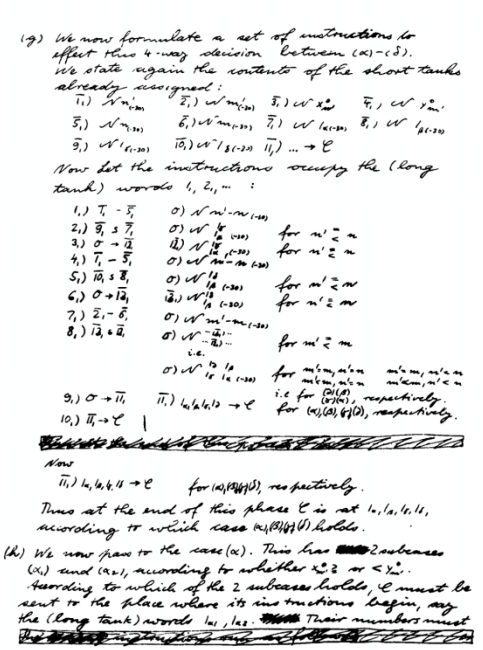
CS3334 Data Structures

Lecture 6: Quick Sort



Chee Wei Tan

Game of Nuts and Bolts

You have a mixed pile of N nuts and N bolts and need to quickly find the corresponding pairs of nuts and bolts.
Each nut matches exactly one bolt, and each bolt matches exactly one nut.
By fitting a nut and bolt together, you can see which is bigger.
But it is not possible to directly compare two nuts or two bolts.
Give an efficient method for solving the problem.

Quick Sort is Optimal

Sir Charles Antony Richard Hoare
Inventor of Quick Sort in 1959

Prof. Robert Sedgewick
His influential Ph.D. thesis in 1975 resolved open issues in Quick Sort

QUICKSORT

by Robert Sedgewick

Abstract

A complete study is presented of the best general purpose method for sorting by computer: C. A. R. Hoare's Quicksort algorithm. Special attention is paid to the methods of mathematical analysis which are used to demonstrate the practical utility of the algorithm. The most efficient known form of Quicksort is developed, and exact formulas are derived for the average, best case, and worst case running times. The merits of the many modifications which have been suggested to improve Quicksort are discussed, with an emphasis on their impact upon the analysis. Van Emden's method, samplesort, and the median-of-three modification are discussed in detail, and it is shown that the latter is the most effective improvement to Quicksort for practical sorting applications.

New results presented include: improvements to the algorithm based on a refined partitioning strategy and a new method of handling small subfiles, the best and worst case analysis, contrasting analyses of minor variants and the study of the effect of equal keys, new implementations of and new approaches to analyzing adaptive partitioning and samplesort, the complete general analysis of fixed sample size partitioning, and the application of "loop unwrapping" to Quicksort and the analysis of the optimized program.

The thesis is presented in an expository fashion so that it may be useful as a textbook in the field of "analysis of algorithms". It is self-contained, and it includes a complete treatment of a simpler sorting algorithm (insertion sorting) as well as three appendices which complement the material in the text.

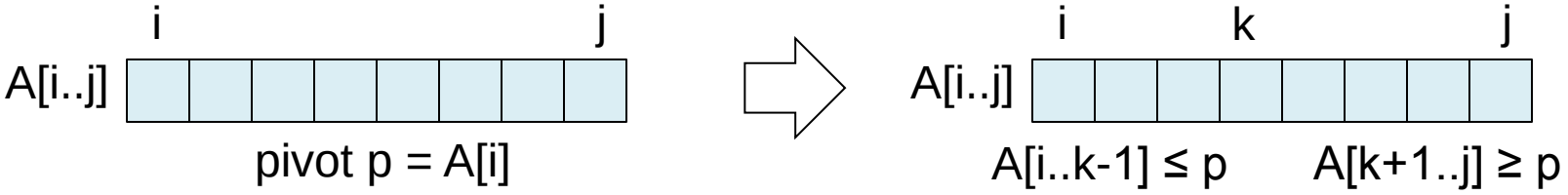
This research was supported in part by The Fannie and John Hertz Foundation. The printing of this paper was supported in part by National Science Foundation grant GJ 36473X and by the Office of Naval Research contract NR 044-402.

Quick Sort

- Imagine Insertion Sort that inserts items into a Binary Search Tree rather than within the list itself. This requires building the tree data structure outside the list.
- What if you can take advantage of characteristics of the Binary Search Tree and yet carry the sort within the original list?
- Best performance implied by entropy consideration

Quick Sort

- Idea (divide-&-conquer):
  - If size ≤ 1, return
  - Else
    - Pick an element in A[0..n-1] as pivot
    - Partition A[0..n-1] into 2 groups: those ≤ pivot and those ≥ pivot
    - Recursively sort each group



Quick Sort (Code)

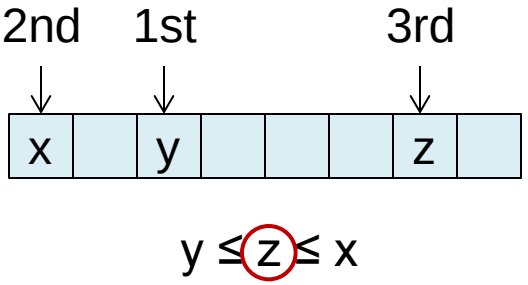
```
void quicksort(item A[], int i, int j)
{
   if (i <j)
   {
      int p = findpivot(A,i,j);  //Pick A[p] as pivot

      //Rearrange A[i..j] s.t. A[i..k] <= pivot
      //and A[k+1..j] >= pivot
      int k = partition(A,i,j,p);

      quicksort(A,i,k);
      quicksort(A,k+1,j);
   }
}
```

