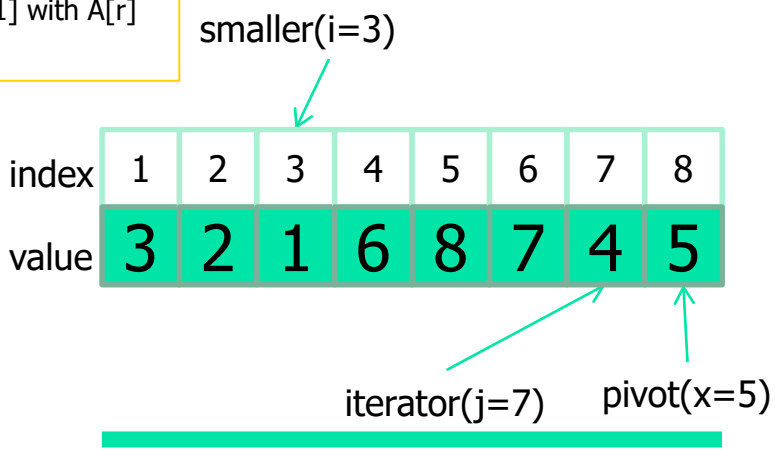
p = 1
r = 8



Quicksort

```
PARTITION(A, p, r)
1.    x = A[r]
2.    i = p – 1
3.    for j = p to r – 1
4.       if A[j] ≤ x
5.          i = i + 1
6.          exchange A[i] with A[j]
7.    exchange A[i +1] with A[r]
8.    return i + 1
```

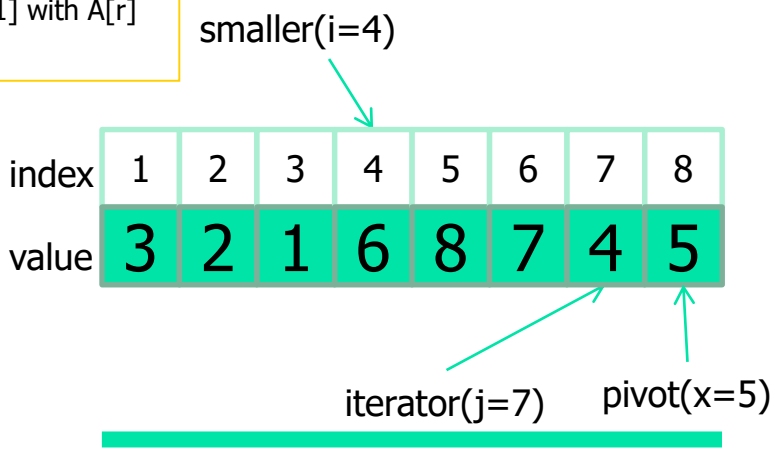
p = 1
r = 8



Quicksort

```
PARTITION(A, p, r)
1.    x = A[r]
2.    i = p – 1
3.    for j = p to r – 1
4.       if A[j] ≤ x
5.          i = i + 1
6.          exchange A[i] with A[j]
7.    exchange A[i +1] with A[r]
8.    return i + 1
```

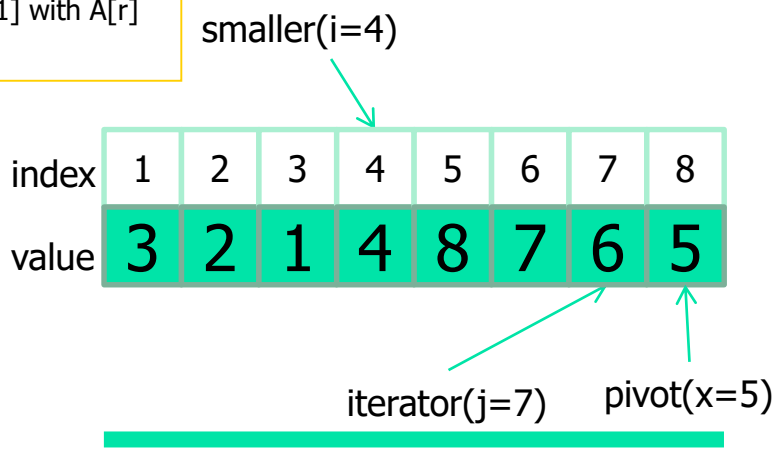
p = 1
r = 8



Quicksort

```
PARTITION(A, p, r)
1.    x = A[r]
2.    i = p – 1
3.    for j = p to r – 1
4.       if A[j] ≤ x
5.          i = i + 1
6.          exchange A[i] with A[j]
7.    exchange A[i +1] with A[r]
8.    return i + 1
```

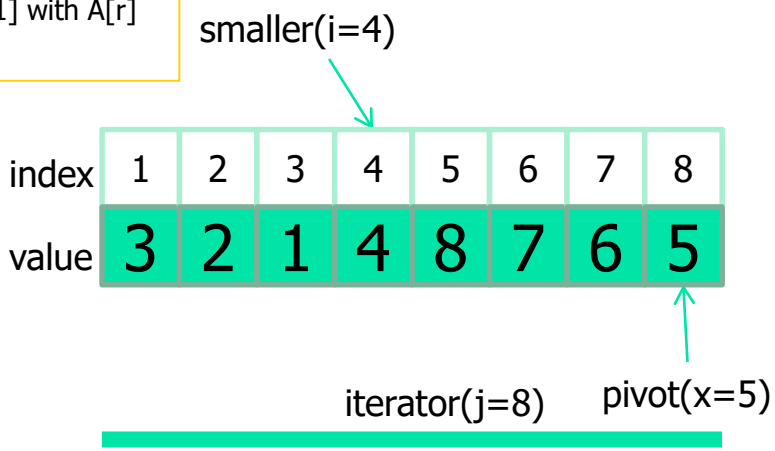
p = 1
r = 8



Quicksort

```
PARTITION(A, p, r)
1.    x = A[r]
2.    i = p – 1
3.    for j = p to r – 1
4.       if A[j] ≤ x
5.          i = i + 1
6.          exchange A[i] with A[j]
7.    exchange A[i +1] with A[r]
8.    return i + 1
```

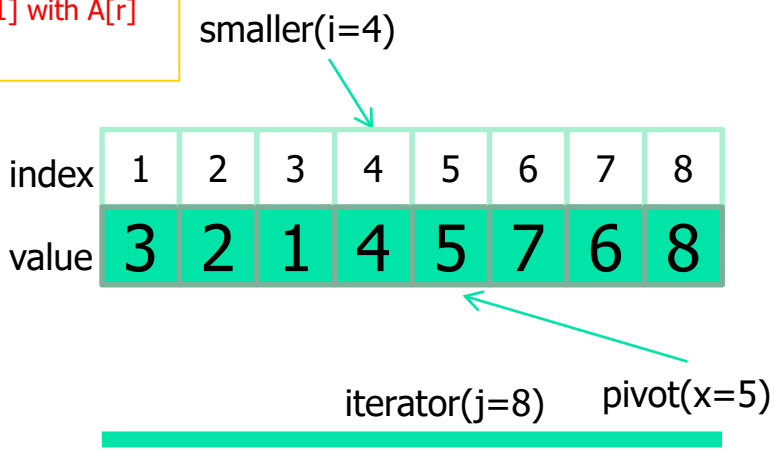
p = 1
r = 8



Quicksort

```
PARTITION(A, p, r)
1.      x = A[r]
2.      i = p – 1
3.      for j = p to r – 1
4.          if A[j] ≤ x
5.              i = i + 1
6.              exchange A[i] with A[j]
7.      exchange A[i +1] with A[r]
8.      return i + 1
```

p = 1
r = 8

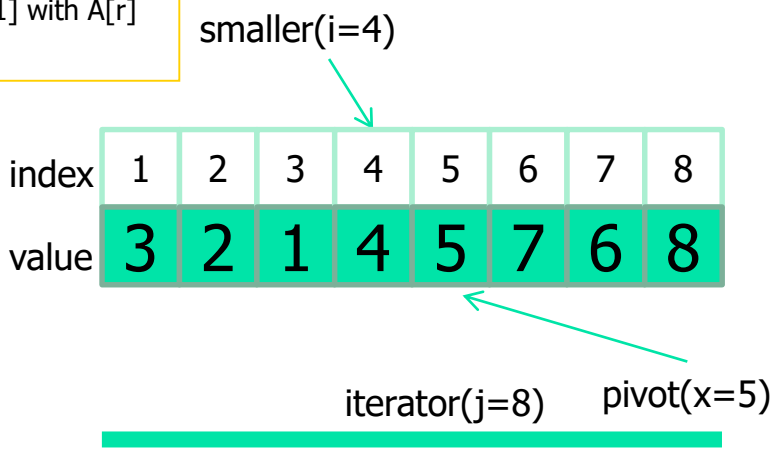


Quicksort

```
PARTITION(A, p, r)
1.    x = A[r]
2.    i = p – 1
3.    for j = p to r – 1
4.       if A[j] ≤ x
5.          i = i + 1
6.          exchange A[i] with A[j]
7.    exchange A[i +1] with A[r]
8.    return i + 1
```

p = 1
r = 8

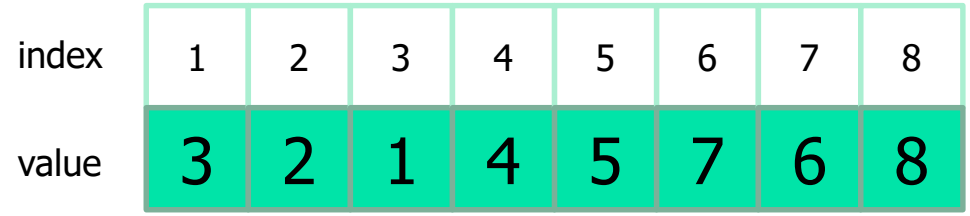
return 5



Quicksort

```
QUICKSORT(A, p, r)
1.   if p < r
2.      q = PARTITION(A, p, r)
3.      QUICKSORT(A, p, q-1)
4.      QUICKSORT(A, q + 1, r)
```

return



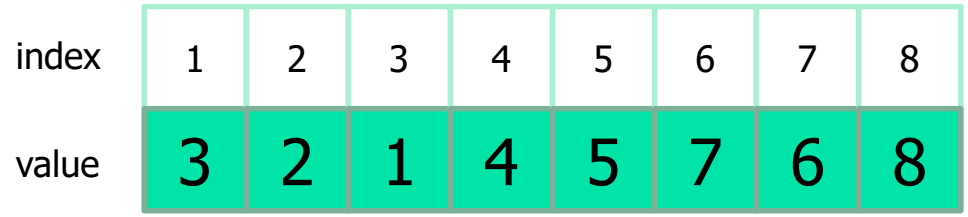
p = 1

r = 8

q = 5

Quicksort

```
QUICKSORT(A, p, r)
1.    if p < r
2.       q = PARTITION(A, p, r)
3.       QUICKSORT(A, p, q-1)
4.       QUICKSORT(A, q + 1, r)
```



p = 1

r = 8

q = 5

Quicksort

```
QUICKSORT(A, p, r)
1.    if p < r
2.       q = PARTITION(A, p, r)
3.       QUICKSORT(A, p, q-1)
4.       QUICKSORT(A, q + 1, r)
```

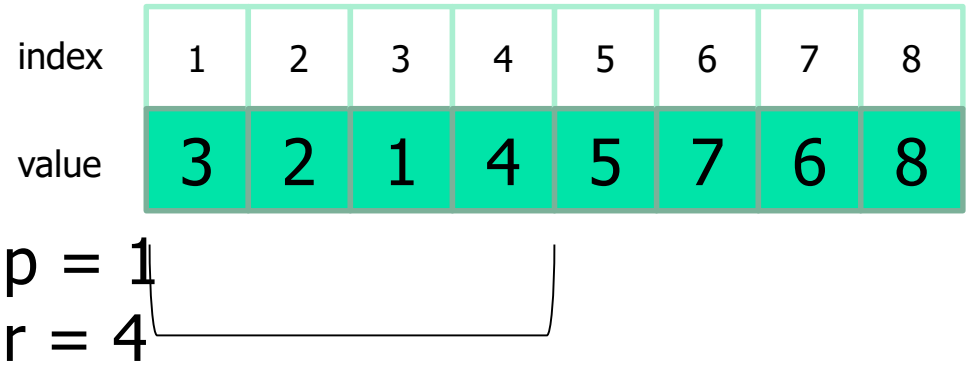
index	1	2	3	4	5	6	7	8
value	3	2	1	4	5	7	6	8

p = 1

r = 4

Quicksort

```
QUICKSORT(A, p, r)
1.    if p < r
2.       q = PARTITION(A, p, r)
3.       QUICKSORT(A, p, q-1)
4.       QUICKSORT(A, q + 1, r)
```



