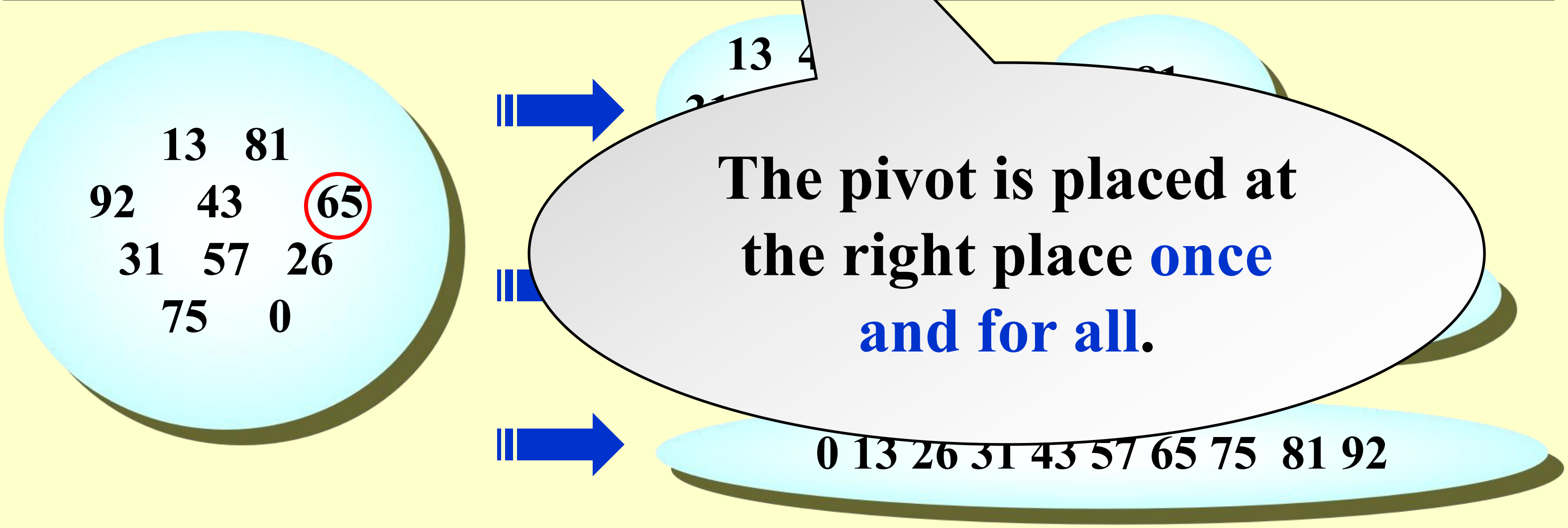
§ 7 Quicksort -- the **fastest** known sorting algorithm in practice

1. The Algorithm

```
void Quicksort ( ElementType A[ ], int N )
{
    if ( N < 2 )  return;
  ? pivot = pick any element in A[ ];
  ? Partition S = { A[ ] \ pivot } into two disjoint sets:
          A1={ a∈S | a ≤ pivot } and A2={ a∈S | a ≥ pivot };
    A = Quicksort ( A1, N1) ∪ { pivot } ∪ Quicksort ( A2, N2);
}
```

The best case $T(N) = O(\ N \log N$



2. Picking the Pivot

☞ **A Wrong Way:** **Pivot = A[0]**

The worst case: **A[ ] is presorted – quicksort will take $O(N^2)$ time to do nothing** ☹

☞ **A Safe Maneuver:** **Pivot = random select from A[ ]**

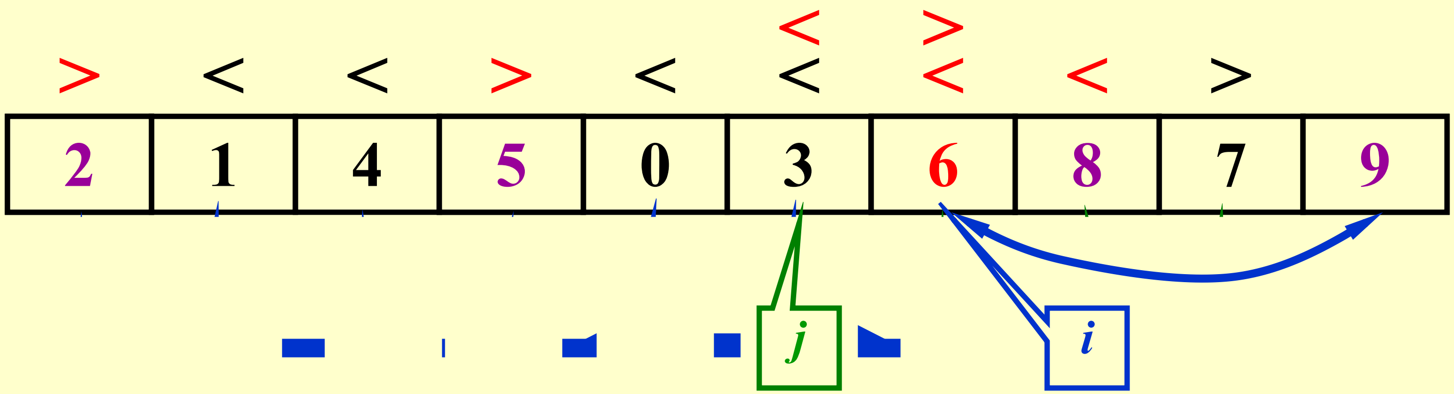
☹ **random number generation is expensive**