Quick Sort vs Merge Sort

- Merge sort:
  - Recursively sort two (approximately) equal halves
  - Then merge
- Quick sort:
  - Locally permute inputs (only use auxiliary small stack)
  - Recursively sort two (not necessarily equal) halves
- Is Quick sort faster than merge sort?
  - Quick sort takes $O(n^2)$ time in the worst case
  - On average quick sort takes O(nlogn) time but *no need for extra buffer*
  - Substantially faster in practice

How to Pick the Pivot?

- **[CLRS] Pick the 1st element (we use this)**
  - Worst case when presorted or reversely sorted
- Pick a random element
  - Need pseudo-random number generator, very expensive
- [Weiss] Pick median of 3



- Pick larger of first 2 distinct elements

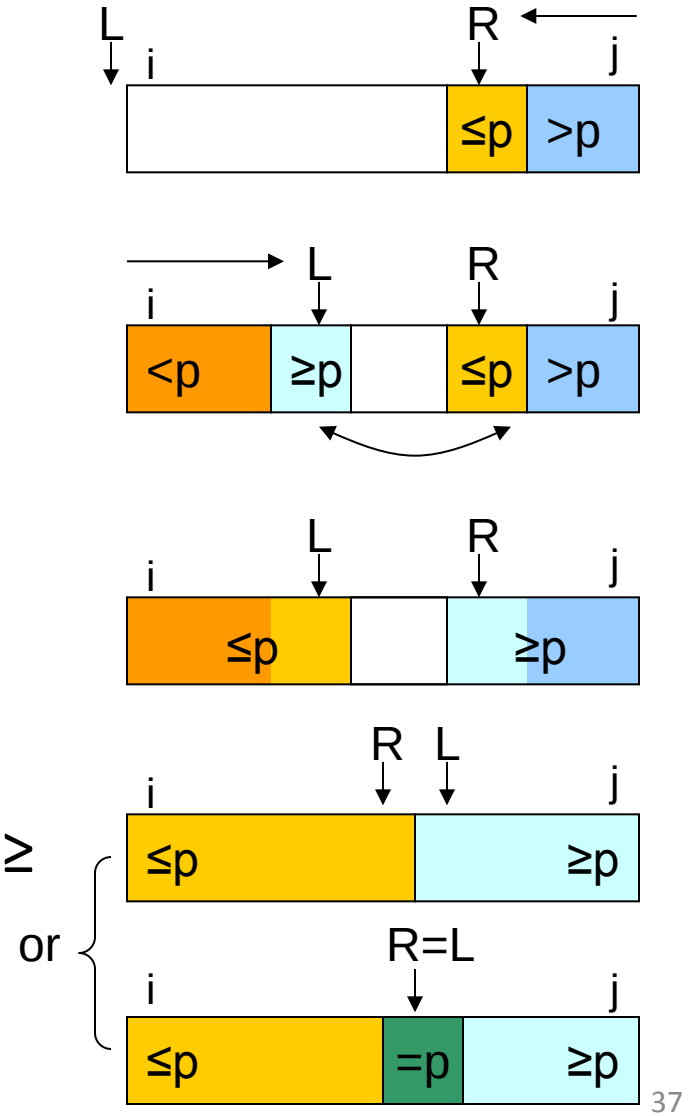
How to Partition?

1) Move R from right to left until A[R] ≤ pivot

2) Move L from left to right until A[L] ≥ pivot

3) If L < R, swap A[L] and A[R], repeat (1) – (3) until L ≥ R

4) If L ≥ R, A[i..R] ≤ pivot and A[L..j] ≥ pivot → Return R

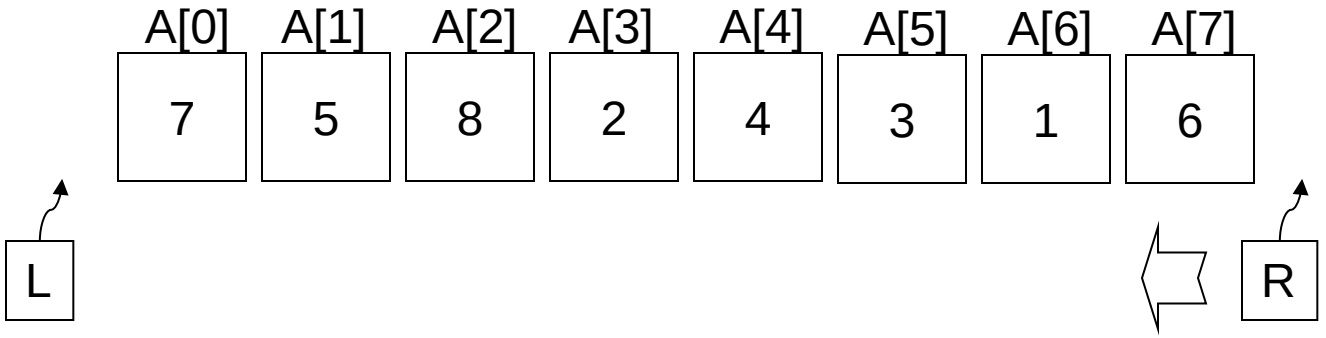


Partition

```
int findpivot(item A[], int i, int j)
{
   return i;     // pick 1st item as pivot
}

int partition(item A[], int i, int j, int p)
{
   item pivot=A[p];
   int L=i-1, R=j+1;
   do {
      do R--; while (A[R]>pivot);
      do L++; while (A[L]<pivot);
      if (L < R)
         swap(A[L],A[R]);
   } while (L < R);
   return R;   // lower boundary of right segment
}
```