Quicksort

```
PARTITION(A, p, r)
1.      x = A[r]
2.      i = p – 1
3.      for j = p to r – 1
4.          if A[j] ≤ x
5.              i = i + 1
6.              exchange A[i] with A[j]
7.      exchange A[i +1] with A[r]
8.      return i + 1
```

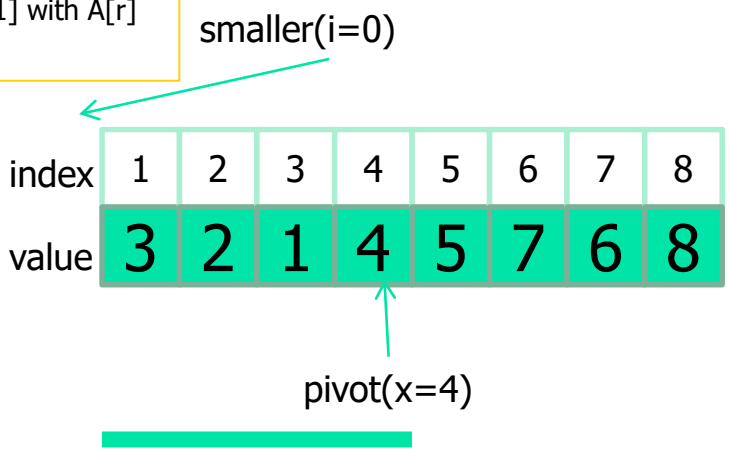
p = 1
r = 4

index	1	2	3	4	5	6	7	8
value	3	2	1	4	5	7	6	8

Quicksort

```
PARTITION(A, p, r)
1.    x = A[r]
2.    i = p – 1
3.    for j = p to r – 1
4.       if A[j] ≤ x
5.          i = i + 1
6.          exchange A[i] with A[j]
7.    exchange A[i +1] with A[r]
8.    return i + 1
```

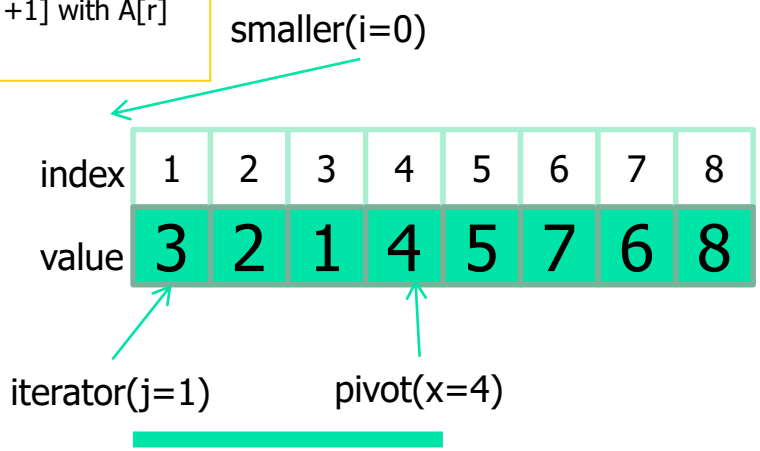
p = 1
r = 4



Quicksort

```
PARTITION(A, p, r)
1.      x = A[r]
2.      i = p – 1
3.      for j = p to r – 1
4.          if A[j] ≤ x
5.              i = i + 1
6.              exchange A[i] with A[j]
7.      exchange A[i +1] with A[r]
8.      return i + 1
```

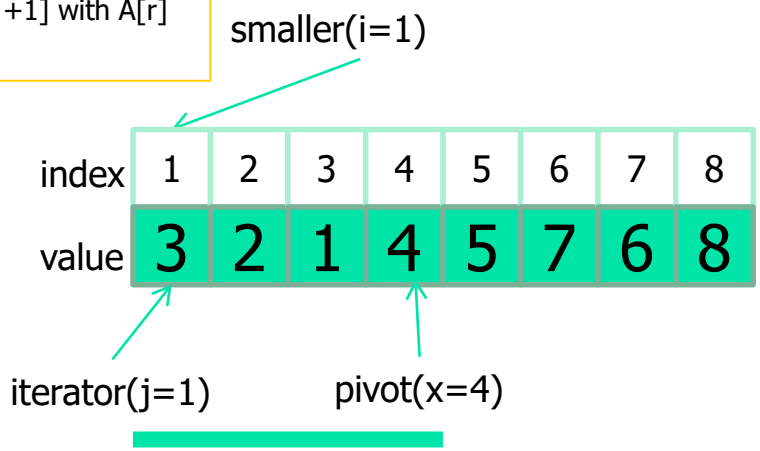
p = 1
r = 4



Quicksort

```
PARTITION(A, p, r)
1.      x = A[r]
2.      i = p – 1
3.      for j = p to r – 1
4.          if A[j] ≤ x
5.              i = i + 1
6.              exchange A[i] with A[j]
7.      exchange A[i +1] with A[r]
8.      return i + 1
```

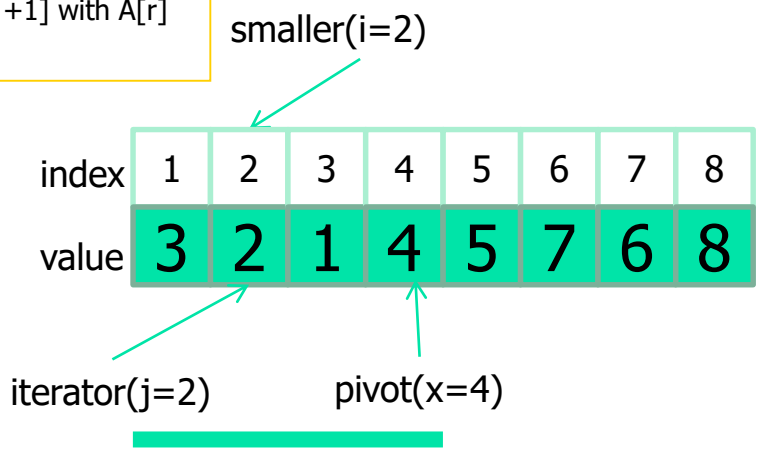
p = 1
r = 4



Quicksort

```
PARTITION(A, p, r)
1.    x = A[r]
2.    i = p – 1
3.    for j = p to r – 1
4.       if A[j] ≤ x
5.          i = i + 1
6.          exchange A[i] with A[j]
7.    exchange A[i +1] with A[r]
8.    return i + 1
```

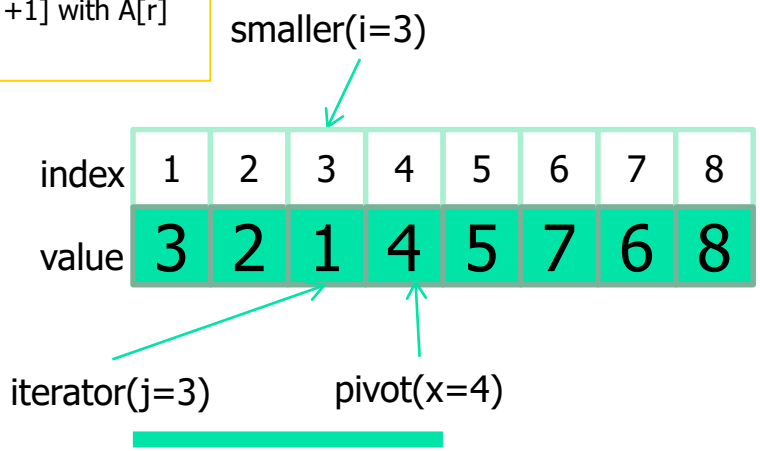
p = 1
r = 4



Quicksort

```
PARTITION(A, p, r)
1.    x = A[r]
2.    i = p – 1
3.    for j = p to r – 1
4.       if A[j] ≤ x
5.          i = i + 1
6.          exchange A[i] with A[j]
7.    exchange A[i +1] with A[r]
8.    return i + 1
```

p = 1
r = 4

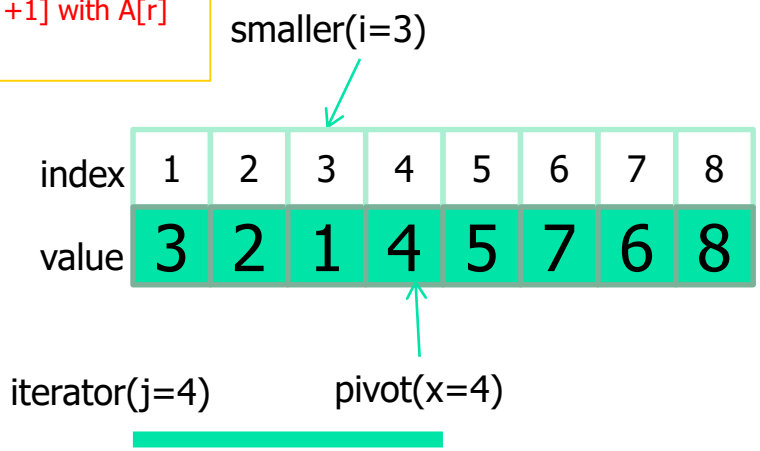


Quicksort

```
PARTITION(A, p, r)
1.      x = A[r]
2.      i = p – 1
3.      for j = p to r – 1
4.          if A[j] ≤ x
5.              i = i + 1
6.              exchange A[i] with A[j]
7.      exchange A[i +1] with A[r]
8.      return i + 1
```

p = 1
r = 4

return 4



Quicksort

```
QUICKSORT(A, p, r)
1.    if p < r
2.       q = PARTITION(A, p, r)
3.       QUICKSORT(A, p, q-1)
4.       QUICKSORT(A, q + 1, r)
```

return

index	1	2	3	4	5	6	7	8
value	3	2	1	4	5	7	6	8

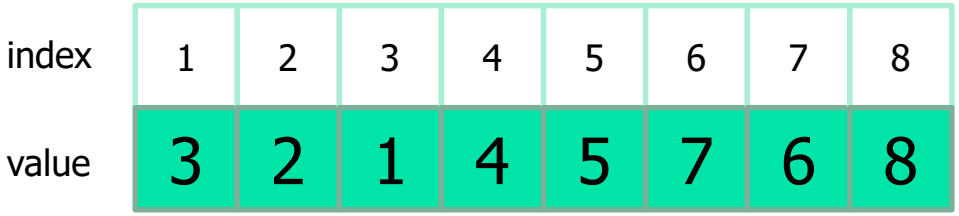
p = 1

r = 4

q = 4

Quicksort

```
QUICKSORT(A, p, r)
1.    if p < r
2.       q = PARTITION(A, p, r)
3.       QUICKSORT(A, p, q-1)
4.       QUICKSORT(A, q + 1, r)
```



p = 1

r = 4

q = 4

Quicksort

```
QUICKSORT(A, p, r)
1.    if p < r
2.       q = PARTITION(A, p, r)
3.       QUICKSORT(A, p, q-1)
4.       QUICKSORT(A, q + 1, r)
```

index	1	2	3	4	5	6	7	8
value	3	2	1	4	5	7	6	8

p = 1

r = 3

Quicksort

```
QUICKSORT(A, p, r)
1.    if p < r
2.       q = PARTITION(A, p, r)
3.       QUICKSORT(A, p, q-1)
4.       QUICKSORT(A, q + 1, r)
```

index	1	2	3	4	5	6	7	8
value	3	2	1	4	5	7	6	8

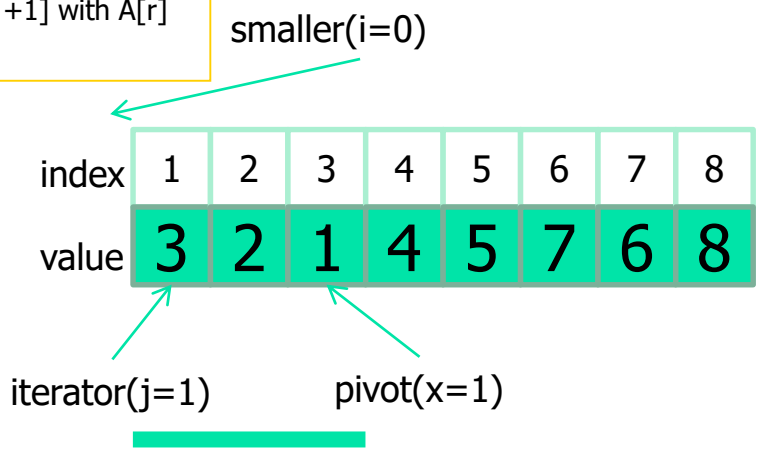
p = 1

r = 3

Quicksort

```
PARTITION(A, p, r)
1.    x = A[r]
2.    i = p – 1
3.    for j = p to r – 1
4.       if A[j] ≤ x
5.          i = i + 1
6.          exchange A[i] with A[j]
7.    exchange A[i +1] with A[r]
8.    return i + 1
```

