Data Structures & Algorithms

《数据结构与算法》课程组
重庆大学计算机学院

4

QUICK SORT AND AVERAGE TIME COMPLEXITY

Outline

4.1 Basic Quick Sort

4.2 Analysis of Quick Sort

4.3 Improving Quick Sort with Medians

4.1 Basic Quick Sort

QUICK SORT

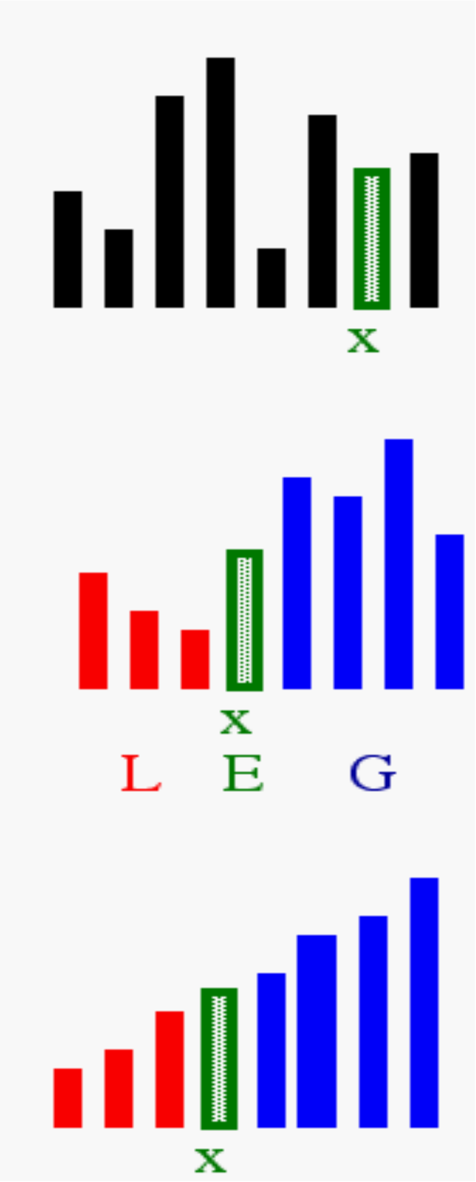
- **Two O(n log n) sorting algorithms:**
  - **Merge sort which is faster but requires more memory**
  - 堆排
  - **Heap sort which allows in-place sorting (will learn later!)**
- **We will now look at a recursive algorithm which may be done *almost* in place and usually faster than heap sort**
  - **Use an object in the array (a pivot) to divide the two**
  - **Average case:** **O(n log n) time and O(log n) memory**
  - **Worst case:** **O(n^2) time and O(n) memory**

Quicksort: Idea

1) Select: pick an element

2) Divide: rearrange elements so that x goes to its final position E

3) Conquer: recursively sort



Quicksort

- **For example, given an unsorted array:**

2	8	7	1	3	5	6	**4**

- **We can select the last entry, 4, and sort the remaining entries into two groups, those less than 4 and those greater than 4:**

2	1	3	4	7	5	6	**8**

- **Note that 4 is now in the correct location once the list is sorted**
  - **Proceed by applying the algorithm to the first 3 and last 4 entries**

A Basic Implementation - PARTITION

```
template <typename E, typename Comp>
inline int partition(E A[], int l, int r, E& pivot) {
  do {              // Move the bounds inward until they meet
    while (Comp::prior(A[++l], pivot));  // Move l right and
    while ((l < r) && Comp::prior(pivot, A[--r])); // r left
    swap(A, l, r);                      // Swap out-of-place values
  } while (l < r);                      // Stop when they cross
  return l;      // Return first position in right partition
}
```

Partition: example

Initial		72	6	57	88	85	42	83	73	48	60
	l										r
Pass 1		72	6	57	88	85	42	83	73	48	60
		l								r	
Swap 1		48	6	57	88	85	42	83	73	72	60
		l								r	
Pass 2		48	6	57	88	85	42	83	73	72	60
					l		r				
Swap 2		48	6	57	42	85	88	83	73	72	60
					l		r				
Pass 3		48	6	57	42	85	88	83	73	72	60
						l,r					

A Simple Implementation – PARTITION

- **PARTITION** (A, p, r)

  $x \leftarrow A[r]$

  $i \leftarrow p$-1

  FOR $j \leftarrow p$ **TO** r-1

      IF $A[j] \leq x$ // **comp::prior($A[j]$, x)**

          THEN $i \leftarrow i + 1$

              exchange $A[i]$ ↔ $A[j]$

  exchange $A[i+1]$ ↔ $A[r]$

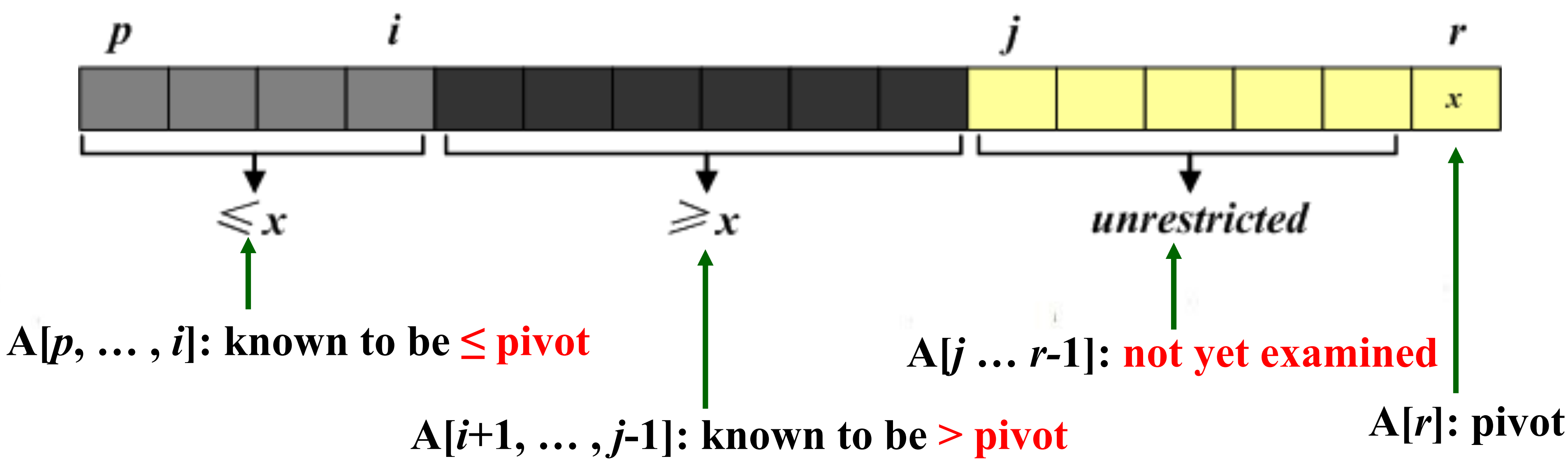
  RETURN $i+1$

A Simple Implementation – PARTITION

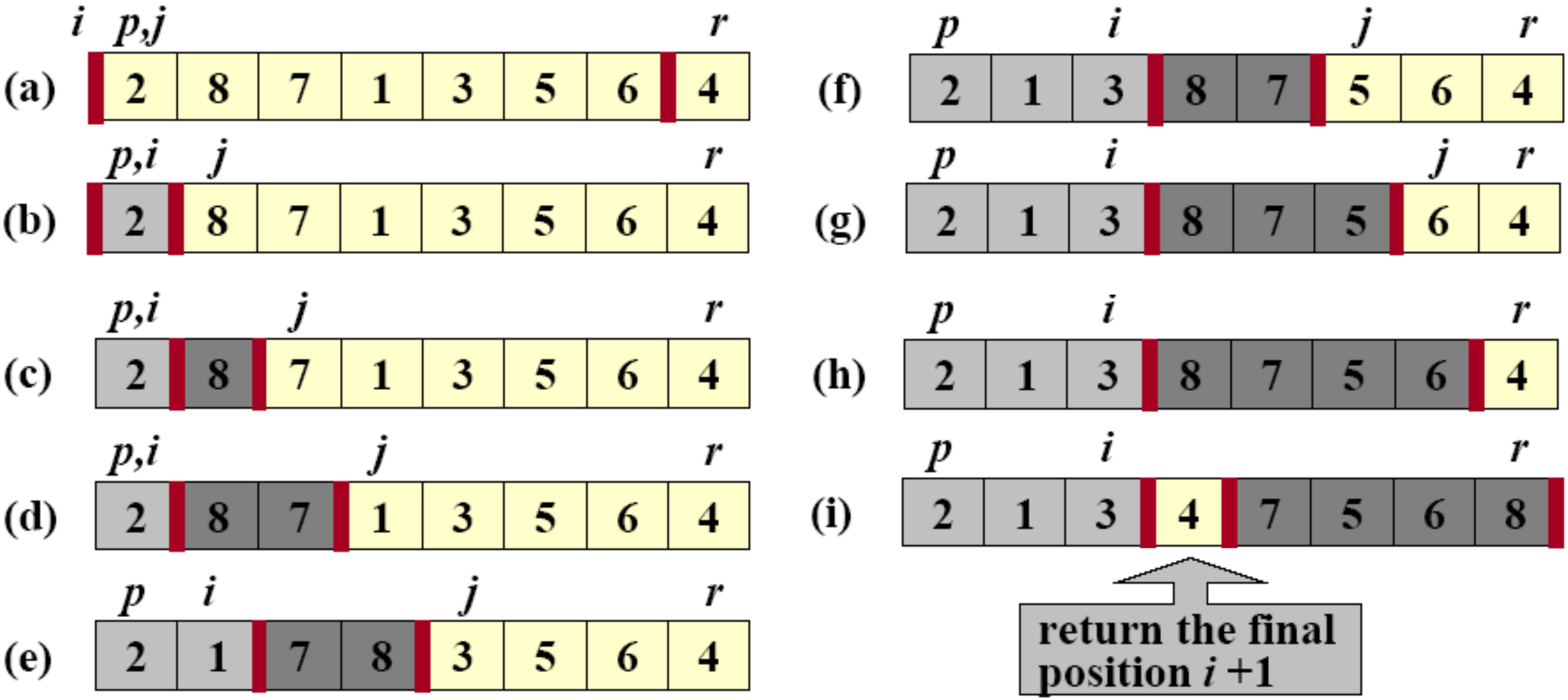
PARTITION

```
PARTITION(A, p, r)          //A[p..r]
    x ← A[r]                //the rightmost element as pivot
    i ← p-1
  for j ← p to r-1
        do if A[j] ≤ x
               then i ← i+1
                    exchange A[i]↔A[j]
  exchange A[i+1]↔A[r]
  return i+1
```

Running time = $O(n)$ for n elements



A Simple Implementation – PARTITION



- The operation of Partition on the sample array. Lightly shaded array elements are all with values no greater than x (the pivot). Heavily shaded array elements are all with values greater than x.