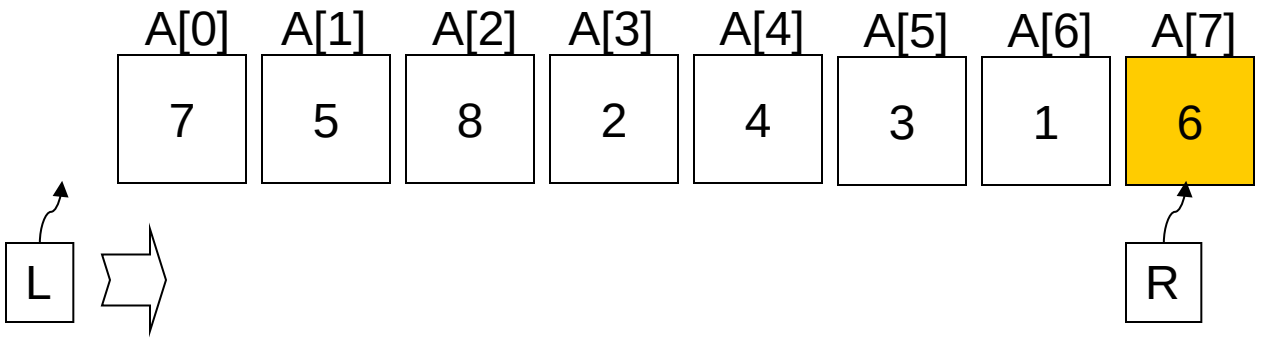
Partition (1/14)

pivot=7; i=0; j=7



Partition (2/14)

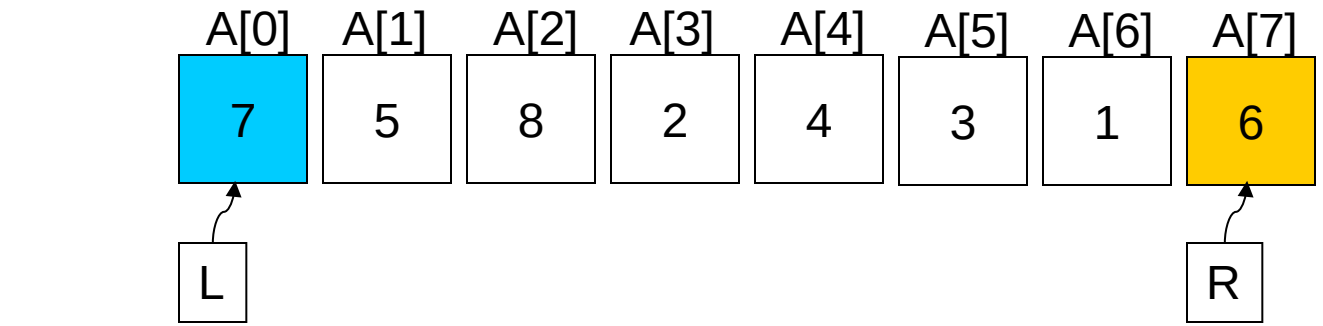
pivot=7; i=0; j=7



since A[R]≤7, stop

Partition (3/14)

pivot=7; i=0; j=7



since A[L]≥7, stop

since A[R]≤7, stop

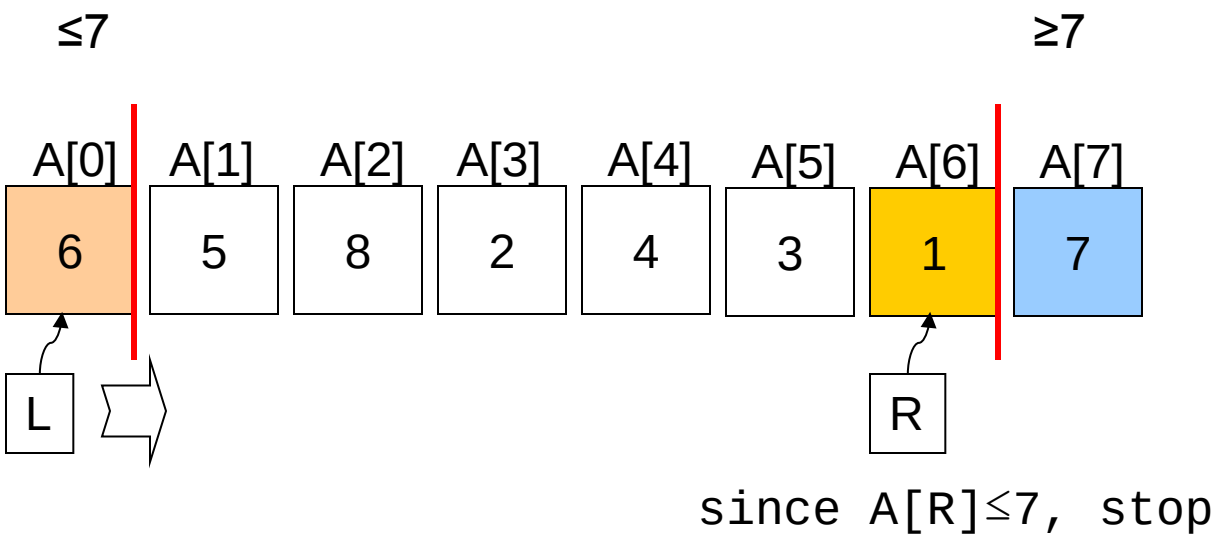
Partition (4/14)