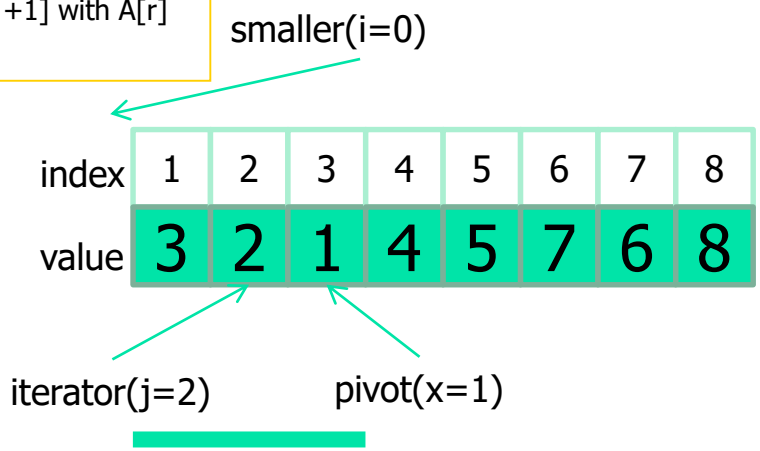
p = 1
r = 3



Quicksort

```
PARTITION(A, p, r)
1.    x = A[r]
2.    i = p – 1
3.    for j = p to r – 1
4.       if A[j] ≤ x
5.          i = i + 1
6.          exchange A[i] with A[j]
7.    exchange A[i +1] with A[r]
8.    return i + 1
```

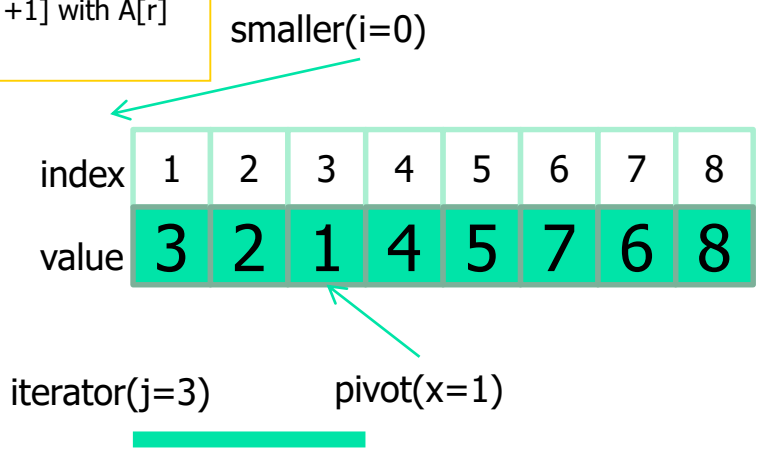
p = 1
r = 3



Quicksort

```
PARTITION(A, p, r)
1.      x = A[r]
2.      i = p – 1
3.      for j = p to r – 1
4.          if A[j] ≤ x
5.              i = i + 1
6.              exchange A[i] with A[j]
7.      exchange A[i +1] with A[r]
8.      return i + 1
```

p = 1
r = 3



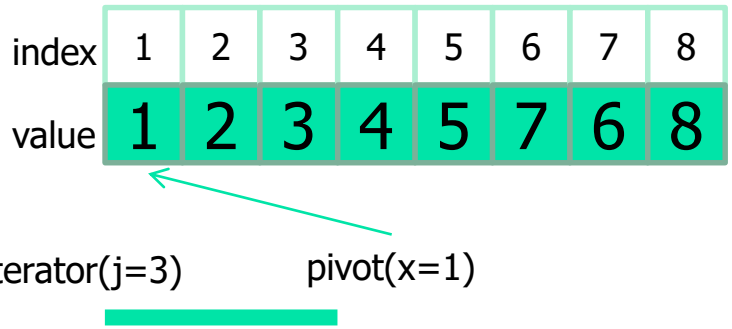
Quicksort

```
PARTITION(A, p, r)
1.      x = A[r]
2.      i = p – 1
3.      for j = p to r – 1
4.          if A[j] ≤ x
5.              i = i + 1
6.              exchange A[i] with A[j]
7.      exchange A[i +1] with A[r]
8.      return i + 1
```

p = 1
r = 3

smaller(i=0)

return 1



Quicksort

```
QUICKSORT(A, p, r)
1.   if p < r
2.      q = PARTITION(A, p, r)
3.      QUICKSORT(A, p, q-1)
4.      QUICKSORT(A, q + 1, r)
```

return

index	1	2	3	4	5	6	7	8
value	1	2	3	4	5	7	6	8

p = 1
r = 3
q = 1

Quicksort

```
QUICKSORT(A, p, r)
1.    if p < r
2.       q = PARTITION(A, p, r)
3.       QUICKSORT(A, p, q-1)
4.       QUICKSORT(A, q + 1, r)
```

index	1	2	3	4	5	6	7	8
value	1	2	3	4	5	7	6	8

p = 1
r = 3
q = 1

Quicksort

```
QUICKSORT(A, p, r)
1.    if p < r
2.       q = PARTITION(A, p, r)
3.       QUICKSORT(A, p, q-1)
4.       QUICKSORT(A, q + 1, r)
```

index	1	2	3	4	5	6	7	8
value	1	2	3	4	5	7	6	8

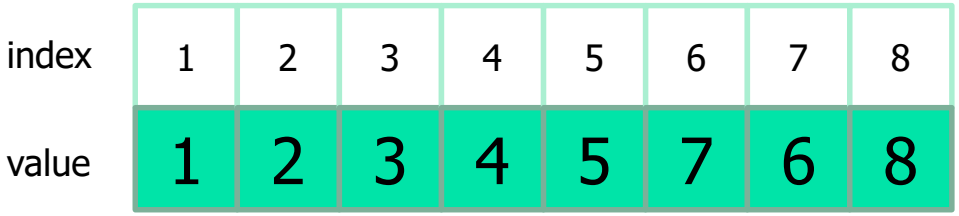
p = 1

r = 0

Quicksort

```
QUICKSORT(A, p, r)
1.    if p < r
2.       q = PARTITION(A, p, r)
3.       QUICKSORT(A, p, q-1)
4.       QUICKSORT(A, q + 1, r)
```

return



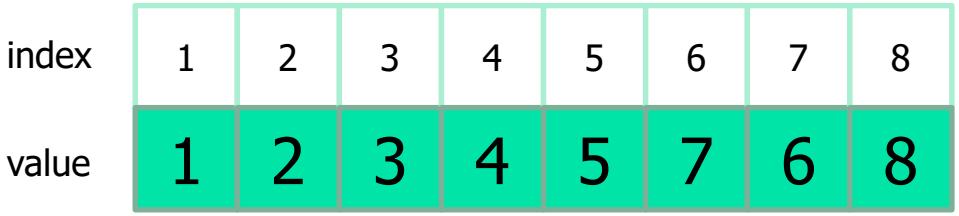
p = 1

r = 3

q = 1

Quicksort

```
QUICKSORT(A, p, r)
1.    if p < r
2.       q = PARTITION(A, p, r)
3.       QUICKSORT(A, p, q-1)
4.       QUICKSORT(A, q + 1, r)
```



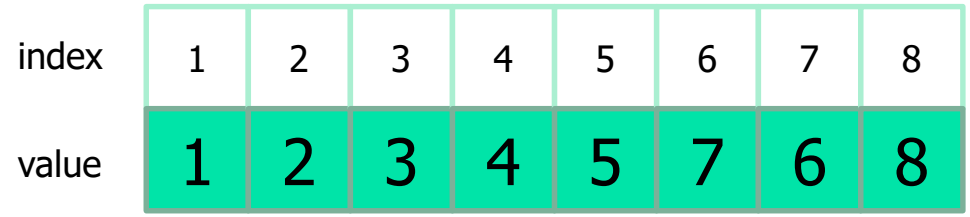
p = 1

r = 3

q = 1

Quicksort

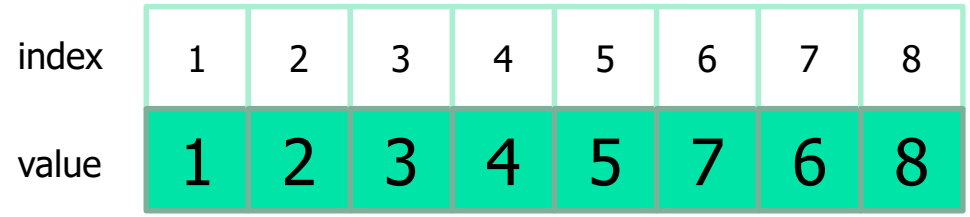
```
QUICKSORT(A, p, r)
1.    if p < r
2.       q = PARTITION(A, p, r)
3.       QUICKSORT(A, p, q-1)
4.       QUICKSORT(A, q + 1, r)
```



p = 2
r = 3

Quicksort

```
QUICKSORT(A, p, r)
1.   if p < r
2.      q = PARTITION(A, p, r)
3.      QUICKSORT(A, p, q-1)
4.      QUICKSORT(A, q + 1, r)
```

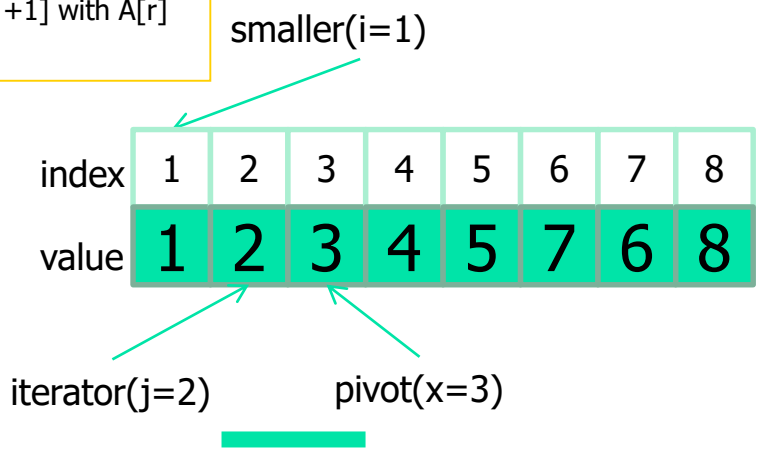


p = 2
r = 3

Quicksort

```
PARTITION(A, p, r)
1.    x = A[r]
2.    i = p – 1
3.    for j = p to r – 1
4.       if A[j] ≤ x
5.          i = i + 1
6.          exchange A[i] with A[j]
7.    exchange A[i +1] with A[r]
8.    return i + 1
```

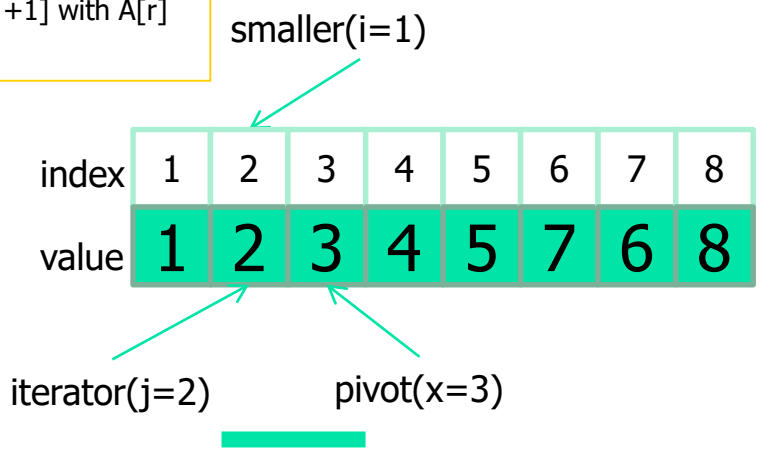
p = 2
r = 3



Quicksort

```
PARTITION(A, p, r)
1.    x = A[r]
2.    i = p – 1
3.    for j = p to r – 1
4.       if A[j] ≤ x
5.          i = i + 1
6.          exchange A[i] with A[j]
7.    exchange A[i +1] with A[r]
8.    return i + 1
```