Main Procedure – QUICKSORT

- QUICKSORT (A, p, r)

  IF $p < r$

  THEN $q \leftarrow$ PARTITION (A, p, r)

  QUICKSORT (A, p, q–1)

  QUICKSORT (A, q+1, r)

- Initial call: QUICKSORT(A, 1, n)

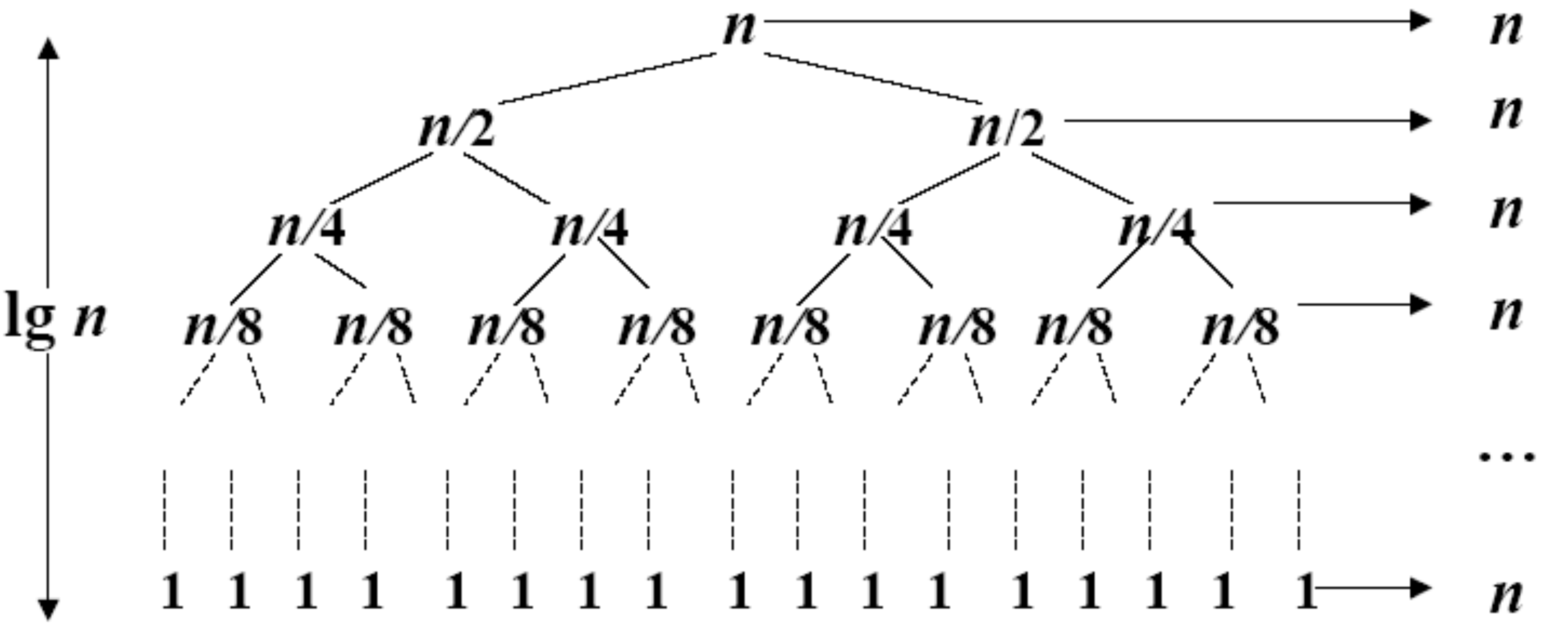
4.2 Analysis of Quick Sort

Run-time Analysis

- **In the best case, the list will be split into two approximately equal sub-lists, and thus, the run time could be very similar to that of merge sort: $\Theta(n \log n)$**

Recursive Tree of the Best Case

- A recursion tree for quick sort in which the partition always balances the two sides of the partition equally. The resulting running time is $\Theta(n \log n)$
- The question is: WHAT happens if we don't get that lucky?



Worst-case Scenario

- Suppose we choose the smallest element as our pivot and we try ordering a sorted list:

80	38	95	84	66	10	79	2	26	87	96	12	43	81	3

- Using 2, we partition the original list into

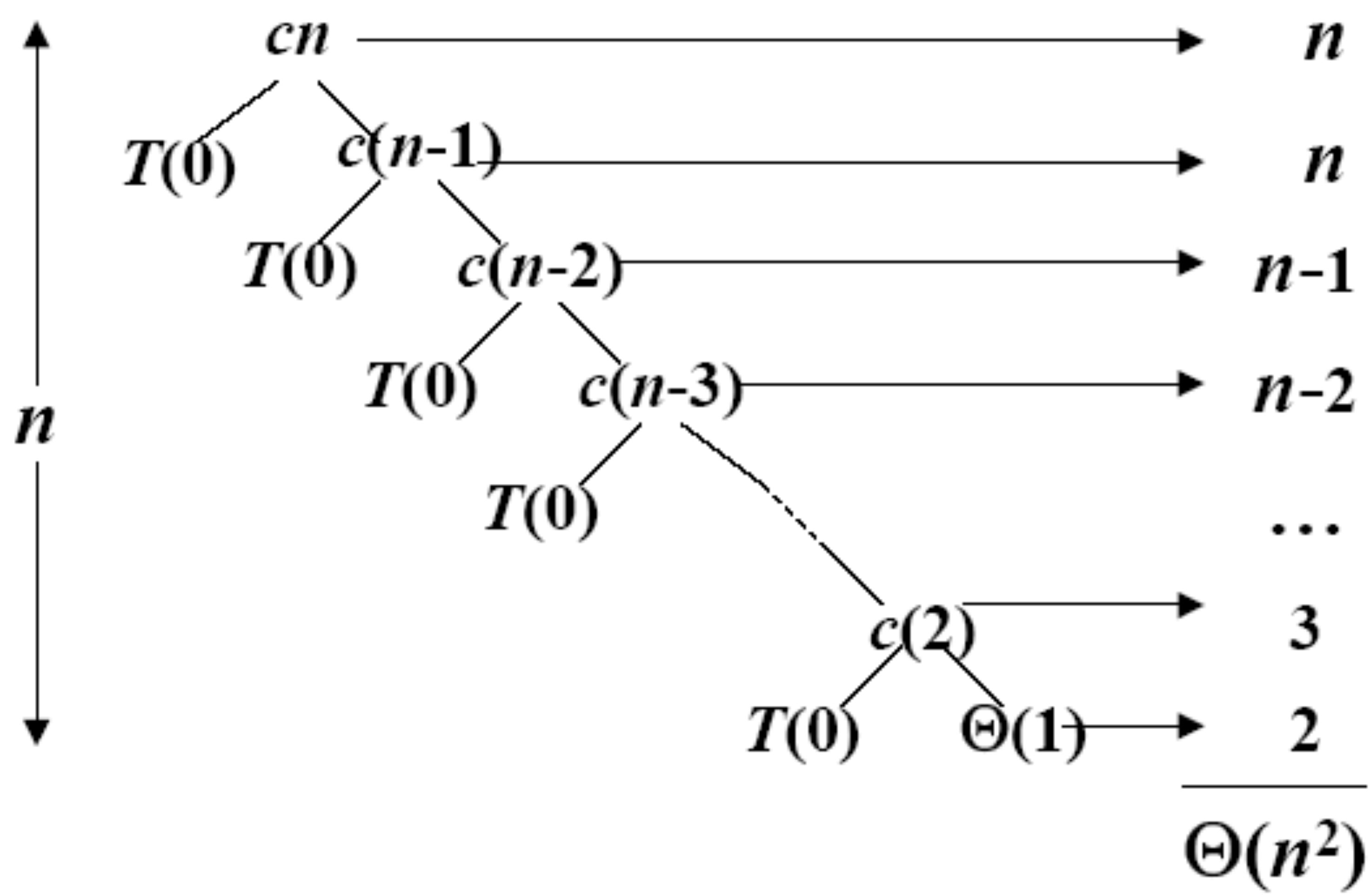
2	80	38	95	84	66	10	79	26	87	96	12	43	81	3

- We still have to sort a list of size n – 1

- The run time is $T(n) = T(n-1) + \Theta(n) = \Theta(n^2)$
  - Thus, the run time drops from $\Theta(n \log n)$ to $\Theta(n^2)$

Recursive Tree of the Worst Case

- **A recursion tree for quick sort in which the partition always puts only a single element on one side of the partition. The resulting running time is $\Theta(n^2)$**



Average-case Scenario

- If we choose a random pivot, this will, on average, divide a set of *n* items into two sets of size $\frac{n}{4}$ and $\frac{3n}{4}$.
- 80 % of the time the width will have a ratio of 1:9 or better.
- 90 % of the time the width will have a ratio of 1:19 or better.