pivot=7; i=0; j=7

since L < R, swap A[L] and A[R]

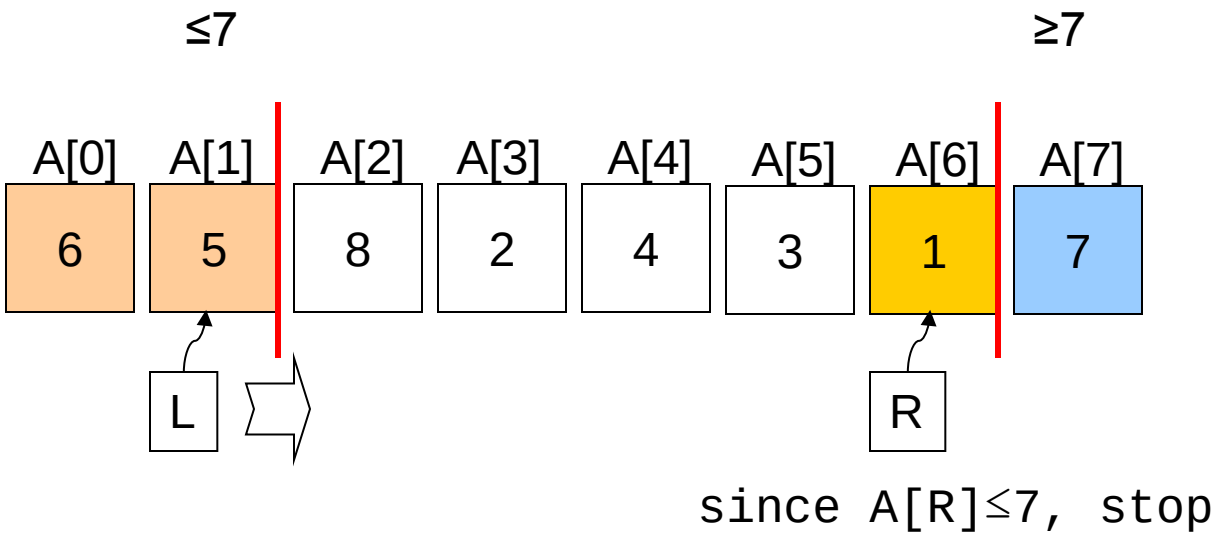
Partition (5/14)

pivot=7; i=0; j=7



Partition (6/14)