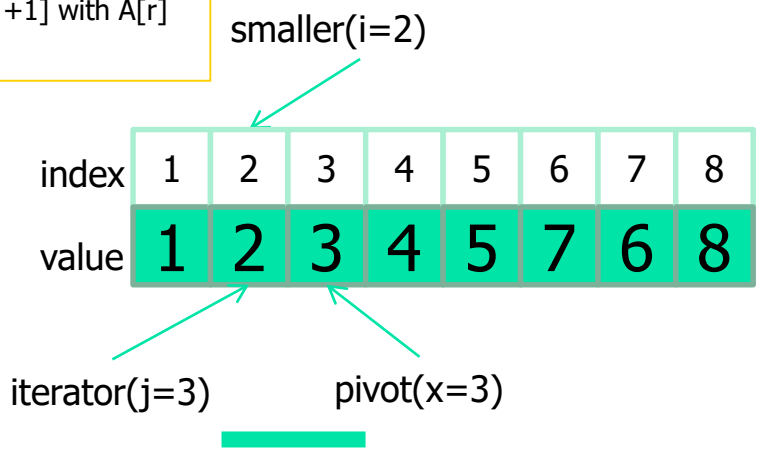
p = 2
r = 3



Quicksort

```
PARTITION(A, p, r)
1.      x = A[r]
2.      i = p – 1
3.      for j = p to r – 1
4.          if A[j] ≤ x
5.              i = i + 1
6.              exchange A[i] with A[j]
7.      exchange A[i +1] with A[r]
8.      return i + 1
```

p = 2
r = 3

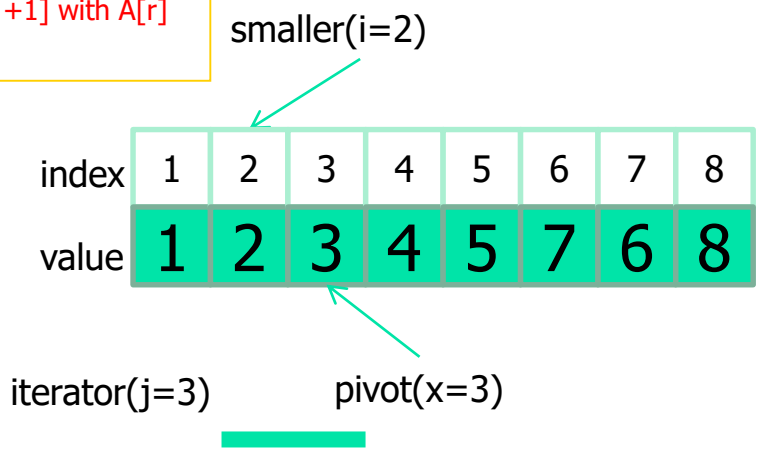


Quicksort

```
PARTITION(A, p, r)
1.      x = A[r]
2.      i = p – 1
3.      for j = p to r – 1
4.          if A[j] ≤ x
5.              i = i + 1
6.              exchange A[i] with A[j]
7.      exchange A[i +1] with A[r]
8.      return i + 1
```

p = 2
r = 3

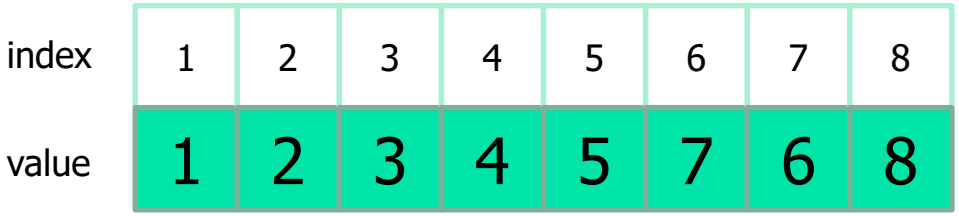
return 3



Quicksort

```
QUICKSORT(A, p, r)
1.   if p < r
2.      q = PARTITION(A, p, r)
3.      QUICKSORT(A, p, q-1)
4.      QUICKSORT(A, q + 1, r)
```

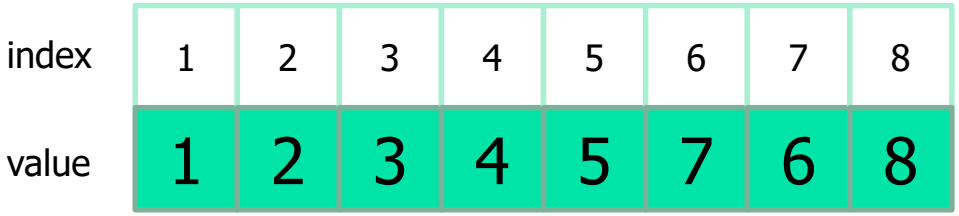
return



p = 2
r = 3
q = 3

Quicksort

```
QUICKSORT(A, p, r)
1.    if p < r
2.       q = PARTITION(A, p, r)
3.       QUICKSORT(A, p, q-1)
4.       QUICKSORT(A, q + 1, r)
```



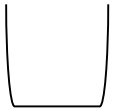
p = 2
r = 3
q = 3

Quicksort

```
QUICKSORT(A, p, r)
1.    if p < r
2.       q = PARTITION(A, p, r)
3.       QUICKSORT(A, p, q-1)
4.       QUICKSORT(A, q + 1, r)
```

index	1	2	3	4	5	6	7	8
value	1	2	3	4	5	7	6	8

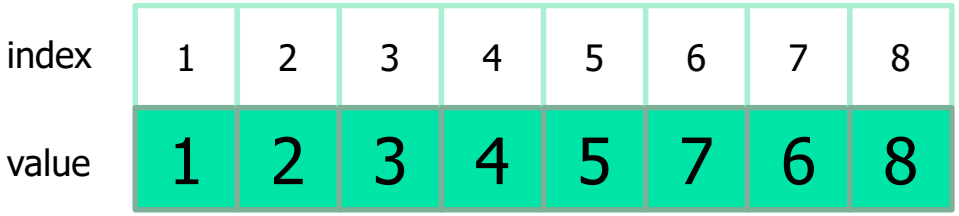
p = 2
r = 2



Quicksort

```
QUICKSORT(A, p, r)
1.   if p < r
2.      q = PARTITION(A, p, r)
3.      QUICKSORT(A, p, q-1)
4.      QUICKSORT(A, q + 1, r)
```

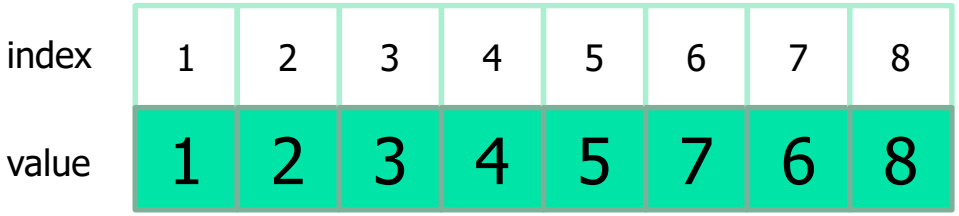
return



p = 2
r = 3
q = 3

Quicksort

```
QUICKSORT(A, p, r)
1.    if p < r
2.       q = PARTITION(A, p, r)
3.       QUICKSORT(A, p, q-1)
4.       QUICKSORT(A, q + 1, r)
```



p = 2
r = 3
q = 3

Quicksort

```
QUICKSORT(A, p, r)
1.    if p < r
2.       q = PARTITION(A, p, r)
3.       QUICKSORT(A, p, q-1)
4.       QUICKSORT(A, q + 1, r)
```

index	1	2	3	4	5	6	7	8
value	1	2	3	4	5	7	6	8

p = 4

r = 3

Quicksort

```
QUICKSORT(A, p, r)
1.    if p < r
2.       q = PARTITION(A, p, r)
3.       QUICKSORT(A, p, q-1)
4.       QUICKSORT(A, q + 1, r)
```

return

index	1	2	3	4	5	6	7	8
value	1	2	3	4	5	7	6	8

p = 1

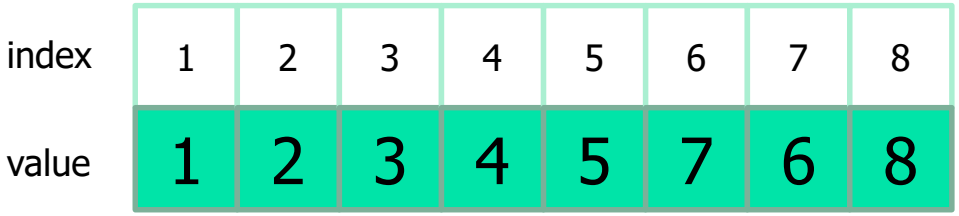
r = 3

q = 1

Quicksort

```
QUICKSORT(A, p, r)
1.   if p < r
2.      q = PARTITION(A, p, r)
3.      QUICKSORT(A, p, q-1)
4.      QUICKSORT(A, q + 1, r)
```

return



p = 1

r = 4

q = 4

Quicksort

```
QUICKSORT(A, p, r)
1.    if p < r
2.       q = PARTITION(A, p, r)
3.       QUICKSORT(A, p, q-1)
4.       QUICKSORT(A, q + 1, r)
```

index	1	2	3	4	5	6	7	8
value	1	2	3	4	5	7	6	8

p = 1

r = 4

q = 4

Quicksort

```
QUICKSORT(A, p, r)
1.    if p < r
2.       q = PARTITION(A, p, r)
3.       QUICKSORT(A, p, q-1)
4.       QUICKSORT(A, q + 1, r)
```

index	1	2	3	4	5	6	7	8
value	1	2	3	4	5	7	6	8

p = 5

r = 4

Quicksort

```
QUICKSORT(A, p, r)
1.    if p < r
2.       q = PARTITION(A, p, r)
3.       QUICKSORT(A, p, q-1)
4.       QUICKSORT(A, q + 1, r)
```

return

index	1	2	3	4	5	6	7	8
value	1	2	3	4	5	7	6	8

p = 1

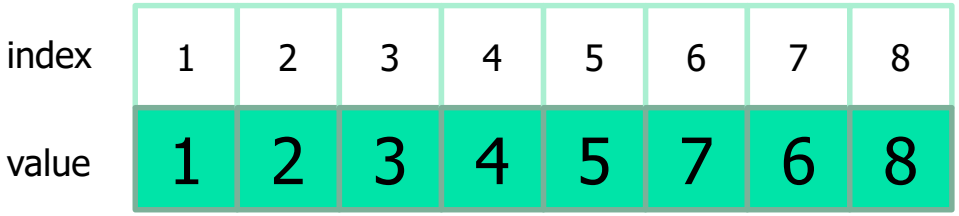
r = 4

q = 4

Quicksort

```
QUICKSORT(A, p, r)
1.    if p < r
2.       q = PARTITION(A, p, r)
3.       QUICKSORT(A, p, q-1)
4.       QUICKSORT(A, q + 1, r)
```

return



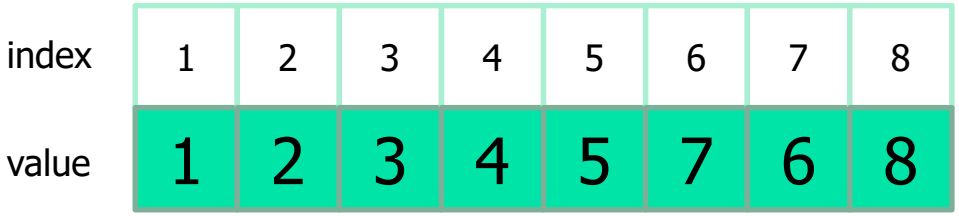
p = 1

r = 8

q = 5

Quicksort

```
QUICKSORT(A, p, r)
1.    if p < r
2.       q = PARTITION(A, p, r)
3.       QUICKSORT(A, p, q-1)
4.       QUICKSORT(A, q + 1, r)
```



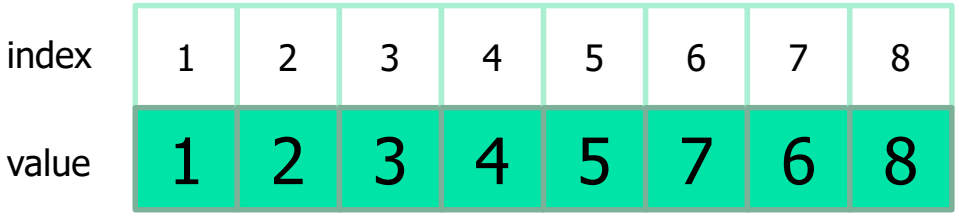
p = 1

r = 8

q = 5

Quicksort

```
QUICKSORT(A, p, r)
1.    if p < r
2.       q = PARTITION(A, p, r)
3.       QUICKSORT(A, p, q-1)
4.       QUICKSORT(A, q + 1, r)
```

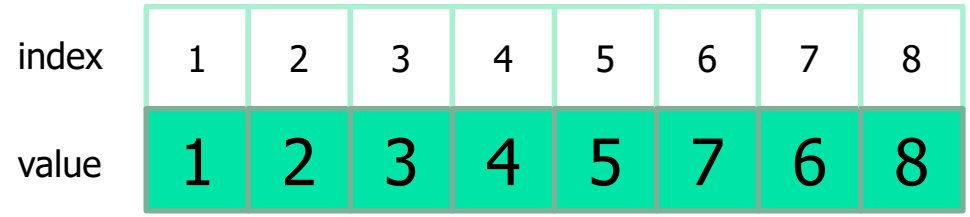


p = 6

r = 8

Quicksort

```
QUICKSORT(A, p, r)
1.    if p < r
2.       q = PARTITION(A, p, r)
3.       QUICKSORT(A, p, q-1)
4.       QUICKSORT(A, q + 1, r)
```