Average-case Scenario (prove)

- 假设 $a_1 < \cdots < a_k < \cdots < a_n$，且$\mathrm{perm}(a_1, \ldots, a_n)$表示$n$个元素的某个序列。因此，共有$n!$种不同的序列，而 a_k $(1 \le k \le n)$在末尾的序列有$(n-1)!$，概率为$\frac{1}{n}$。
- 考虑a_k出现在序列末尾，作为基准值划分的结果如下：

$$\mathrm{perm}(a_1, \ldots a_{k-1}),\ a_k, \mathrm{perm}(a_{k+1}, \ldots, a_n)$$

划分比例（平衡性） = 数量少的子数组长度 / 数量多的子数组长度

- a_k 划分出的数量少的子数组长度 = $\begin{cases} k-1, & \text{if } k \le \frac{n}{2} \\ n-k, & \text{if } k > \frac{n}{2} \end{cases}$
- 数量少的子数组的平均长度

$$\frac{1}{n}\Big(\sum_{1\le k\le\frac{n}{2}}(k-1) + \sum_{\frac{n}{2}<k\le n}(n-k)\Big) = \frac{1}{n}\sum_{1\le k\le\frac{n}{2}}\left[k-1+n-\left(\frac{n}{2}+k\right)\right]$$

$$= \frac{1}{n}\sum_{1\le k\le\frac{n}{2}}\left(\frac{n}{2}-1\right) \approx \frac{n}{4} \quad (n \to \infty)$$

选a_k作为基准的概率（一样分布）

Average-case Scenario (prove)

- 假设 $a_1 < \cdots < a_k < \cdots < a_n$，且$\text{perm}(a_1, \ldots, a_n)$表示$n$个元素的某个序列。因此，共有$n!$种不同的序列，而 a_k $(1 \leq k \leq n)$在末尾的序列有$(n-1)!$，概率为$\frac{1}{n}$。
- 考虑a_k出现在序列末尾，作为基准值划分的结果如下：

$$\text{perm}(a_1, \ldots a_{k-1}),\ a_k, \text{perm}(a_{k+1}, \ldots, a_n)$$

如果每个元素被选为基准值的机会一样，则划分出的（数量少和数量多）两组的平均长度比为 1 : 3

Q: 长度比不低于 1:9 的概率是多少？

A: 如果选 a_k $(k \in \left[\frac{n}{10}, \frac{9n}{10}\right])$为基准值，则长度比大于等于 1:9

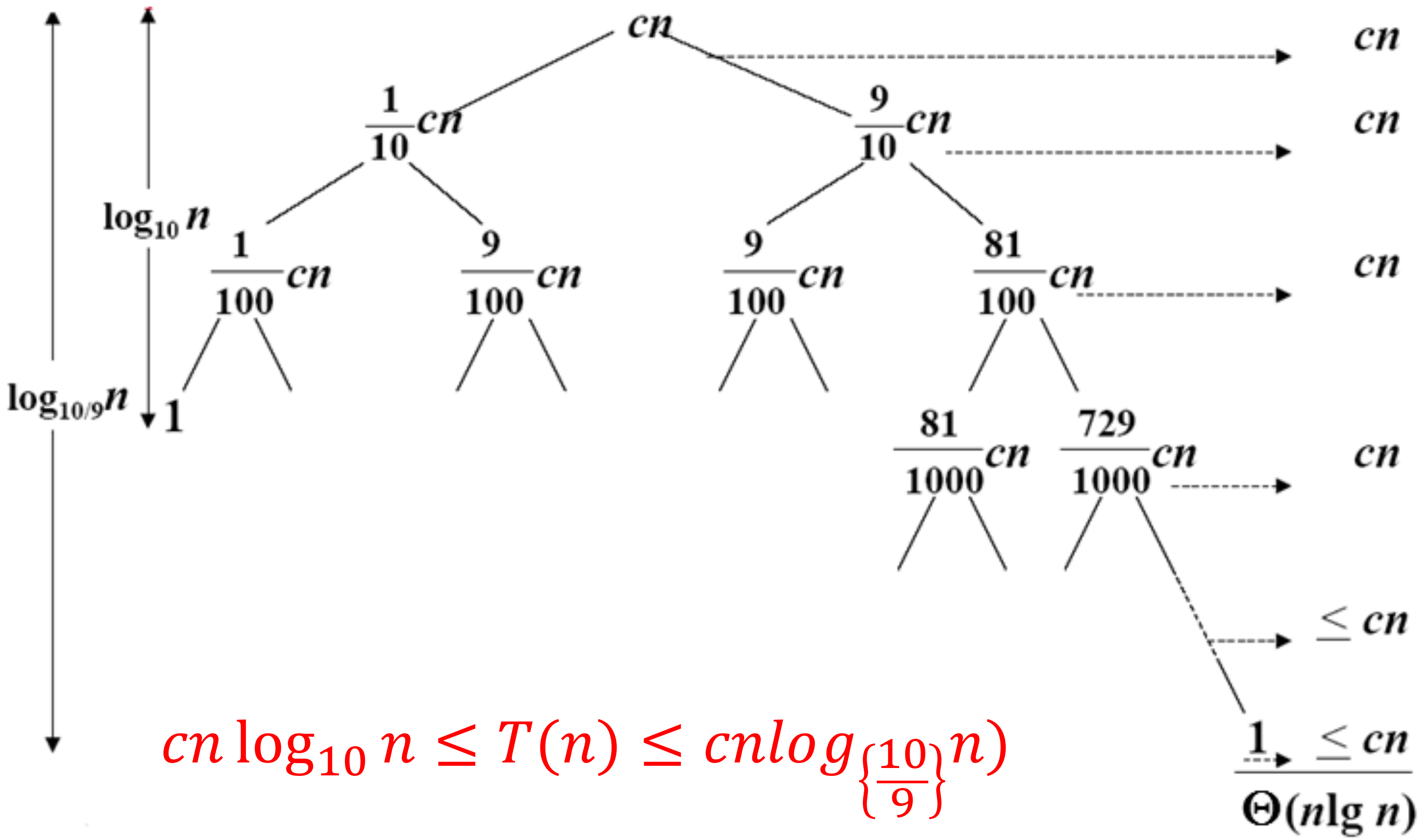
A: 因此，概率 = $\frac{1}{n} * \left(\frac{9n}{10} - \frac{n}{10}\right) = 0.8$

随机选择（或选择末尾）元素作为基准，80%以上的几率划分出的两组长度比好于 1 : 9

Recursive Tree of the Unbalanced Case

- **What if the split is always 1:9?**
  - $T(n) = T(9n/10) + T(n/10) + \Theta(n)$
  - **What is the solution to this recurrence?**

Recursive Tree of the Unbalanced Case



$$cn \log_{10} n \leq T(n) \leq cnlog_{\left\{\frac{10}{9}\right\}} n)$$

- A recursion tree for quick sort in which partition always produces a 1-to-9 split, yielding a running time of **$\Theta(n \log n)$**

Recursive Tree of the Unbalanced Case

- **What if the split is always 1:9?**
  - $T(n) = T(9n/10) + T(n/10) + \Theta(n)$
  - Prove $T(n) = \Theta\left(n \log(n)\right)$ by induction

Recursive Tree of the Unbalanced Case

– $\mathrm{T}(n) = \mathrm{T}(9n/10) + \mathrm{T}(n/10) + \Theta(n)$

Assume $T(n) \le cn\log(n)$ **Then**

$$T(n) \le \frac{9}{10}cn\log\left(\frac{9}{10}n\right) + \frac{1}{10}cn\log\left(\frac{1}{10}n\right) + dn$$

$$= \frac{9}{10}cn\log(n) + \frac{9}{10}cn\log\left(\frac{9}{10}\right) + \frac{1}{10}cn\log(n) + \frac{1}{10}cn\log\left(\frac{1}{10}\right) + dn$$

$$= cn\log(n) + \left(\underbrace{d}_{>0} + \underbrace{c\left[\frac{9}{10}\log\left(\frac{9}{10}\right) + \frac{1}{10}\log\left(\frac{1}{10}\right)\right]}_{<0} \right) n$$

$$\exists\, c > 0: \quad d + c\left[\frac{9}{10}\log\left(\frac{9}{10}\right) + \frac{1}{10}\log\left(\frac{1}{10}\right)\right] \le 0$$

$$T(n) = O\big(n\log(n)\big)$$

- Similarly, we can prove $T(n) = \Omega\big(n\log(n)\big)$

4.3 Improving Quick Sort with Medians

Alternate Strategy

- Our goal is to choose the median element in the list as our pivot:

80	38	95	84	66	10	79	2	26	87	96	12	43	81	3

- Unfortunately, it's DIFFICULT to find
- Alternate strategy: take the median of a subset of entries
  - For example, take the median of the first, middle, and last entries

Choose the Median-of-Three

- It is difficult to find the median so consider another strategy:
  - Choose the median of the first, middle, and last entries in the list

80	38	95	84	99	10	79	44	26	87	96	12	43	81	3

- This will usually give a **much better** approximation of the actual median

Choose the Median-of-Three

- Sorting the elements based on **44** results in two sub-lists, each of which must be sorted (again, using quicksort)

- We select the **26** to partition the first sub-list:

38	10	26	12	43	3	44	80	95	84	99	79	87	96	81

- and **81** to partition the second sub-list:

38	10	26	12	43	3	44	80	95	84	99	79	87	96	81

A Simple Implementation of Median

```
median(A, a, b, c)
if A[a] < A[b]
    then if A[b] < A[c]
            then return b
         else if A[a] < A[c]
                 then return c
              else
                 return a
else if A[c] < A[b]
        then return b
     else if A[a] < A[c]
        then return c
     else
        return a
```

Choose the Median-of-Three

三数取中

```
median(A, a, b, c)
    if A[a] < A[b]
        then if A[b] < A[c]
                then return b
             else if A[a] < A[c]
                then return c
             else
                return a
    else if A[c] < A[b]
                then return b
        else if A[a] < A[c]
                then return c
             else
                return a
```

Q: 设 $a_1 < \cdots < a_k < \cdots < a_n$，随机挑选元素，$a_k$被选的概率为$\frac{1}{n}$。但按三数取中法，$a_k$被选着基准(pivot)的概率是多少？

A: a_k成为基准(pivot)的条件：

三数中
- ① 选了a_k ---------------------- $p_1 = \frac{1}{n}$
- ② 选了比a_k小的元素 ------ $p_2 = \frac{k-1}{n-1}$
- ③ 选了比a_k大的元素 ------ $p_3 = \frac{n-k}{n-2}$

- 上述1,2,3的顺序可以交换

增大靠近中位值的概率.