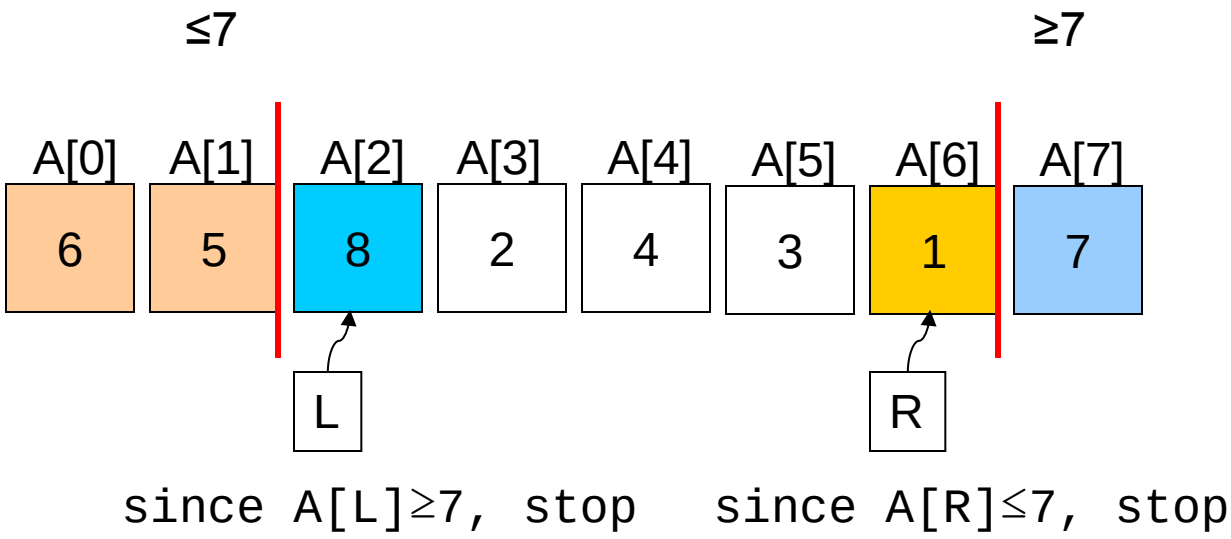
pivot=7; i=0; j=7



since A[R]≤7, stop

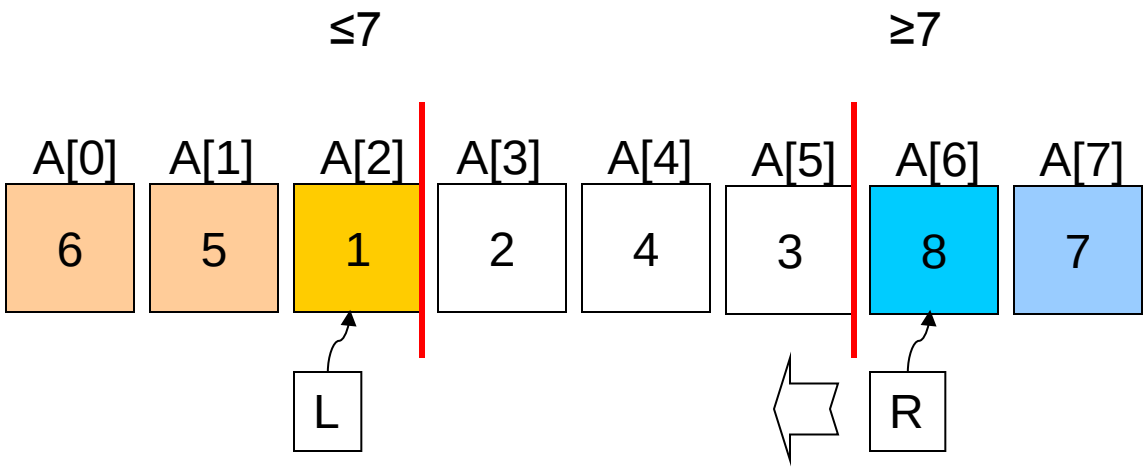
Partition (7/14)

pivot=7; i=0; j=7



Partition (8/14)

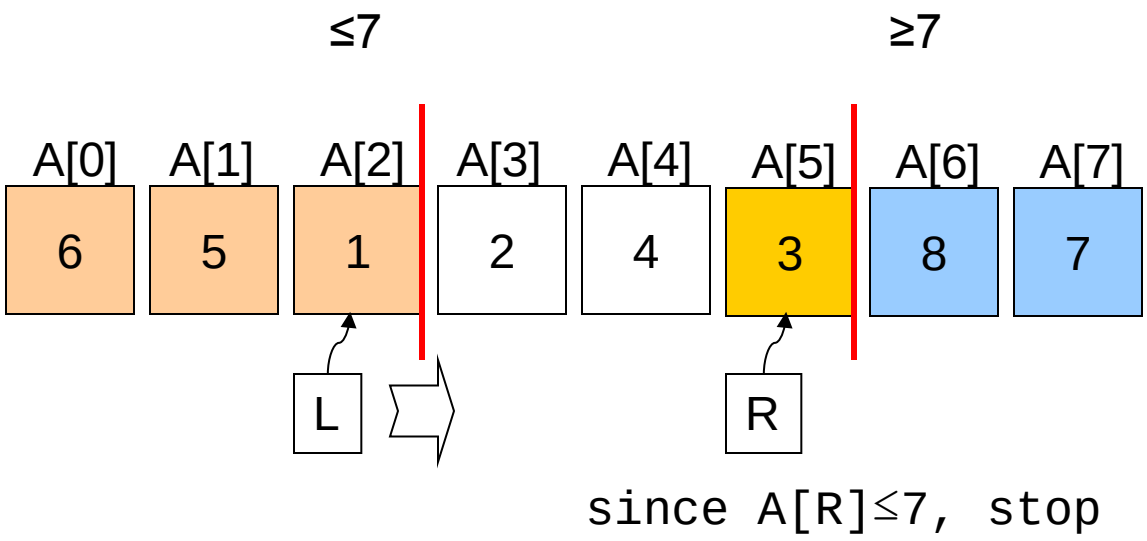
pivot=7; i=0; j=7



since L < R, swap A[L] and A[R]

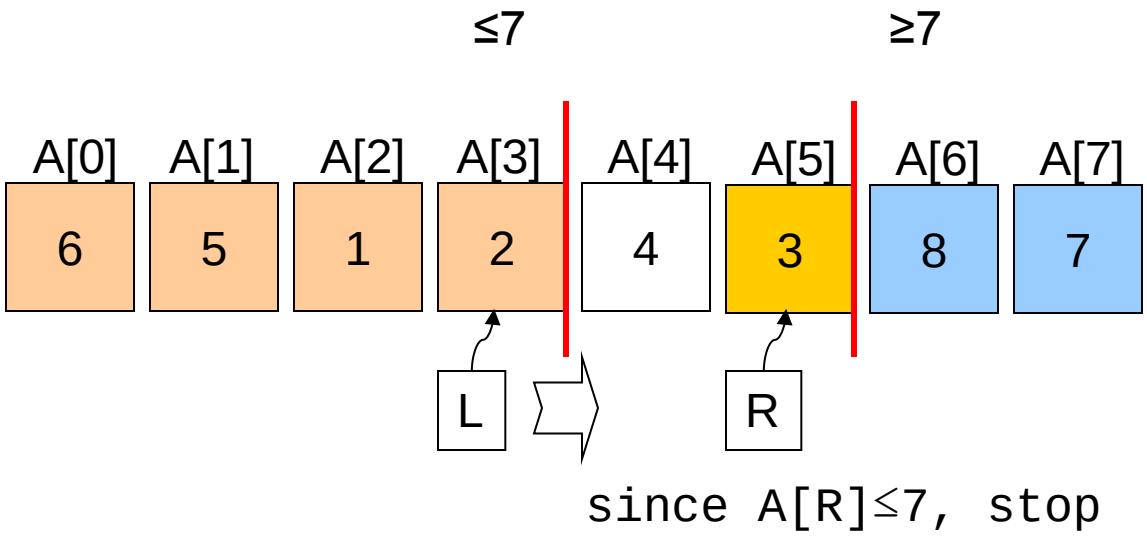
Partition (9/14)