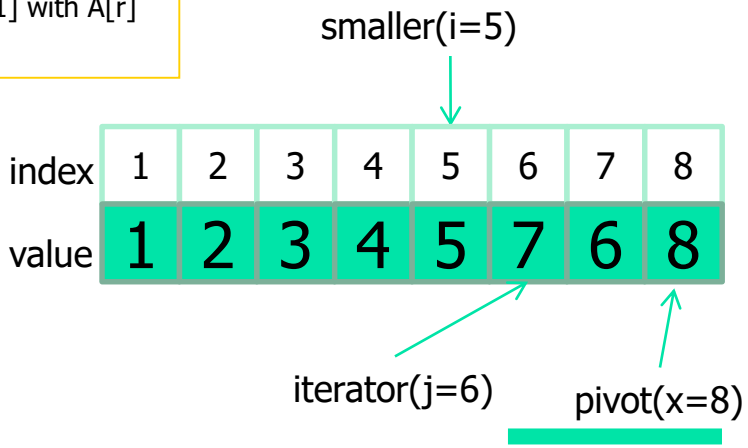


p = 6
r = 8

Quicksort

```
PARTITION(A, p, r)
1.    x = A[r]
2.    i = p – 1
3.    for j = p to r – 1
4.       if A[j] ≤ x
5.          i = i + 1
6.          exchange A[i] with A[j]
7.    exchange A[i +1] with A[r]
8.    return i + 1
```

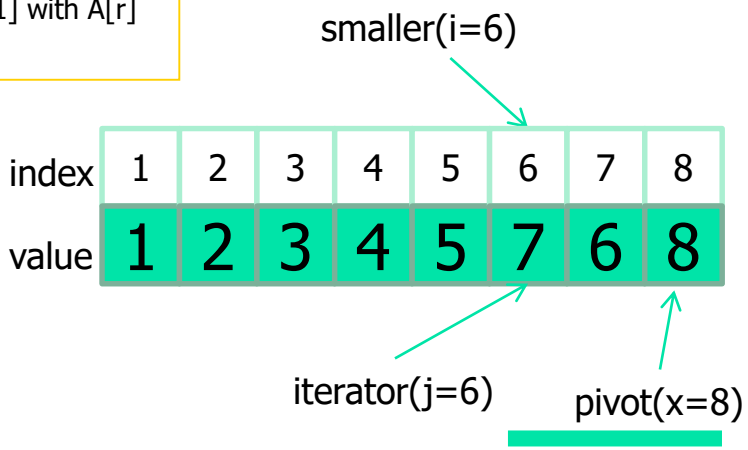
p = 6
r = 8



Quicksort

```
PARTITION(A, p, r)
1.    x = A[r]
2.    i = p – 1
3.    for j = p to r – 1
4.       if A[j] ≤ x
5.          i = i + 1
6.          exchange A[i] with A[j]
7.    exchange A[i +1] with A[r]
8.    return i + 1
```

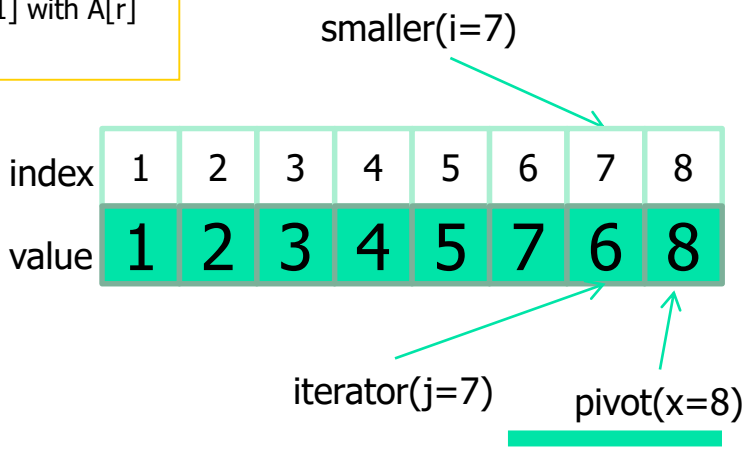
p = 6
r = 8



Quicksort

```
PARTITION(A, p, r)
1.    x = A[r]
2.    i = p – 1
3.    for j = p to r – 1
4.       if A[j] ≤ x
5.          i = i + 1
6.          exchange A[i] with A[j]
7.    exchange A[i +1] with A[r]
8.    return i + 1
```

p = 6
r = 8

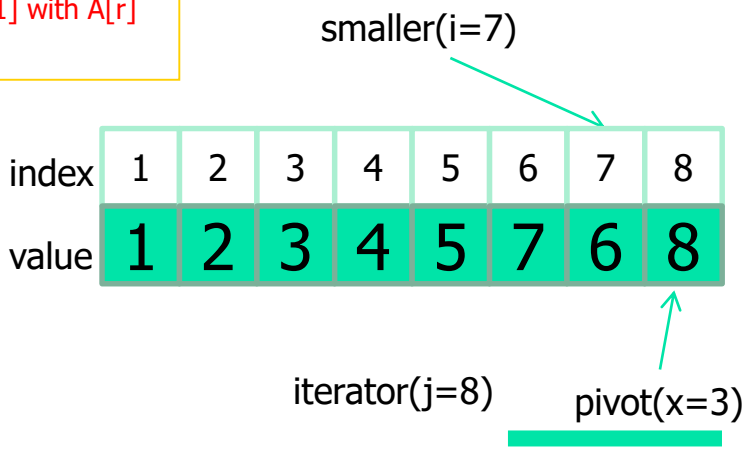


Quicksort

```
PARTITION(A, p, r)
1.      x = A[r]
2.      i = p – 1
3.      for j = p to r – 1
4.          if A[j] ≤ x
5.              i = i + 1
6.              exchange A[i] with A[j]
7.      exchange A[i +1] with A[r]
8.      return i + 1
```

p = 6
r = 8

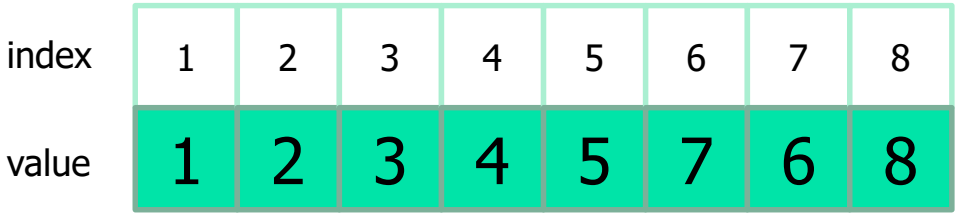
return 8



Quicksort

```
QUICKSORT(A, p, r)
1.    if p < r
2.       q = PARTITION(A, p, r)
3.       QUICKSORT(A, p, q-1)
4.       QUICKSORT(A, q + 1, r)
```

return



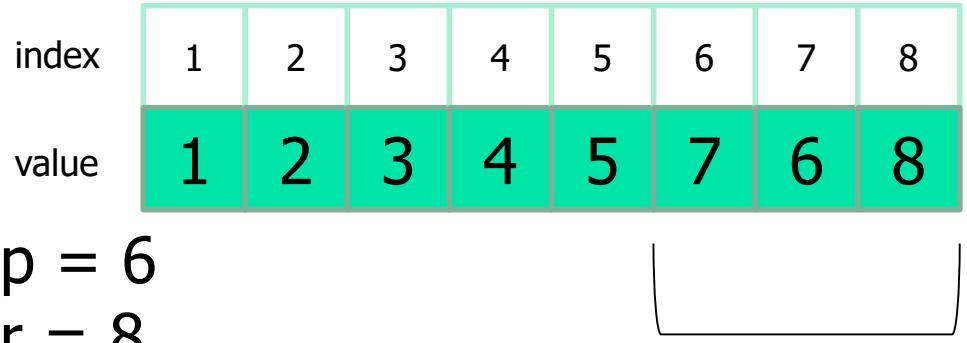
p = 6

r = 8

q = 8

Quicksort

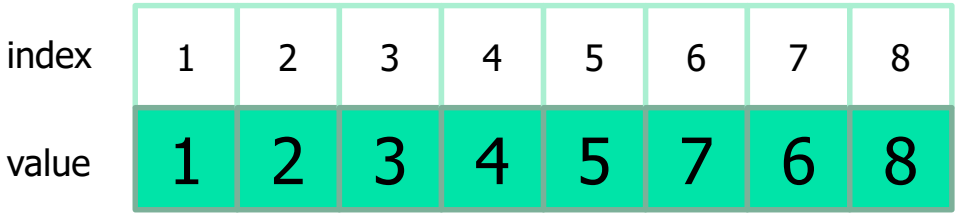
```
QUICKSORT(A, p, r)
1.    if p < r
2.       q = PARTITION(A, p, r)
3.       QUICKSORT(A, p, q-1)
4.       QUICKSORT(A, q + 1, r)
```



p = 6
r = 8
q = 8

Quicksort

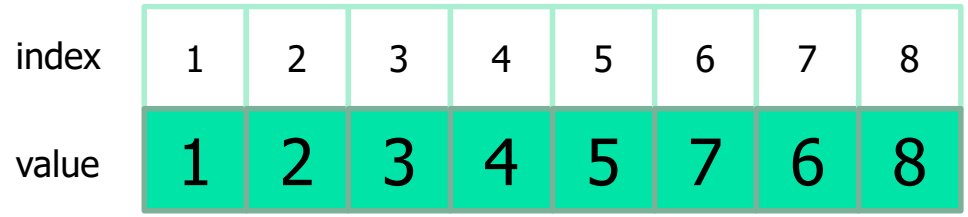
```
QUICKSORT(A, p, r)
1.    if p < r
2.       q = PARTITION(A, p, r)
3.       QUICKSORT(A, p, q-1)
4.       QUICKSORT(A, q + 1, r)
```



p = 6
r = 7

Quicksort

```
QUICKSORT(A, p, r)
1.    if p < r
2.       q = PARTITION(A, p, r)
3.       QUICKSORT(A, p, q-1)
4.       QUICKSORT(A, q + 1, r)
```

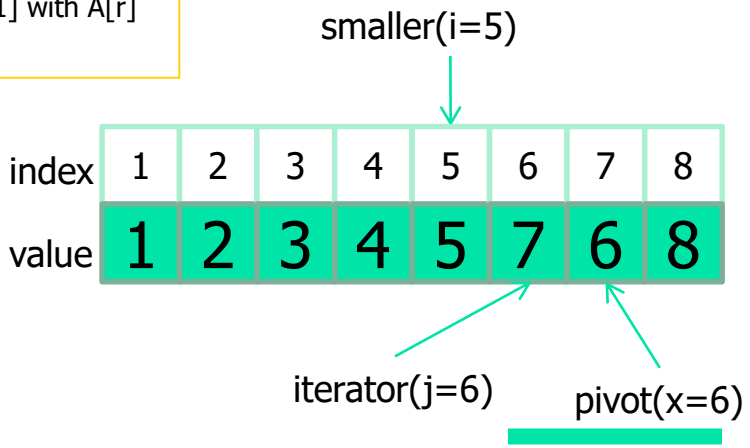


p = 6
r = 7

Quicksort

```
PARTITION(A, p, r)
1.      x = A[r]
2.      i = p – 1
3.      for j = p to r – 1
4.          if A[j] ≤ x
5.              i = i + 1
6.              exchange A[i] with A[j]
7.      exchange A[i +1] with A[r]
8.      return i + 1
```