A: $P(a_k \rightarrow pivot) = \dfrac{3!\,(k-1)(n-k)}{n(n-1)(n-2)}$

Choose the Median-of-Three

- If we choose a random pivot, this will, on average, divide a set of n items into two sets of size $\frac{n}{4}$ and $\frac{3n}{4}$.
  - 90 % of the time the width will have a ratio of 1:19 or better.
- Choosing the median-of-three will, on average, divide the n items into two sets of size $\frac{5n}{16}$ and $\frac{11n}{16}$.
  - Median-of-three helps speed the algorithm
  - 90 % of the time the width will have a ratio of 1:6.388 or better.
- Further, we can apply insertion sort to sorting the small sub-arrays.

Improved Quick Sort Example

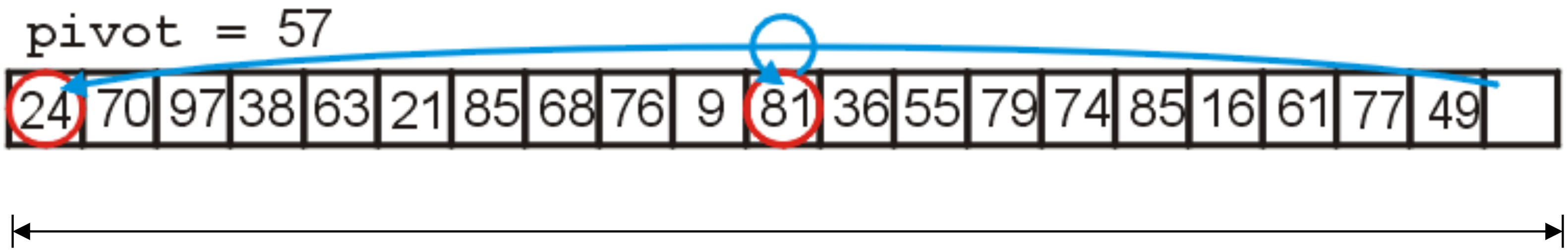
- First, we examine the first, middle, and last entries of the full list
- The span below will indicate which list we are currently sorting

pivot =

57	70	97	38	63	21	85	68	76	9	81	36	55	79	74	85	16	61	77	49	24

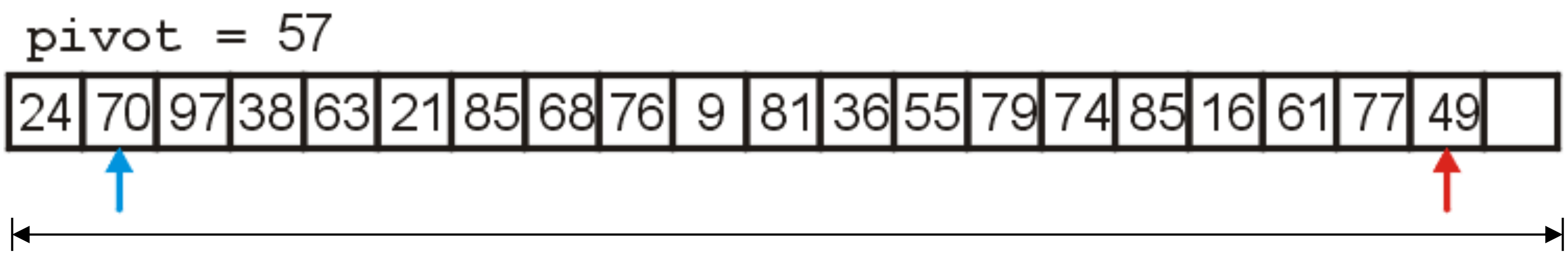
Improved Quick Sort Example

- We select **57** to be our pivot
- We move **24** into the first location



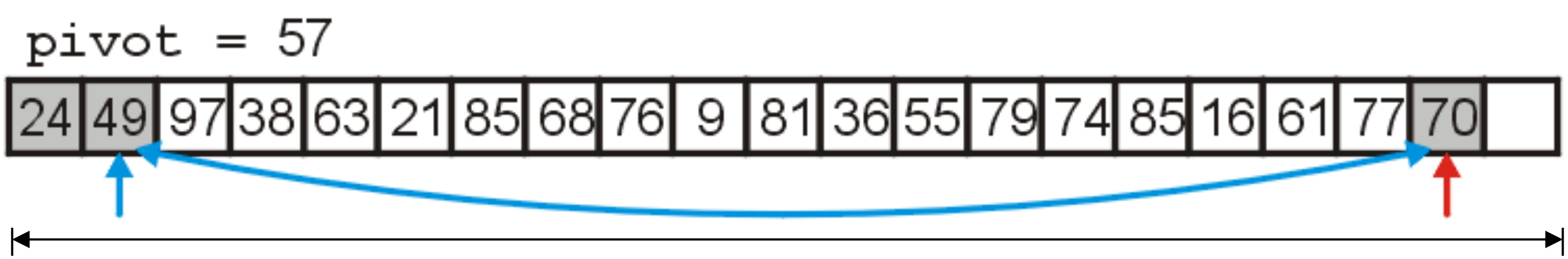
Improved Quick Sort Example

- Starting at the 2^{nd} and 2^{nd}-last locations:
- we search forward till we find **70 > 57**
- we search backward till we find **49 < 57**



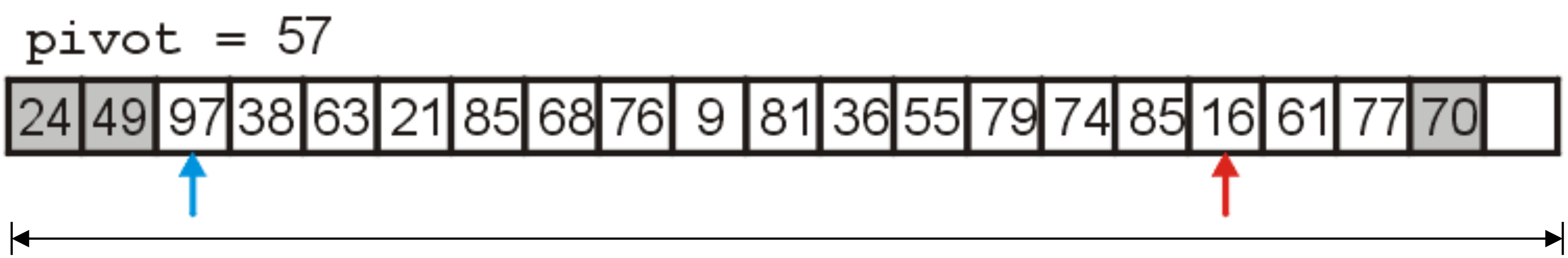
Improved Quick Sort Example

- We swap 70 and 49, placing them in order with respect to each other



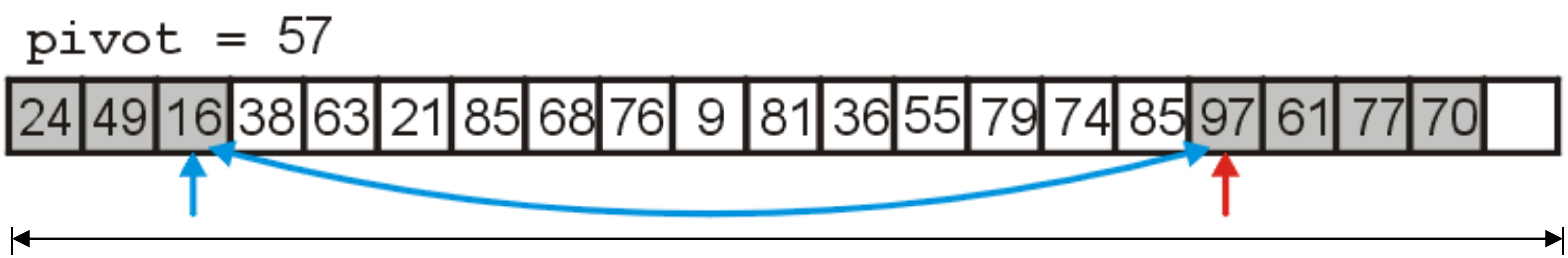
Improved Quick Sort Example

- We search forward until we find **97 > 57**
- We search backward until we find **16 < 57**