pivot=7; i=0; j=7



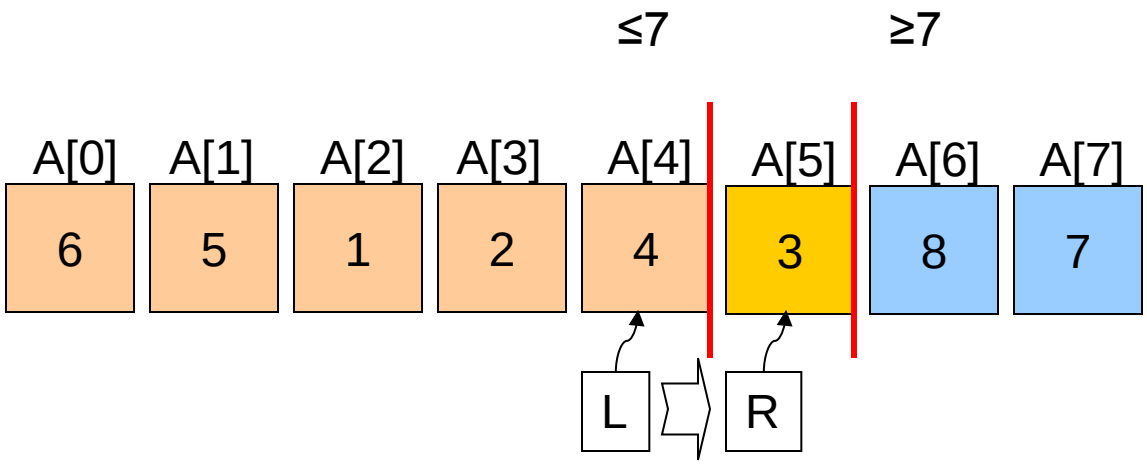
Partition (10/14)

pivot=7; i=0; j=7



Partition (11/14)

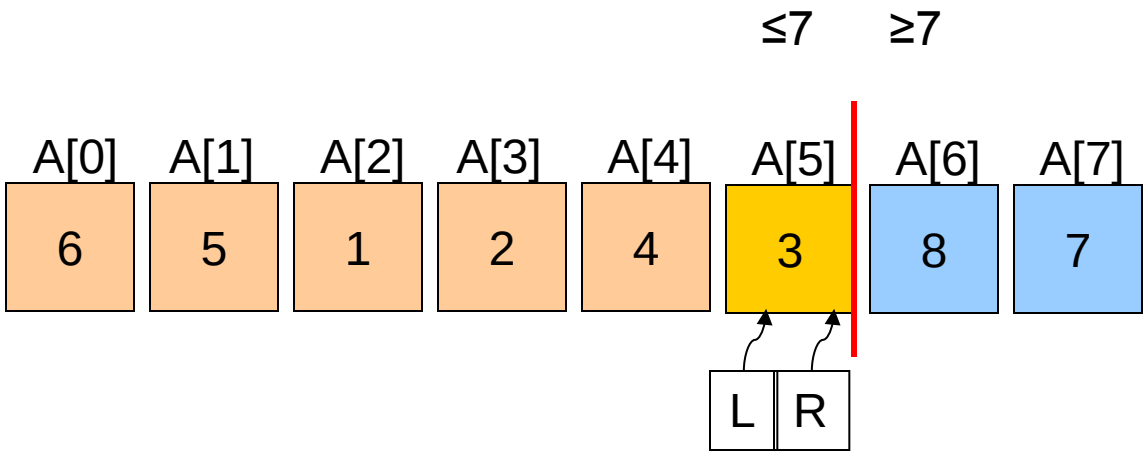
pivot=7; i=0; j=7



since A[R]≤7, stop

Partition (12/14)