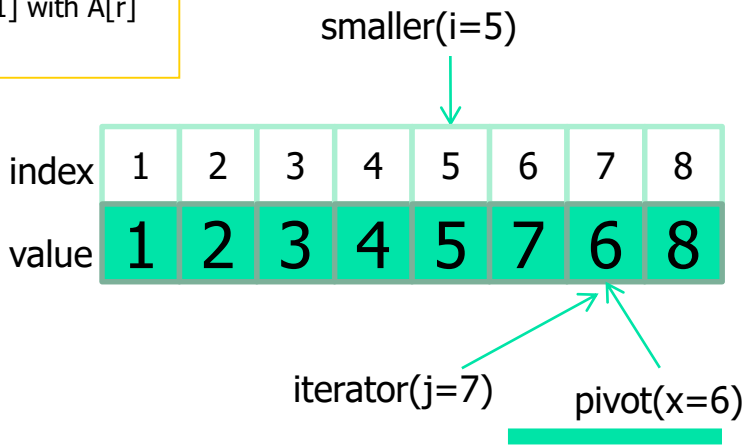
p = 6
r = 7



Quicksort

```
PARTITION(A, p, r)
1.      x = A[r]
2.      i = p – 1
3.      for j = p to r – 1
4.          if A[j] ≤ x
5.              i = i + 1
6.              exchange A[i] with A[j]
7.      exchange A[i +1] with A[r]
8.      return i + 1
```

p = 6
r = 7

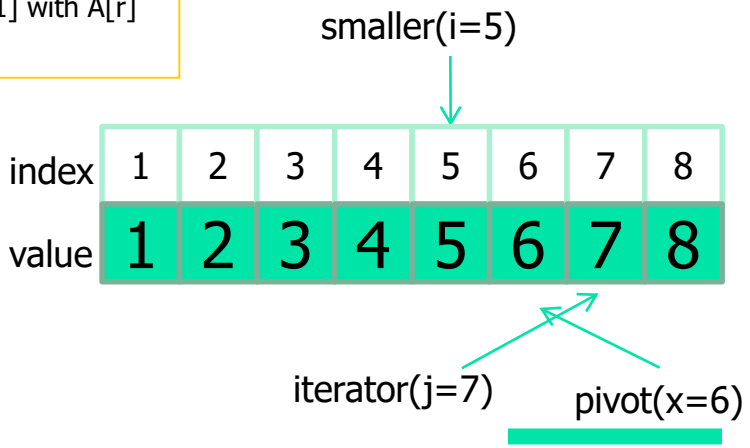


Quicksort

```
PARTITION(A, p, r)
1.      x = A[r]
2.      i = p – 1
3.      for j = p to r – 1
4.          if A[j] ≤ x
5.              i = i + 1
6.              exchange A[i] with A[j]
7.      exchange A[i +1] with A[r]
8.      return i + 1
```

p = 6
r = 7

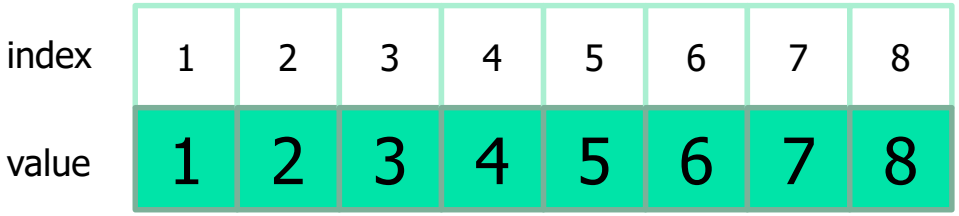
return 6



Quicksort

```
QUICKSORT(A, p, r)
1.    if p < r
2.       q = PARTITION(A, p, r)
3.       QUICKSORT(A, p, q-1)
4.       QUICKSORT(A, q + 1, r)
```

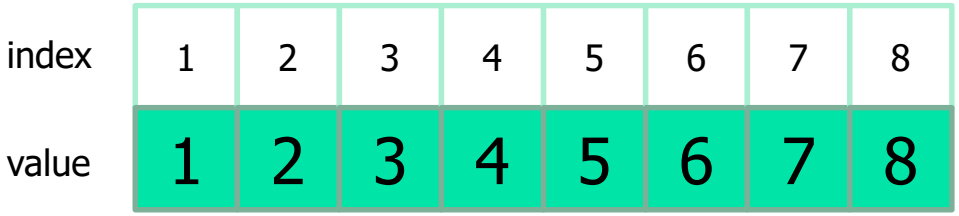
return



p = 6
r = 7
q = 6

Quicksort

```
QUICKSORT(A, p, r)
1.    if p < r
2.       q = PARTITION(A, p, r)
3.       QUICKSORT(A, p, q-1)
4.       QUICKSORT(A, q + 1, r)
```



p = 6
r = 7
q = 6

Quicksort

```
QUICKSORT(A, p, r)
1.    if p < r
2.       q = PARTITION(A, p, r)
3.       QUICKSORT(A, p, q-1)
4.       QUICKSORT(A, q + 1, r)
```

index	1	2	3	4	5	6	7	8
value	1	2	3	4	5	6	7	8

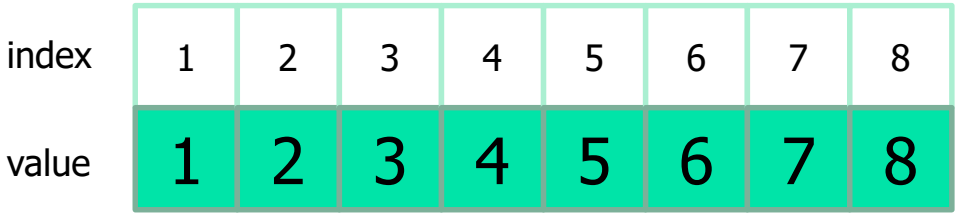
p = 6

r = 5

Quicksort

```
QUICKSORT(A, p, r)
1.    if p < r
2.       q = PARTITION(A, p, r)
3.       QUICKSORT(A, p, q-1)
4.       QUICKSORT(A, q + 1, r)
```

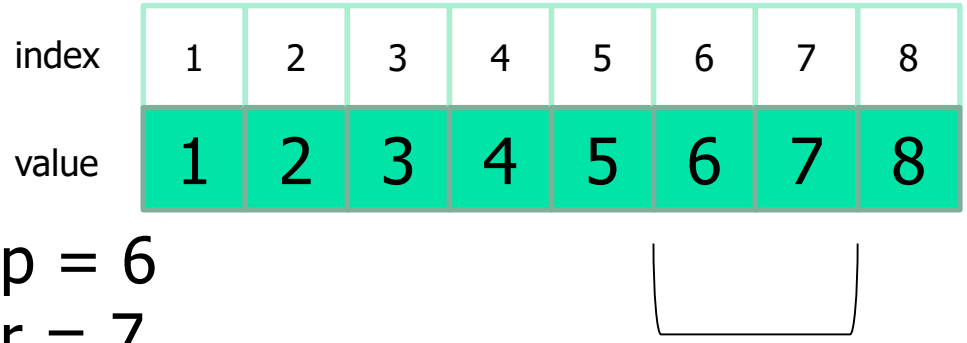
return



p = 6
r = 7
q = 6

Quicksort

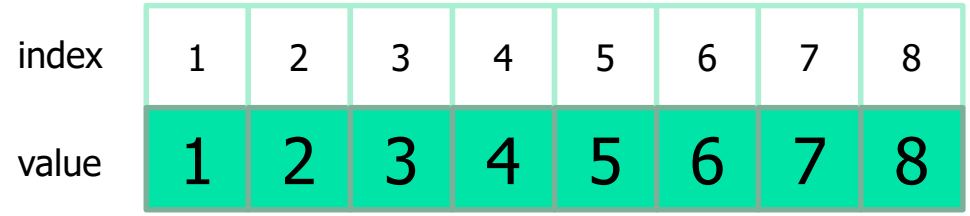
```
QUICKSORT(A, p, r)
1.    if p < r
2.       q = PARTITION(A, p, r)
3.       QUICKSORT(A, p, q-1)
4.       QUICKSORT(A, q + 1, r)
```



p = 6
r = 7
q = 6

Quicksort

```
QUICKSORT(A, p, r)
1.    if p < r
2.       q = PARTITION(A, p, r)
3.       QUICKSORT(A, p, q-1)
4.       QUICKSORT(A, q + 1, r)
```

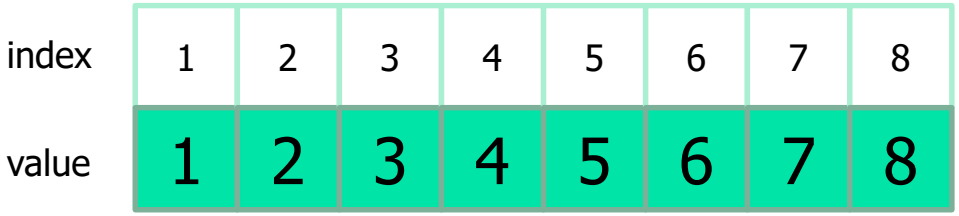


p = 7
r = 7

Quicksort

```
QUICKSORT(A, p, r)
1.    if p < r
2.       q = PARTITION(A, p, r)
3.       QUICKSORT(A, p, q-1)
4.       QUICKSORT(A, q + 1, r)
```

return



p = 6

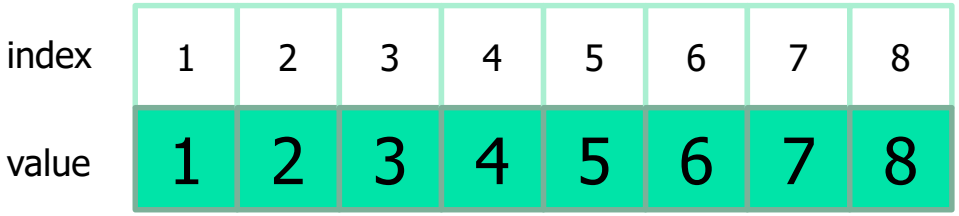
r = 7

q = 6

Quicksort

```
QUICKSORT(A, p, r)
1.   if p < r
2.      q = PARTITION(A, p, r)
3.      QUICKSORT(A, p, q-1)
4.      QUICKSORT(A, q + 1, r)
```

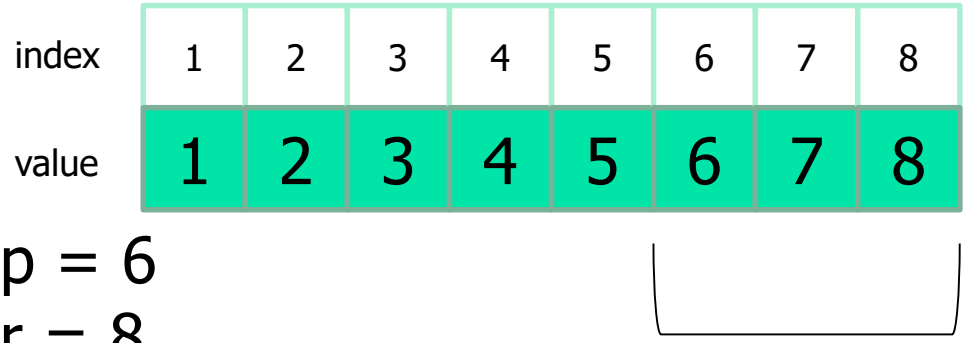
return



p = 6
r = 8
q = 8

Quicksort

```
QUICKSORT(A, p, r)
1.    if p < r
2.       q = PARTITION(A, p, r)
3.       QUICKSORT(A, p, q-1)
4.       QUICKSORT(A, q + 1, r)
```



p = 6
r = 8
q = 8

Quicksort

```
QUICKSORT(A, p, r)
1.    if p < r
2.       q = PARTITION(A, p, r)
3.       QUICKSORT(A, p, q-1)
4.       QUICKSORT(A, q + 1, r)
```

index	1	2	3	4	5	6	7	8
value	1	2	3	4	5	6	7	8

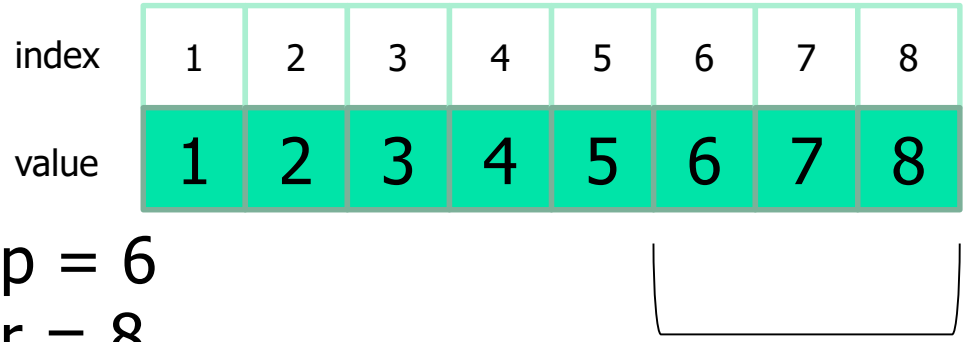
p = 9

r = 8

Quicksort

```
QUICKSORT(A, p, r)
1.    if p < r
2.       q = PARTITION(A, p, r)
3.       QUICKSORT(A, p, q-1)
4.       QUICKSORT(A, q + 1, r)
```

return

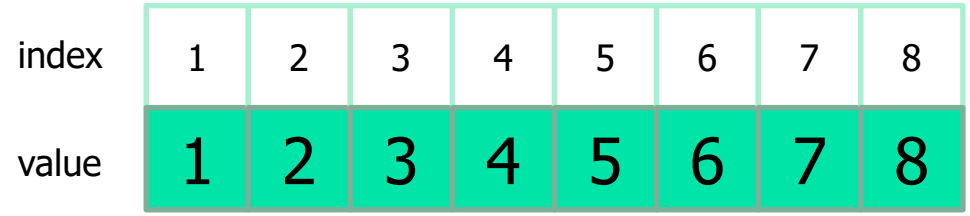


p = 6
r = 8
q = 8

Quicksort

```
QUICKSORT(A, p, r)
1.    if p < r
2.       q = PARTITION(A, p, r)
3.       QUICKSORT(A, p, q-1)
4.       QUICKSORT(A, q + 1, r)
```

return



p = 1

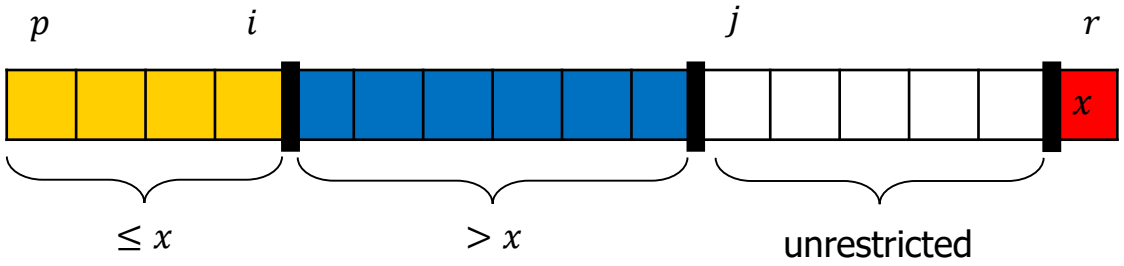
r = 8

q = 5

Fin.