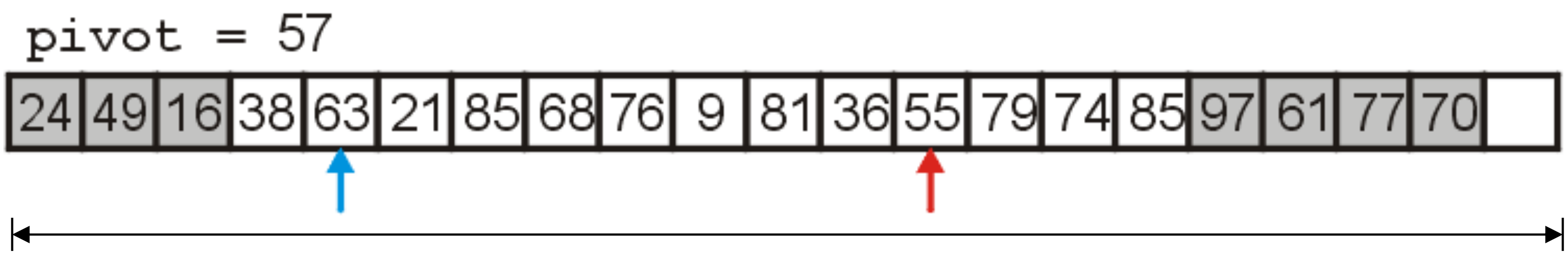
Improved Quick Sort Example

- **We swap 16 and 97 which are now in order with respect to each other**



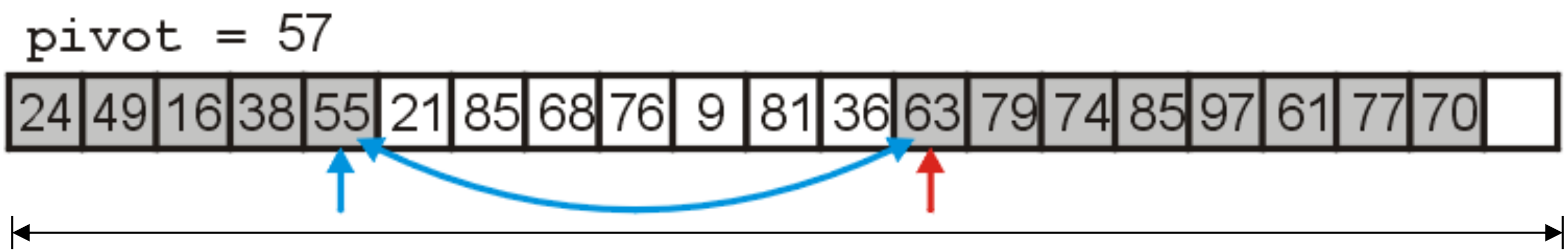
Improved Quick Sort Example

- We search forward till we find **63 > 57**
- We search backward till we find **55 < 57**



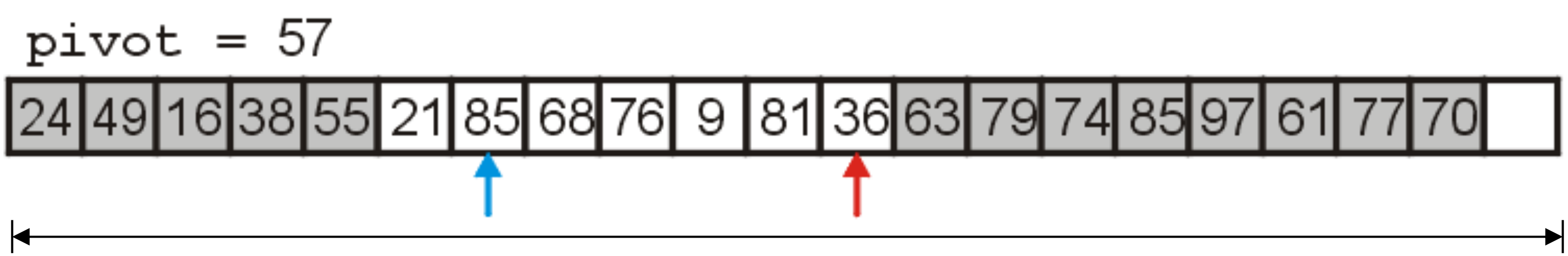
Improved Quick Sort Example

- We swap 63 and 55



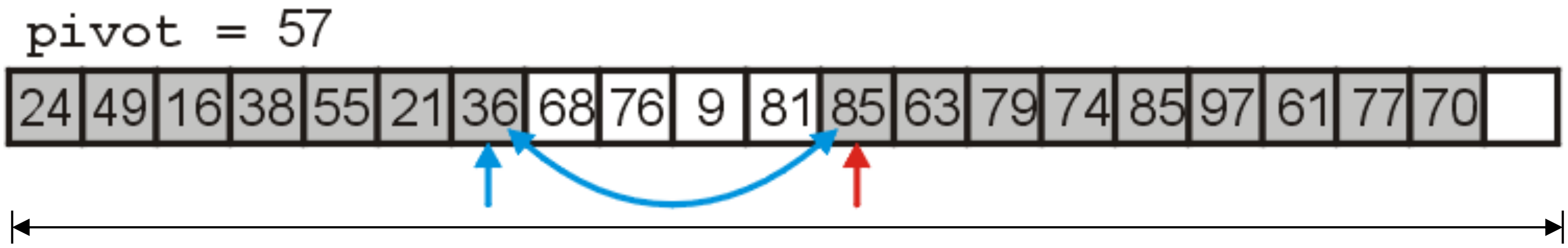
Improved Quick Sort Example

- **We search forward till we find** **85 > 57**
- **We search backward till we find** **36 < 57**



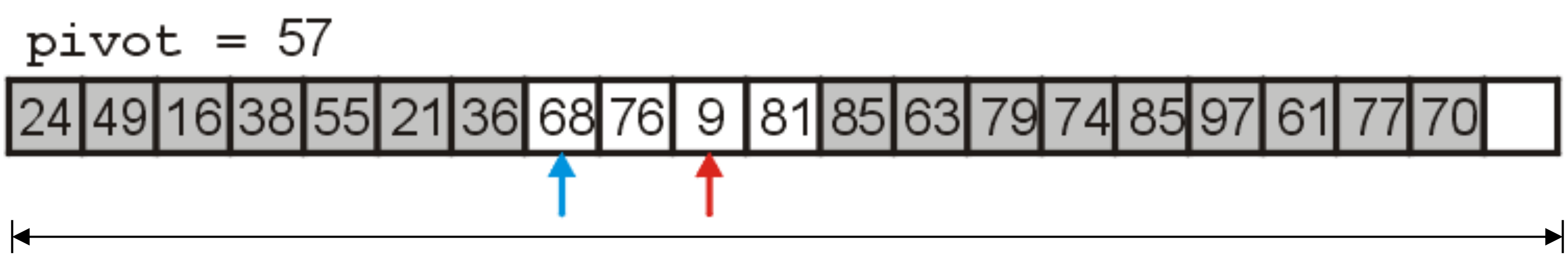
Improved Quick Sort Example

- **We swap 85 and 36, placing them in order with respect to each other**



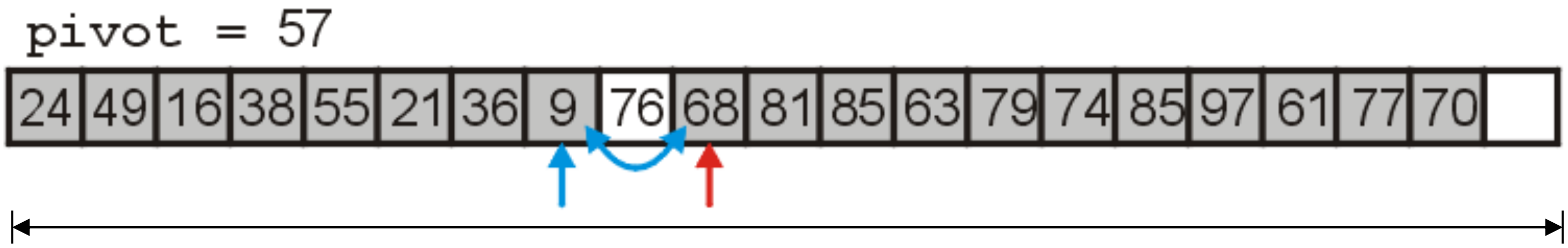
Improved Quick Sort Example

- We search forward until we find $68 > 57$
- We search backward until we find $9 < 57$



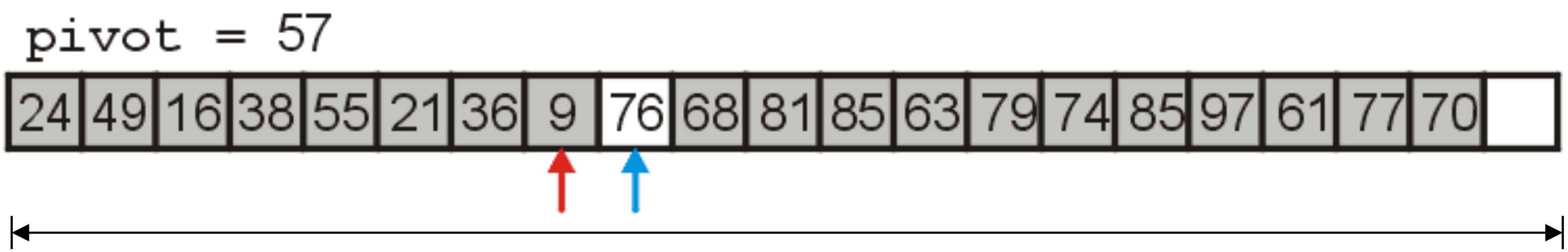
Improved Quick Sort Example

- **We swap 68 and 9**



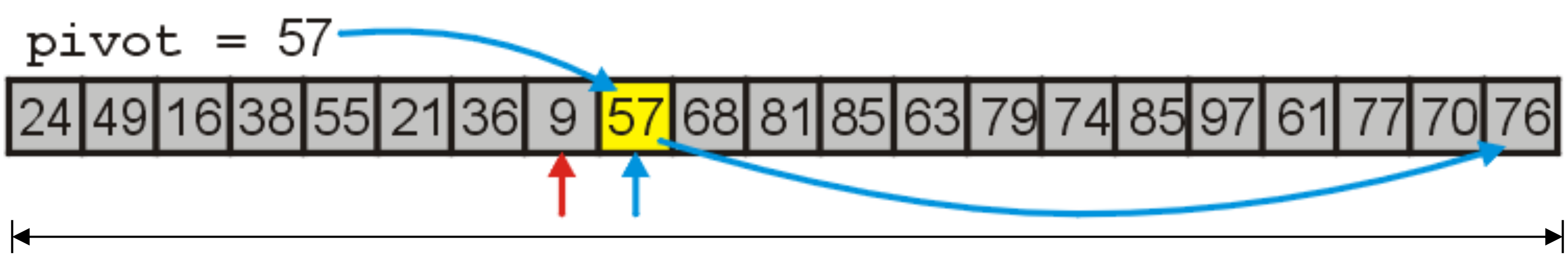
Improved Quick Sort Example

- We search forward until we find **76 > 57**
- We search backward until we find **9 < 57**
  - The indices are out of order, so we stop