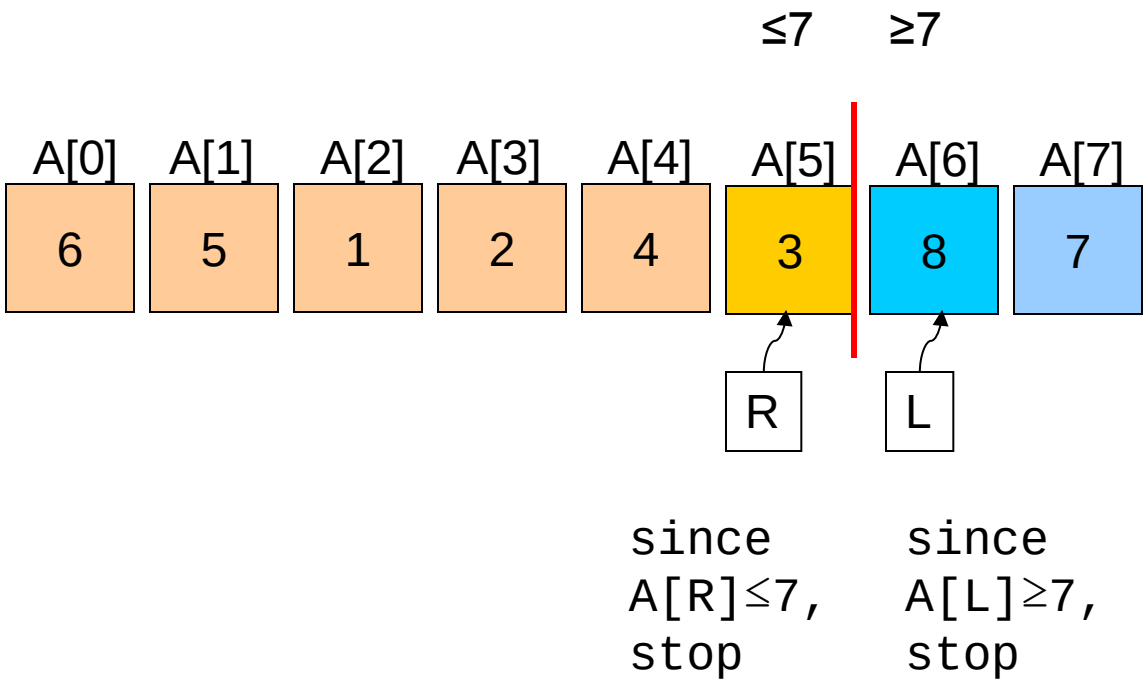
pivot=7; i=0; j=7



since A[R]≤7, stop

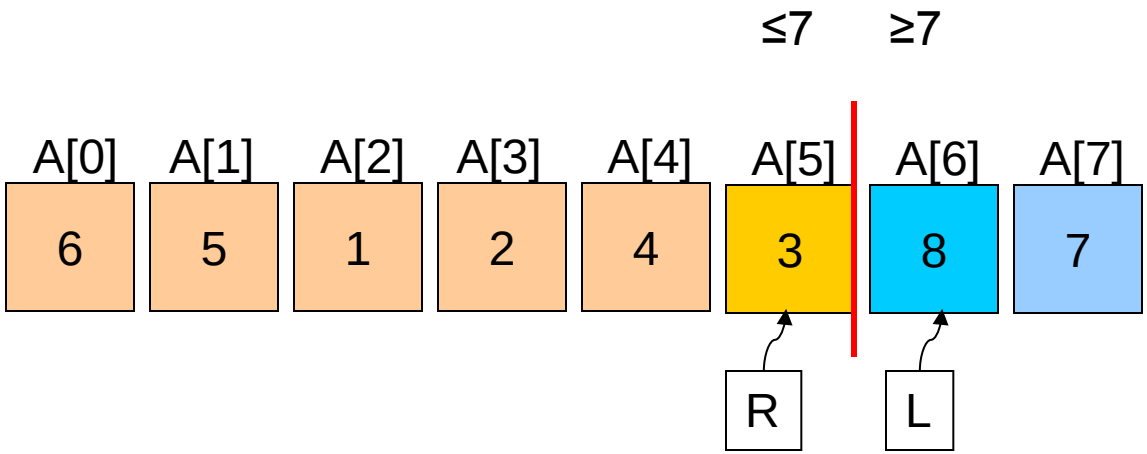
Partition (13/14)

pivot=7; i=0; j=7



Partition (14/14)