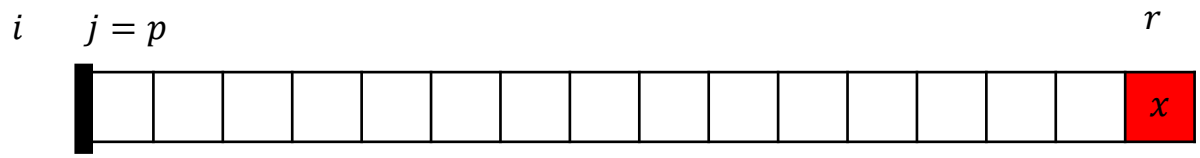
Correctness of PARTITION Procedure

- Loop Invariant:
  - At the start of each iteration for the for-loop of lines 3-6, the subarray $A[p \ldots i]$, $A[i+1 \ldots j-1]$ and $L[k]$ satisfy
    1. If $p \le k \le i$, then $A[k] \le x$
    2. If $i+1 \le k \le j-1$, then $A[k] > x$
    3. If $k = r$, then $A[k] = x$



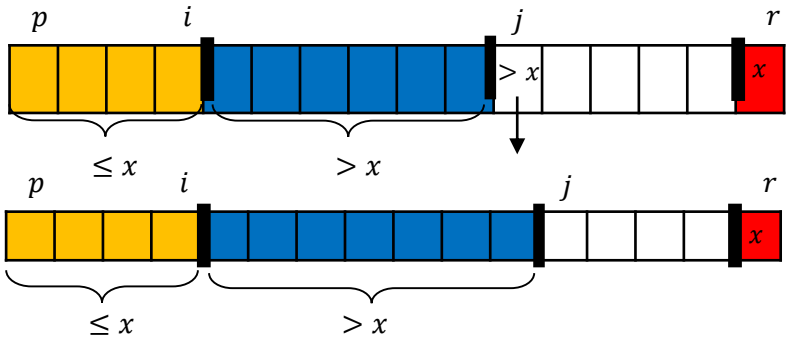
Correctness of PARTITION Procedure

- **Loop Invariant:** At the start of each iteration for the for-loop of lines 3-6, the subarray $A[p \ldots i]$, $A[i+1 \ldots j-1]$ and $L[k]$ satisfy
  1. If $p \leq k \leq i$, then $A[k] \leq x$
  2. If $i+1 \leq k \leq j-1$, then $A[k] > x$
  3. If $k = r$, then $A[k] = x$
- **Initialization:**
  - Prior to the first iteration of the loop, we have i=p-1 and j=p. So the subarrays $A[p \ldots i]$ and $A[i+1 \ldots j-1]$ are empty. The conditions (1) and (2) are trivially true.
  - The assignment in line 1 satisfies the condition (3)



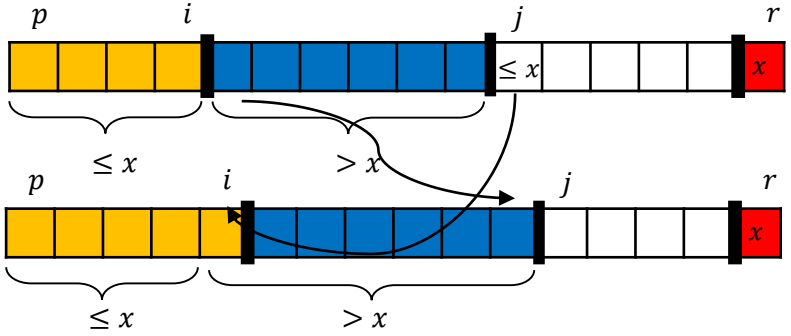
Correctness of PARTITION Procedure

- **Loop Invariant:** At the start of each iteration for the for-loop of lines 3-6, the subarray $A[p \ldots i]$, $A[i+1 \ldots j-1]$ and $L[k]$ satisfy
  1. If $p \leq k \leq i$, then $A[k] \leq x$
  2. If $i+1 \leq k \leq j-1$, then $A[k] > x$
  3. If $k = r$, then $A[k] = x$
- **Maintenance:**
  - When A[j]>x, the only action in the loop is to increment j . After j is incremented, condition (2) holds for A[j-1] and all other entries remain unchanged.



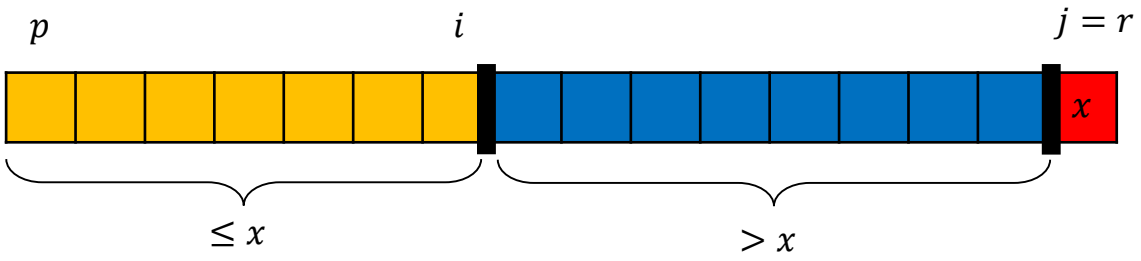
Correctness of PARTITION Procedure

- **Loop Invariant:** At the start of each iteration for the for-loop of lines 3-6, the subarray $A[p \ldots i]$, $A[i+1 \ldots j-1]$ and $L[k]$ satisfy
  1. If $p \leq k \leq i$, then $A[k] \leq x$
  2. If $i+1 \leq k \leq j-1$, then $A[k] > x$
  3. If $k = r$, then $A[k] = x$
- **Maintenance:**
  - When A[j]≤x, the loop increments i, swaps A[i]and A[j], and then increments j . Because of the swap, A[i] ≤ x, and condition (1) is satisfied. Similarly, we also have A[j-1] > x.

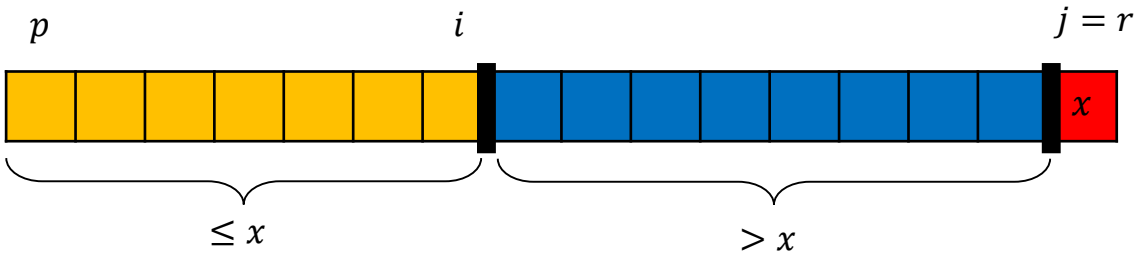


Correctness of PARTITION Procedure

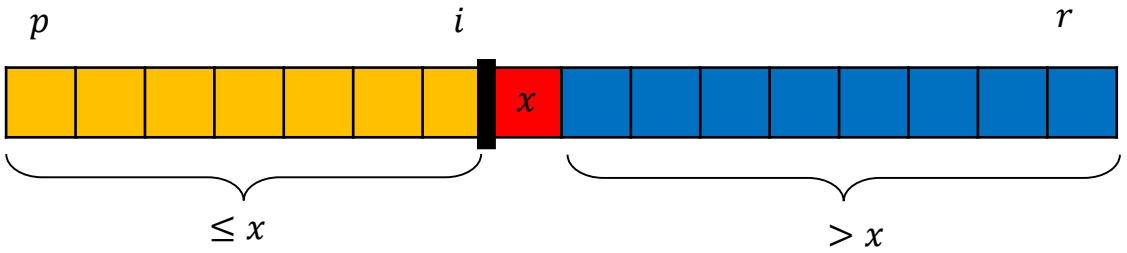
- **Loop Invariant:** At the start of each iteration for the for-loop of lines 3-6, the subarray $A[p \ldots i]$, $A[i+1 \ldots j-1]$ and $L[k]$ satisfy
  1. If $p \leq k \leq i$, then $A[k] \leq x$
  2. If $i+1 \leq k \leq j-1$, then $A[k] > x$
  3. If $k = r$, then $A[k] = x$
- **Termination:**
  - At termination, j=r. Thus, every entry in the array is in one of the three sets described by the invariant, and we have partitioned the values in the array into three sets: those less than or equal to x, those greater than x, and a singleton set containing x.



Output of PARTITION



- The final two lines of PARTITION finish up by swapping the pivot element with the leftmost element greater than x, we get the following and return the pivot's new index



Analysis of Quicksort

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

Performance depends on the selection of pivot.

Worst-case partitioning:

Best-case partitioning:

Analysis of Quicksort

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

Performance depends on the selection of pivot.

Worst-case partitioning: divide $n - 1$ elements and 0 element

$$\begin{aligned} T(n) &= T(n-1) + n \\ &= T(n-2) + (n-1) + n \\ &= T(1) + \sum_{i=2}^{n} i \\ &= O(n^2) \end{aligned}$$

Best-case partitioning:

Analysis of Quicksort

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

Performance depends on the selection of pivot.

Worst-case partitioning: divide $n - 1$ elements and 0 element

$$\begin{aligned} T(n) &= T(n-1) + n \\ &= T(n-2) + (n-1) + n \\ &= T(1) + \sum_{i=2}^{n} i \\ &= O(n^2) \end{aligned}$$

Best-case partitioning: divide $\frac{n}{2}$ and $\frac{n}{2}$ elements

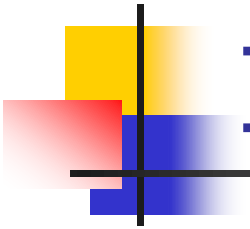
$$\begin{aligned} T(n) &= 2T(n/2) + n \\ &= 2\left(2T\left(\frac{n}{4}\right) + \frac{n}{2}\right) + n = 4T\left(\frac{n}{4}\right) + 2n \\ &= O(n \log n) \end{aligned}$$

Analysis of Quicksort

- Worst-case analysis
  - $T(n) = \max_{0 \le q \le n-1}\big(T(q) + T(n-q-1)\big) + \Theta(n)$
- Guess $T(n) \le cn^2$
  - $T(n) = \max_{0 \le q \le n-1}(cq^2 + c(n-q-1)^2) + \Theta(n)$
    $= c \cdot \max_{0 \le q \le n-1}(q^2 + (n-q-1)^2) + \Theta(n)$
- Since $q^2 + (n-q-1)^2$ has maximum at either endpoint over the parameter's range $0 \le q \le n-1$
  - $\max_{0 \le q \le n-1}(q^2 + (n-q-1)^2) \le (n-1)^2$
- Thus, $T(n) \le cn^2 - c(2n-1) + \Theta(n) \le cn^2$

Intuition for the Average Case

- Suppose that PARTITION algorithm always produce a 9-to-1 proportional split.
  - Quite unlalanced
  - T(n) = T(9n/10) + T(n/10) + cn
  - Figure 7.4 shows a recursion tree for this recurrence
    - Every level of the tree has cost cn until the recursion reaches a boundary condition at depth $\log_{10} n = \theta(\lg n)$
    - Then, the levels have cost at most cn
    - The recursion terminates at depth $\log_{9/10} n = \theta(\lg n)$
    - Thus, the total cost of quicksort is O(n lg n)
    - Asymtotically the same as if the split were right down the middle
- Even a 99-to-1 split yields an O(n lg n) running time