Improved Quick Sort Example

- We move the larger indexed item to the vacancy at the end of the array
- We fill the empty location with the pivot, 57
- The pivot is now in the correct location



Improved Quick Sort Example

- We will now recursively call quick sort on the first half of the list
- When we are finished, all entries < 57 will be sorted

24	49	16	38	55	21	36	9	57	68	81	85	63	79	74	85	97	61	77	70	76

Improved Quick Sort Example

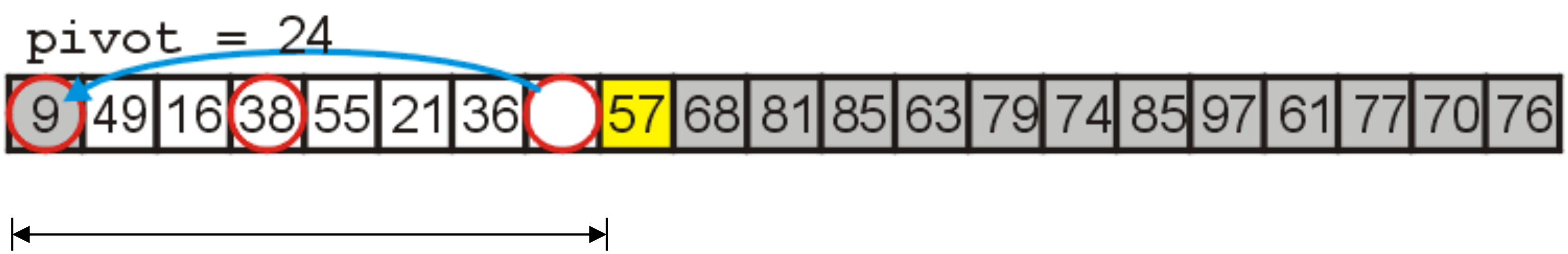
- We examine the first, middle, and last elements of this sub list

pivot =

24	49	16	38	55	21	36	9	57	68	81	85	63	79	74	85	97	61	77	70	76

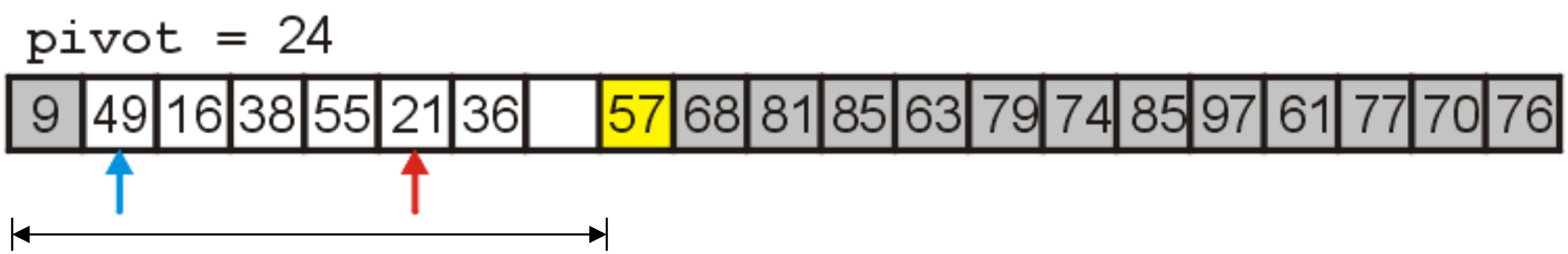
Improved Quick Sort Example

- We choose 24 to be our pivot
- We move 9 into the first location in this sub-list



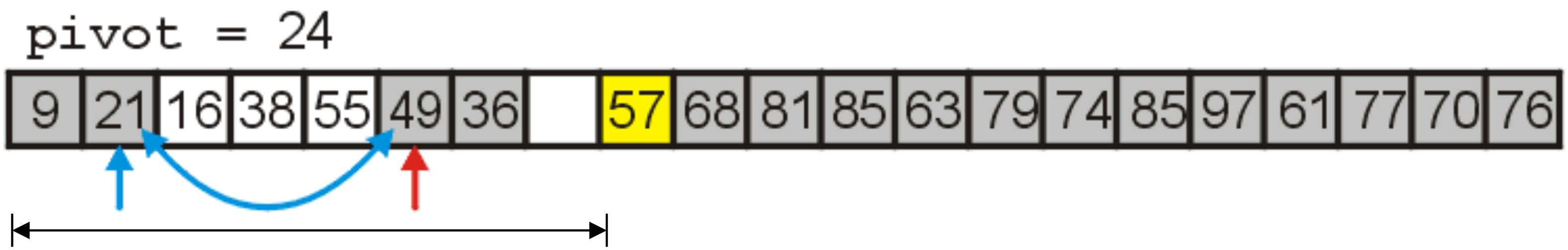
Improved Quick Sort Example

- We search forward till we find **49 > 24**
- We search backward till we find **21 < 24**



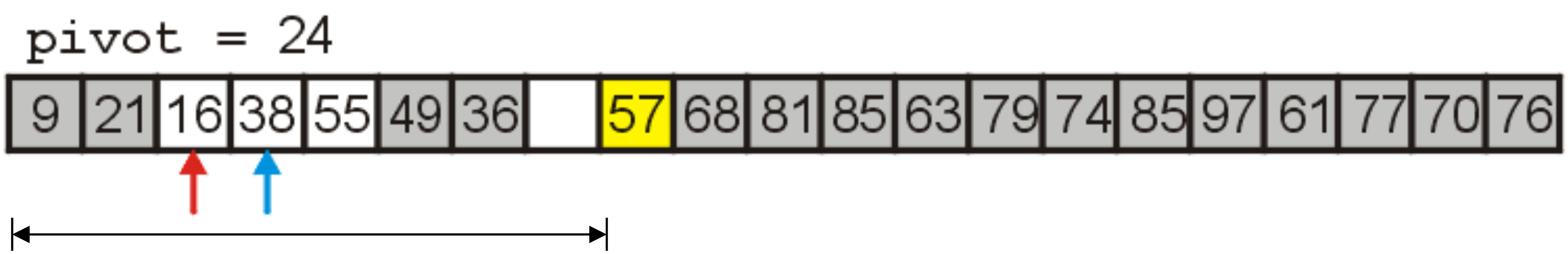
Improved Quick Sort Example

- **We swap 49 and 21, placing them in order with respect to each other**



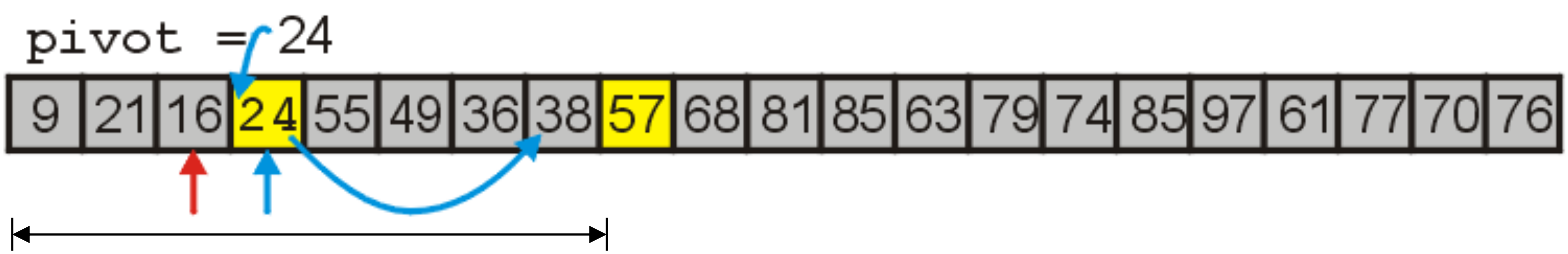
Improved Quick Sort Example

- We search forward till we find 38 > 24
- We search backward till we find 16 < 24
- The indices are reversed, so we stop



Improved Quick Sort Example

- We move 38 to the vacant location and move the pivot 24 into the previous location of 38
- 24 is now in the correct location



Improved Quick Sort Example

- **We will now recursively call quick sort on the left and right halves of those entries which are < 57**

9	21	16	24	55	49	36	38	57	68	81	85	63	79	74	85	97	61	77	70	76

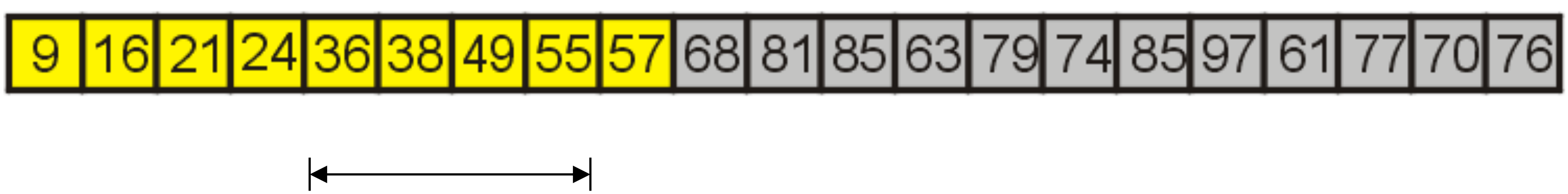
Improved Quick Sort Example

- The first partition has three entries, so we sort it using insertion sort

9	16	21	24	55	49	36	38	57	68	81	85	63	79	74	85	97	61	77	70	76

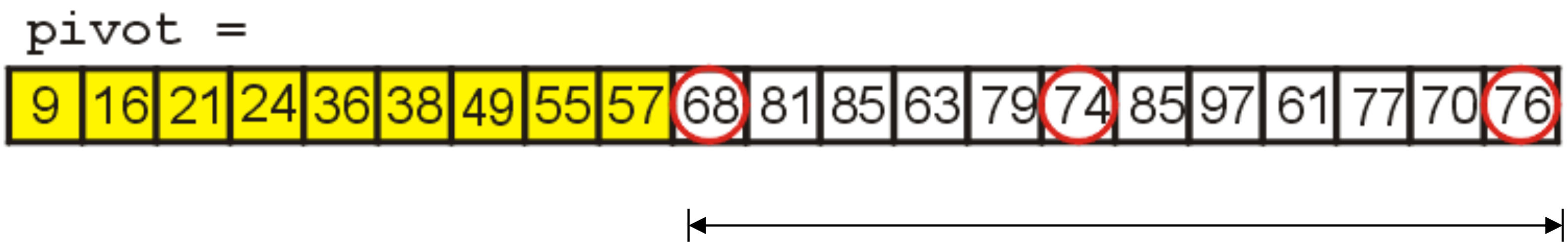
Improved Quick Sort Example

- **The second partition also has only four entries, so again, we use insertion sort**