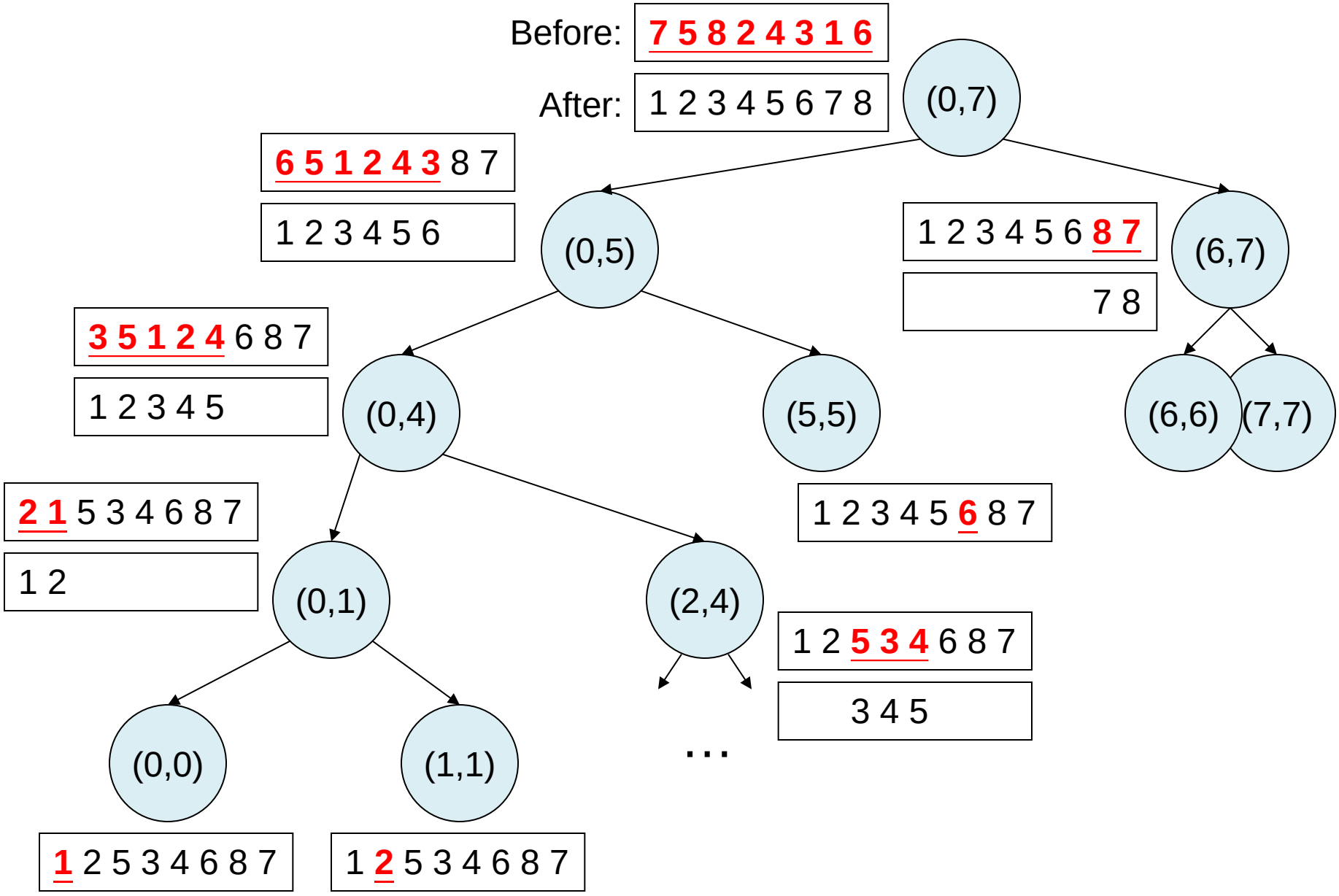
pivot=7; i=0; j=7; k=5



since L ≥ R, no swapping and partition completed

sort A[i..R] (i.e., A[0..5]) &
A[L..j] (i.e., A[6..7]) recursively

An illustration



Worst Case Analysis

- $T(0), T(1) \leq c_1$, for some constant c_1
- Local work: if-statement, findpivot, partition
  - At most cn time
- Recursive calls: T(i) and T(n-i)
- Worst case: one of the segments is singleton

$T(n) \leq T(n-1) + c'n$ (where $c' \geq c + c_1$)

$T(n-1) \leq T(n-2) + c'(n-1)$

$T(n-2) \leq T(n-3) + c'(n-2)$

…

+) $T(2) \leq T(1) + c'(2)$

$T(n) \leq T(1) + c'(2+\ldots+n)$

$= T(1) + c'(2+n)(n-1)/2$

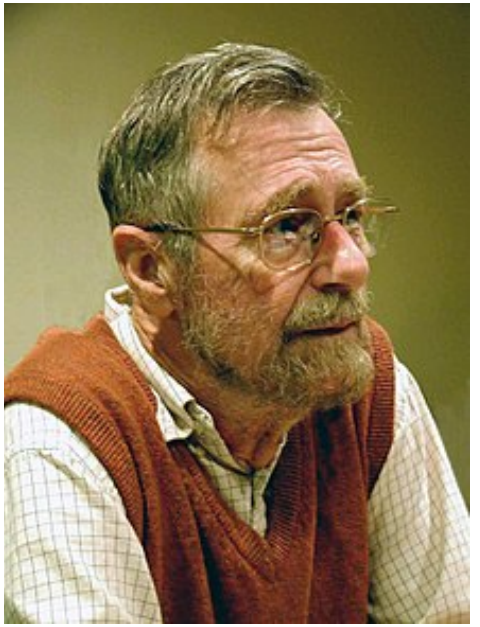
$= O(n^2)$

Average Case Analysis

- Same model as in Lec 4:
  - Assume input array is a permutation of {1,…, n}
  - For a fixed n, there are n! possible inputs
  - "Average case time complexity" means "average running time of these n! inputs"
- Let $T_A(n)$ = average case time complexity of quick sort
- Theorem: $T_A(n) = O(n\log n)$
- Advanced proof involving concepts of entropy and probability (covered in more advanced algorithm courses).

The Dutch National Flag Problem

The flag of the Netherlands consists of three colors: red, white and blue. Given balls of these three colors arranged randomly in a line (the actual number of balls does not matter), arrange the balls such that all balls of the same color are together and their collective color groups are in the correct order.



Edsger W. Dijkstra
One of the most influential figures of computing science's founding generation, Dijkstra was a theoretical physicist whose career was a computer programmer. His ideas lay the foundations for the birth and development of the professional discipline of software engineering. And he gave his name to one of the most famous algorithms in graph theory.