Intuition for the Average Case

- Suppose that the good and bad splits alternate levels in the tree
  - Good splits: the best-case splits
  - The bad split – the worst-case splits
- At the root of the tree, (the worst case)
  - the cost is n for partitioning
  - The subarrays produced have size n-1 and 0
- At the next level, (the best case)
  - The subarray of size n-1 undergoes the bse-case partitioning into subarrays of size (n-1)/2 – 1 and (n-1)/2
- Assume that the boundary condition cost is 1 for the subarray of size 0

Intuition for the Average Case

- The combination of the bad split followed by the good split produces three subarrays of size 0, (n-1)/2 – 1 and (n-1)/2 at a combined partitioning cost of $\theta(n) + \theta(n-1) = \theta(n)$
- This situation is no worse than that in the single level of partitioning that produces two subarrays of size (n-1)/2 at a cost of $\theta(n)$ (i.e., balanced!)
- Intuitively, $\theta(n-1)$ cost of the bad split can be absorbed into the $\theta(n)$ cost of the good split
- Thus, the running time of quick sort, when levels alternate between good and bad split, is like the running time for good splits alone: O(n lg n)

A Randomized Quicksort

```
RANDOMIZED-PARTITION(A, p, r)
1.  i = RANDOM(p, r)
2.  exchange A[r] with A[i]
3.  return PARTITION(A, p, r)
```

```
RANDOMIZED-QUICKSORT(A, p, r)
1.  if p < r
2.     q = RANDOMIZED-PARTITION(A, p, r)
3.     RANDOMIZED-QUICKSORT(A, p, q-1)
4.     RANDOMIZED-QUICKSORT(A, q + 1, r)
```

Analysis of Quicksort

- Lemma 7.1
  - Let X be the number of comparisons performed in line 4 of PARTITION over the entire execution of QUICKSORT on an n-element array. Then the running time of QUICKSORT is $O(n + X)$
- Our goal is to compute X, the total number of comparisons in all calls to PARTITION

```
PARTITION(A, p, r)
1.    x = A[r]
2.    i = p – 1
3.    for j = p to r – 1
4.        if A[j] ≤ x
5.            i = i + 1
6.            exchange A[i] with A[j]
7.    exchange A[i +1] with A[r]
8.    return i + 1
```

Analysis of Quicksort

- Observations:
  - Each time PARTITION is called, it selects a pivot which is never included in any further recursive calls to QUICKSORT and PARTITION
  - Thus, there can be at most n calls to PARTITION over the entire execution of the quicksort
  - In each invocation of PARTITION, a constant amount of work is done and then the for-loop is executed for some number of times.
  - Thus, the running time of QUICKSORT is O(n+X)
- We want to count the number of times that line 4 in PARTITION is executed

Analysis of Quicksort

- Rename the elements of the array A as $z_1, z_2, \dots, z_n$ with z_i being the i-th smallest elements
  - e.g.) $A = \{3,5,1,9\}$, $z_1 = 1$, $z_2 = 3$, $z_3 = 5$, $z_4 = 9$
- $Z_{i,j} = \{z_i, z_{i+1}, \dots z_j\}$: the set of elements between z_i an z_j
- $X_{i,j} = \mathrm{I}\{z_i \text{ is compared to } z_j\}$: the indicator random variable where $\mathrm{I} = 1$ when z_i is compared to z_j, otherwise $\mathrm{I} = 0$

Analysis of Quicksort

- Total number of comparisons

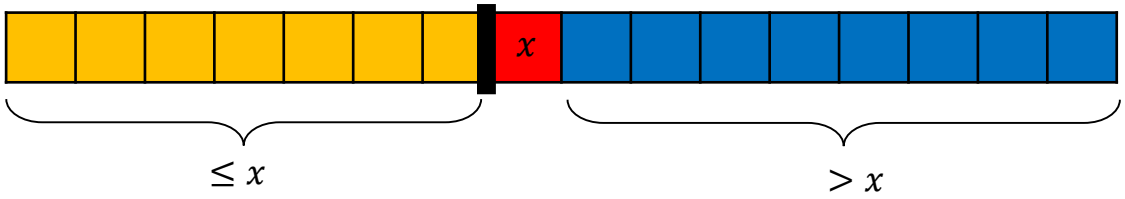
  $X = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{i,j}$

- Expectation of total number of comparisons

$$\begin{aligned} \mathrm{E}[X] &= \mathrm{E}\left[\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{i,j}\right] \\ &= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \mathrm{E}\left[X_{i,j}\right] \\ &= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \Pr\{z_i \text{ is compared to } z_j\} \end{aligned}$$

Analysis of Quicksort

- When do we compare z_i and z_j?
  - Each pair of elements is compared at most once.
  - Why? - Elements are compared only to the pivot and it is never compared to any other elements.
  - If $z_i < x < z_j$ is chosen as a pivot, z_i and z_j cannot ever be compared
  - If z_i is chosen as a pivot, z_i will be compared to every item in $Z_{i,j}$.
  - If z_j is chosen as a pivot, z_j will be compared to every item in $Z_{i,j}$.



Analysis of Quicksort

- $\Pr\{z_i \text{ compared to } z_j\} = 2/(j-i+1)$
- Proof
  - We assume that the RANDOMIZED-PARTITION chooses each pivot randomly and independently.
  - $\Pr\{z_i \text{ compared to } z_j \text{ from } Z_{i,j}\} = \frac{2}{j-i+1}$
  - $\Pr\{z_i \text{ compared to } z_j \text{ from } Z_{i-1,j}\}$
    $= \frac{1}{j-i+2} \cdot \Pr\{z_i \text{ compared to } z_j \text{ from } Z_{i,j}\} + \frac{2}{j-i+2} \cdot 1 + \frac{j-i-1}{j-i+2} \cdot 0$
    $= \frac{1}{j-i+2} \cdot \frac{2}{j-i+1} + \frac{2}{j-i+2} \cdot 1 + \frac{j-i-1}{j-i+2} \cdot 0 = \frac{2}{j-i+1}$
  - $\Pr\{z_i \text{ compared to } z_j \text{ from } Z_{i,j+1}\}$
    $= \frac{1}{j-i+2} \cdot \Pr\{z_i \text{ compared to } z_j \text{ from } Z_{i,j}\} + \frac{2}{j-i+2} \cdot 1 + \frac{j-i-1}{j-i+2} \cdot 0$
    $= \frac{1}{j-i+2} \cdot \frac{2}{j-i+1} + \frac{2}{j-i+2} \cdot 1 + \frac{j-i-1}{j-i+2} \cdot 0 = \frac{2}{j-i+1}$

Analysis of Quicksort

- $\Pr\{z_i$ compared to z_j from $Z_{i-1,j+1}\}$

$$= \frac{1}{j-i+3} \cdot \Pr\{z_i \text{ compared to } z_j \text{ from } Z_{i,j+1}\} \cdot 1$$

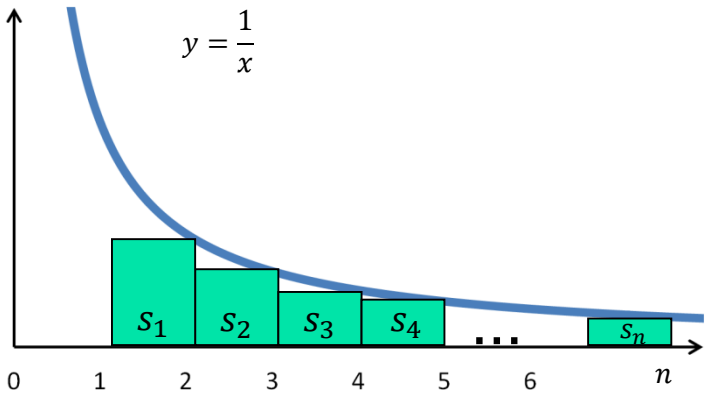
$$+ \frac{1}{j-i+3} \cdot \Pr\{z_i \text{ compared to } z_j \text{ from } Z_{i-1,j}\} \cdot 1 + \frac{2}{j-i+3} \cdot 1 + \frac{j-i-1}{j-i+3} \cdot 0$$

$$= \frac{2}{j-i+3} \cdot \left(\frac{2}{j-i+1}\right) \cdot 1 + \frac{2}{j-i+3} \cdot 1 + \frac{j-i-1}{j-i+3} \cdot$$

$$= \frac{2}{j-i+3} \cdot \left(\frac{2}{j-i+1} + 1\right) \cdot 1 = \frac{2}{j-i+3} \cdot \left(\frac{2+2j-2i+4}{j-i+1}\right) = \frac{2}{j-i+1}$$

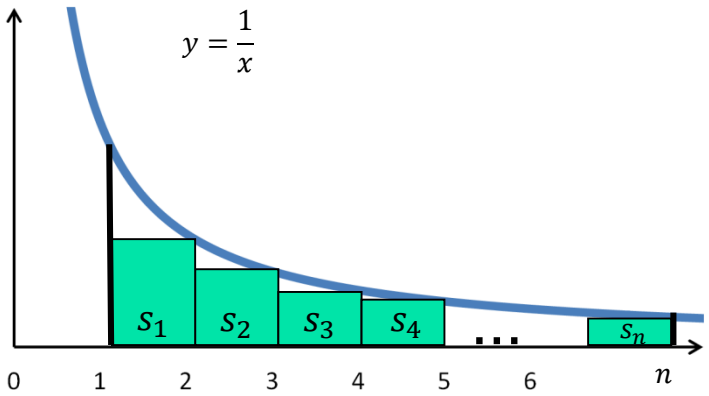
- $\Pr\{z_i$ compared to z_j from $Z_{1,j}\} = \frac{2}{j-i+1}$
- $\Pr\{z_i$ compared to $z_j$from $Z_{1,n}\} = \frac{2}{j-i+1}$

Analysis of Quicksort



- $S = s_1 + s_2 + s_3 + \cdots s_n = \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n}$

Analysis of Quicksort



- $S = s_1 + s_2 + s_3 + \cdots s_n = \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n}$

$$< \int_1^n \frac{1}{x} dx = [\ln x]_1^n = \ln n$$

Analysis of Quicksort

- $\sum_{k=2}^{n} \frac{1}{k} = \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n} < \int_{1}^{n} \frac{1}{x} dx = [\ln x]_1^n = \ln n$
- $\sum_{k=1}^{n} \frac{2}{k+1} = 2\left(\frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n} + \frac{1}{n+1}\right) < 2\ln(n+1) = O(\lg n)$

Analysis of Quicksort

- $E[X] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \Pr\{z_i \text{ is compared to } z_j\}$

$$= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j-i+1}$$

$$= \sum_{i=1}^{n-1} \sum_{k=1}^{n-i} \frac{2}{k+1}$$

$$< \sum_{i=1}^{n-1} \sum_{k=1}^{n} \frac{2}{k+1}$$

$$= \sum_{i=1}^{n-1} O(\lg n)$$

$$= O(n \lg n)$$

Quick Sort Average Time

- Lemma 7.1: Let $T_{avg}(n)$ be the expected time for function QuickSort to sort a list with n records. Then there exists a constant k such that $T_{avg}(n) \leq knlog_e n$ for $n \geq 2$.

부분적분

- $\int x \log x \, dx = \frac{x^2 \log x}{2} - \frac{x^2}{4}$

Quick Sort Average Time

- Proof ($T_{avg}(n) \le kn\log_e n$ for $n \ge 2$)
  - We have

$$T_{avg}(n) \le cn + \frac{1}{n}\sum_{j=1}^{n}(T_{avg}(j-1) + T_{avg}(n-j)) = cn + \frac{2}{n}\sum_{j=0}^{n-1} T_{avg}(j) \quad (7.1)$$

  - We assume $T_{avg}(0) \le b$ and $T_{avg}(1) \le b$
  - Induction base: For n=2, $T_{avg}(2) \le 2c + 2b \le 2k\log_e 2.$
  - Induction hypothesis: Assume $T_{avg}(n) \le kn\log_e n$ for $1 \le n < m$
  - Induction step: From Eq. (7.1) and the induction hypothesis we have

$$T_{avg}(m) \le cm + \frac{4b}{m} + \frac{2}{m}\sum_{j=2}^{m-1} T_{avg}(j) \le cm + \frac{4b}{m} + \frac{2k}{m}\sum_{j=2}^{m-1} j\log_e j \quad (7.2)$$

  - Since $j\log_e j$ is an increasing function of j, Eq. (7.2) yields

$$T_{avg}(m) \le cm + \frac{4b}{m} + \frac{2k}{m}\sum_{j=2}^{m-1} j\log_e j \le cm + \frac{4b}{m} + \frac{2k}{m}\int_2^m x\log_e x\,dx = cm + \frac{4b}{m} + \frac{2k}{m}\left[\frac{m^2\log_e m}{2} - \frac{m^2}{4}\right]$$

$$= cm + \frac{4b}{m} + km\log_e m - \frac{km}{2} \le km\log_e m, \text{ for } m \ge 2$$

Any Question?