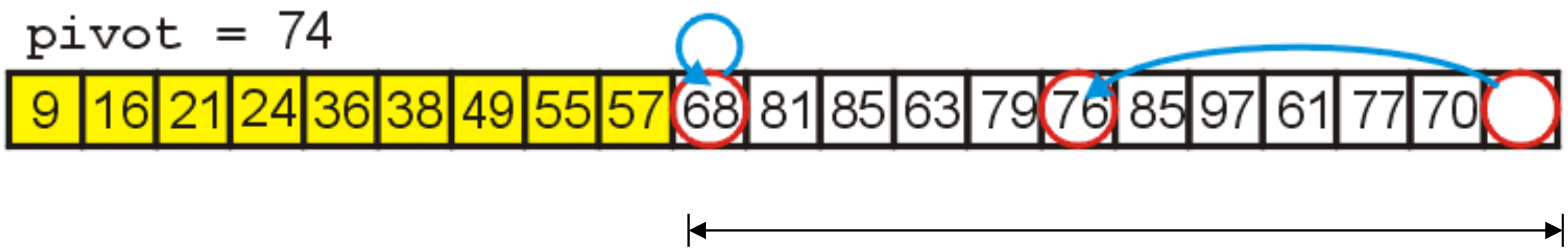
Improved Quick Sort Example

- **First we examine the first, middle, and last entries of the sub-list**



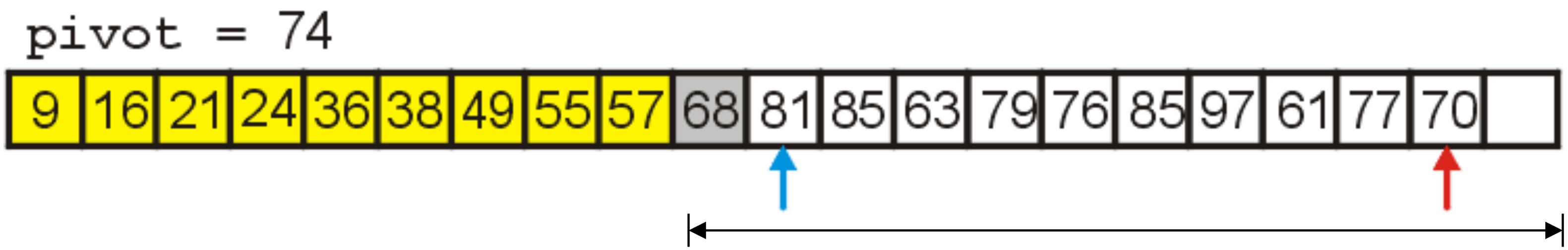
Improved Quick Sort Example

- We choose 74 to be our pivot
- We move 76 to the vacancy left by 74



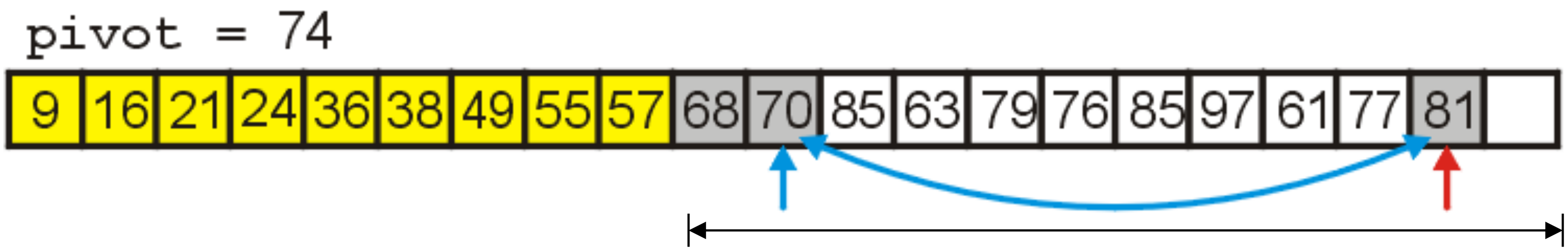
Improved Quick Sort Example

- **We search forward till we find** **81 > 74**
- **We search backward till we find** **70 < 74**



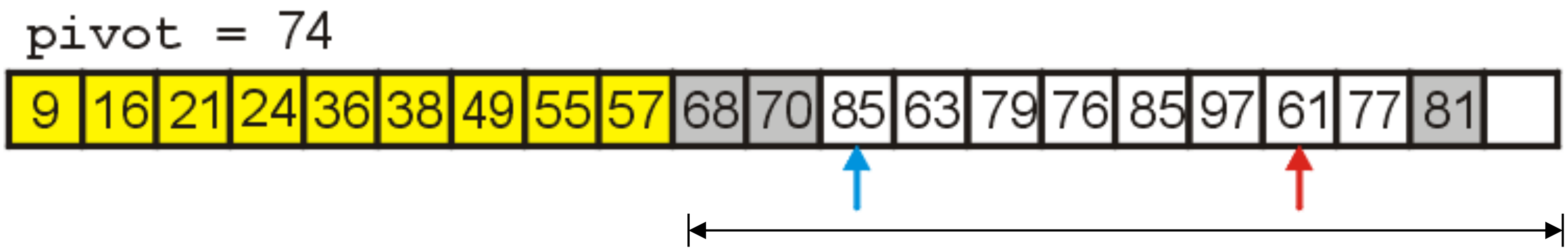
Improved Quick Sort Example

- We swap 70 and 84 placing them in order



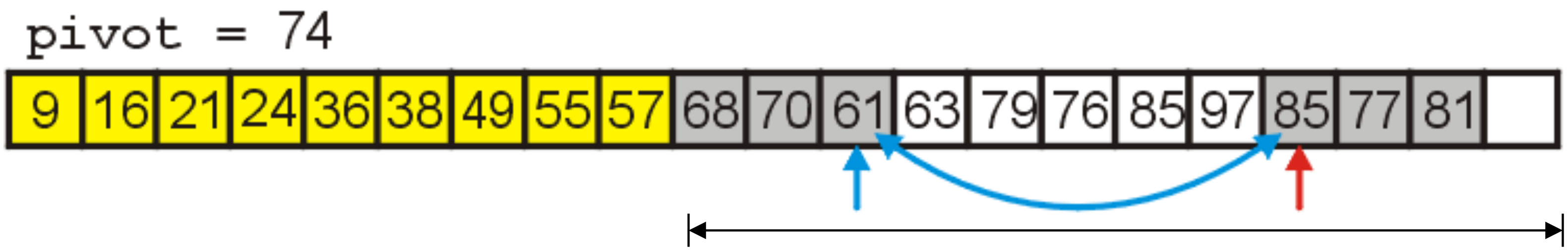
Improved Quick Sort Example

- We search forward till we find **85 > 74**
- We search backward till we find **61 < 74**



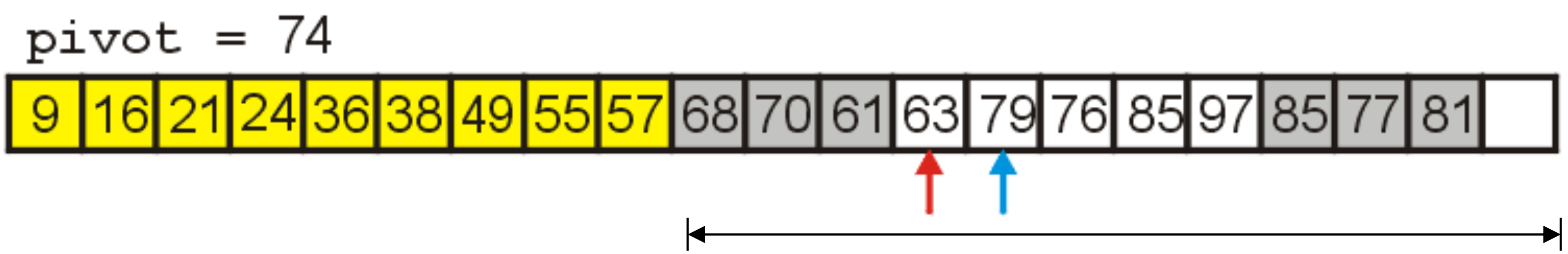
Improved Quick Sort Example

- We swap 85 and 61 placing them in order



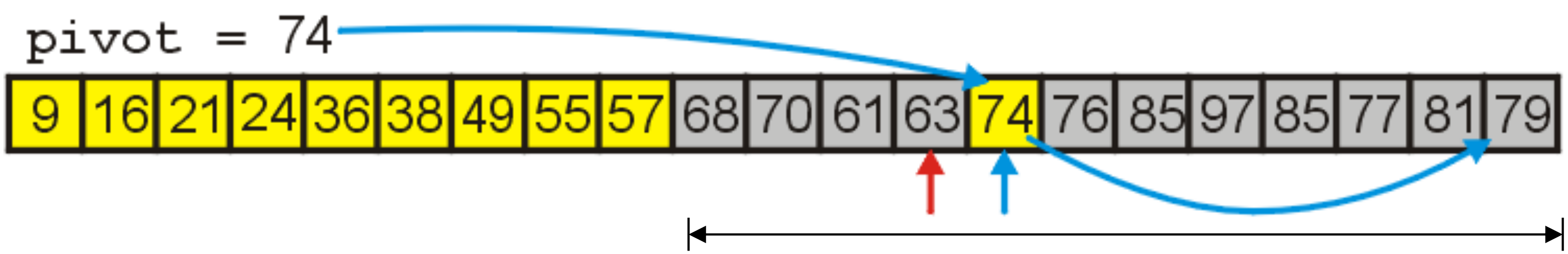
Improved Quick Sort Example

- We search forward till we find 79 > 74
- We search backward till we find 63 < 74
- The indices are reversed, so we stop



Improved Quick Sort Example

- We move 79 to the vacant location and move the pivot 74 into previous location of 79
- 74 is now in the correct location



Improved Quick Sort Example

- We sort the left sub-list first
- It has 4 elements, so we simply use insertion sort

9	16	21	24	36	38	49	55	57	68	70	61	63	74	76	85	97	85	77	81	79

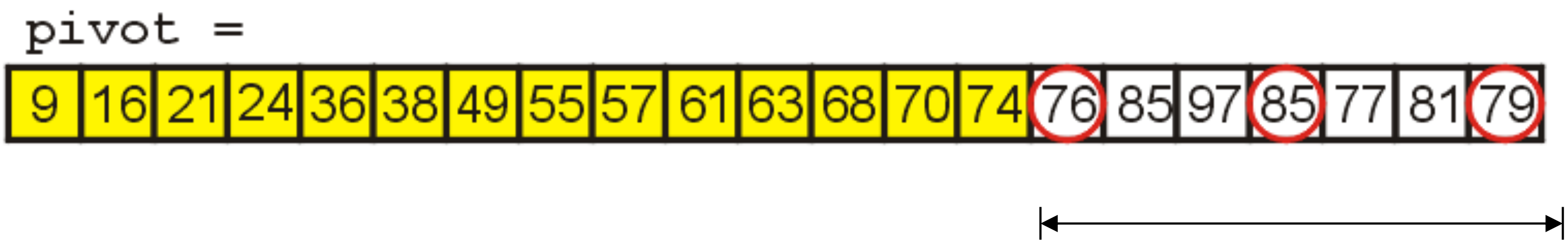
Improved Quick Sort Example

- **Having sorted the four elements, we focus on the remaining sub-list of seven entries**

9	16	21	24	36	38	49	55	57	61	63	68	70	74	76	85	97	85	77	81	79

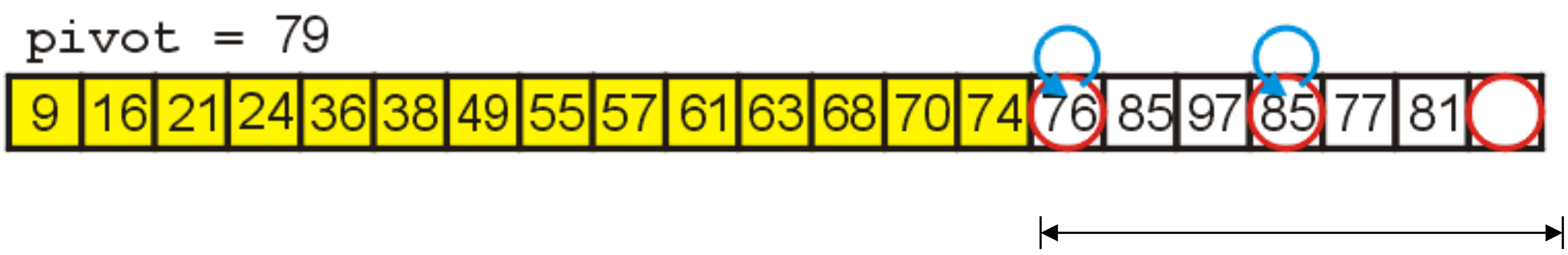
Improved Quick Sort Example

- To sort the next sub-list, we examine the first, middle, and last entries



Improved Quick Sort Example

- We select 79 as our pivot and move:
- 76 into the lowest position
- 85 into the highest position

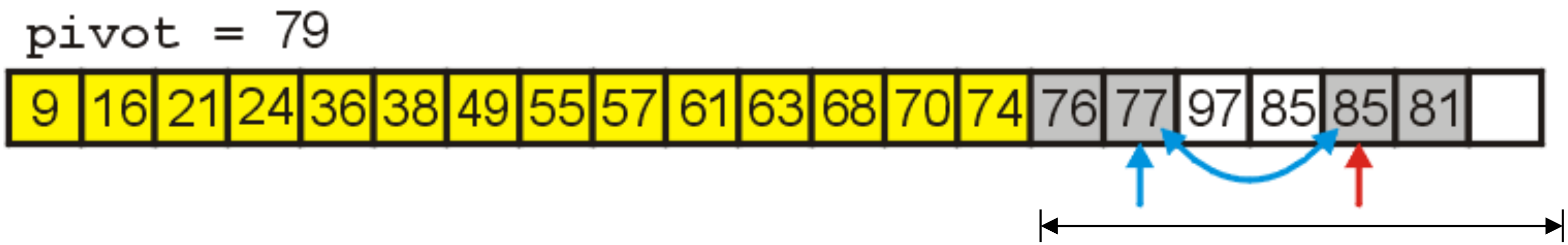


Improved Quick Sort Example

- **We search forward till we find** **$85 > 79$**
- **We search backward till we find** **$77 < 79$**

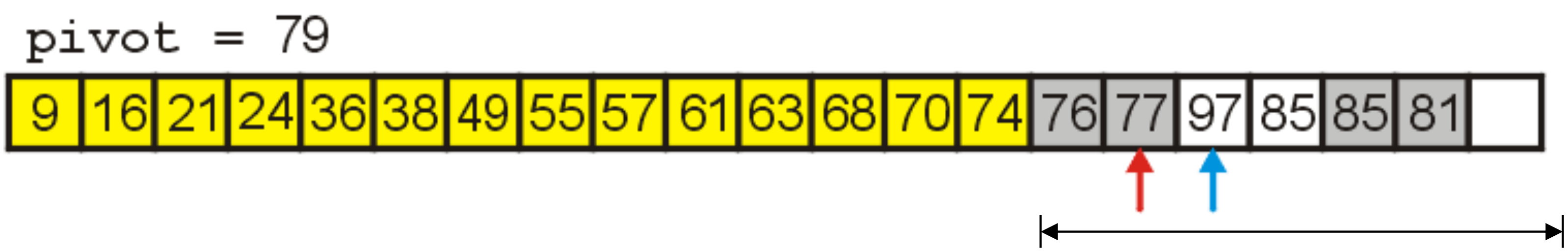
Improved Quick Sort Example

- We swap **85** and **77**, placing them in order



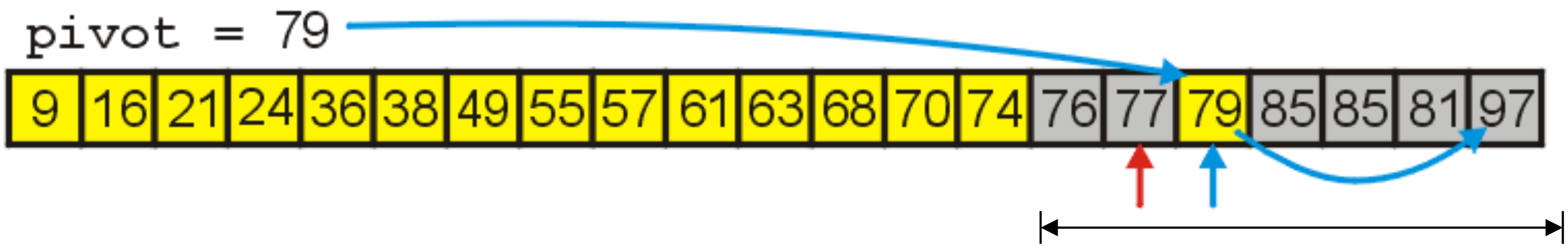
Improved Quick Sort Example

- **We search forward till we find** **97 > 79**
- **We search backward till we find** **77 < 79**
- **The indices are reversed, so we stop**



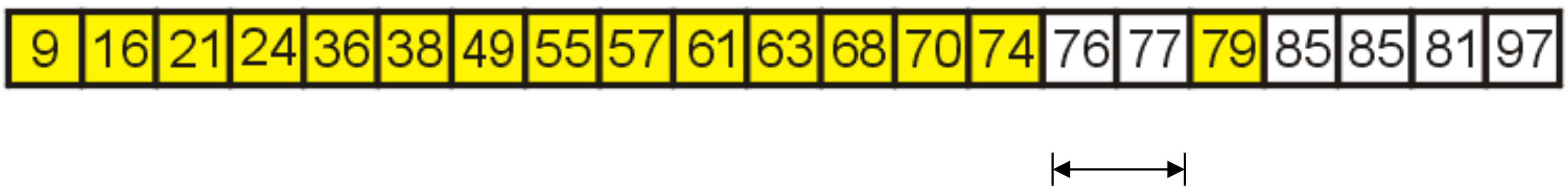
Improved Quick Sort Example

- Finally, we move 97 to the vacant location and copy 79 into the appropriate location
- 79 is now in the correct location



Improved Quick Sort Example

- This splits the sub-list into two sub-lists of size 2 and 4
- We use insertion sort for the first sub-list



Improved Quick Sort Example

- **We are left with one sub-list with four entries, so again, we use insertion sort**

9	16	21	24	36	38	49	55	57	61	63	68	70	74	76	77	79	85	85	81	97

Improved Quick Sort Example

- **Sorting the last sub-list, we arrive at an ordered list**

9	16	21	24	36	38	49	55	57	61	63	68	70	74	76	77	79	81	85	85	97

Run Time Summery

- To summarize all three O(n log n) algorithms

	Average Run Time	Worst-case Run Time	Average Memory	Worst-case Memory
Heap Sort	O(n log n)	O(n log n)	O(1)	O(1)
Merge Sort	O(n log n)	O(n log n)	O(n)	O(n)
Quick Sort	O(n log n)	O(n^2)	O(log n)	O(n)

《数据结构与算法》课程组

重庆大学计算机学院

End of Section.