☞ **Median-of-Three Partitioning:**

Pivot = median (left, center, right)

Eliminates the bad case for sorted input and actually reduces the running time by about 5%.

3. Partitioning Strategy



Then $T(N) = O(N^2)$.
So we'd better stop both i and j
and take some extra swaps.

4. Small Arrays

Problem: Quicksort is slower than insertion sort for small N (≤ 20).

Solution: Cutoff when N gets small (e.g. $N = 10$) and use other efficient algorithms (such as insertion sort).

5. Implementation

```
void  Quicksort( ElementType A[ ], int N )
{
        Qsort( A, 0, N - 1 );
        /* A:      the array      */
        /* 0:      Left index     */
        /* N – 1: Right index     */
}
```

```
/* Return median of Left, Center, and Right */
/* Order these and hide the pivot */

ElementType Median3( ElementType A[ ], int Left, int Right )
{
    int  Center = ( Left + Right ) / 2;
    if ( A[ Left ] > A[ Center ] )
        Swap( &A[ Left ], &A[ Center ] );
    if ( A[ Left ] > A[ Right ] )
        Swap( &A[ Left ], &A[ Right ] );
    if ( A[ Center ] > A[ Right ] )
        Swap( &A[ Center ], &A[ Right ] );
    /* Invariant: A[ Left ] <= A[ Center ] <= A[ Right ] */
    Swap( &A[ Center ], &A[ Right - 1 ] ); /* Hide pivot */
    /* only need to sort A[ Left + 1 ] … A[ Right – 2 ] */
    return  A[ Right - 1 ];  /* Return pivot */
}
```

```
void  Qsort( ElementType A[ ], int Left, int Right )
{   int  i,  j;
    ElementType  Pivot;
    if ( Left + Cutoff <= Right ) {  /* if the sequence is not too short */
        Pivot = Median3( A, Left, Right );  /* select pivot */
        i = Left;     j = Right – 1;  /* why not set Left+1 and Right-2? */
        for( ; ; ) {
            while ( A[ + +i ] < Pivot ) { }  /* scan from left */
            while ( A[ – –j ] > Pivot ) { }  /* scan from right */
            if ( i < j )
               Swap( &A[ i ], &A[ j ] );  /* adjust partition */
            else     break;  /* partition done */
        }
        Swap( &A[ i ], &A[ Right - 1 ] ); /* restore pivot */
        Qsort( A, Left, i - 1 );      /* recursively sort left part */
        Qsort( A, i + 1, Right );    /* recursively sort right part */
    }  /* end if - the sequence is long */
    else /* do an insertion sort on the short subarray */
        InsertionSort( A + Left, Right - Left + 1 );
}
```

6. Analysis

$$T(N) = T(i) + T(N - i - 1) + c\,N$$

☞ **The Worst Case:**

$$T(N) = T(N - 1) + c\,N \;\Rightarrow\; T(N) = O(N^2)$$

☞ **The Best Case:** [… …] • [… …]

$T(N) = 2T(N/2) + c\,N$ … $\log N$)

☞ **The Average Case:**

Assume the average valu …

$$T(N) = \frac{2}{N}\left[\sum_{j=0}^{N-1} T(j)\right] + c \;\Rightarrow\; T(N) = O(N \log N)$$

Read Figure 6.16 on p.214 for the 5th algorithm on solving this problem.

〖Example〗 **Given a list of N elements and an integer k. Find the kth largest element.**

§ 8 Sorting Large Structures

Problem: Swapping large structures can be very much expensive.

Solution: Add a pointer field to the structure and swap pointers instead – **indirect sorting**. Physically rearrange the structures at last if it is really necessary.

〖Example〗 Table Sort

list	[0]	[1]	[2]	[3]	[4]	[5]
key	d	b	f	c	a	e
table	4	1	3	0	5	2

The sorted list is

list [table[0]], list [table[1]], ……, list [table[n–1]]

Note: Every permutation is made up of disjoint cycles.

list	[0]	[1]	[2]	[3]	[4]	[5]
key	a	b	c	d	e	f
table	0	1	2	3	4	5

temp = d
current = 3
next = 3

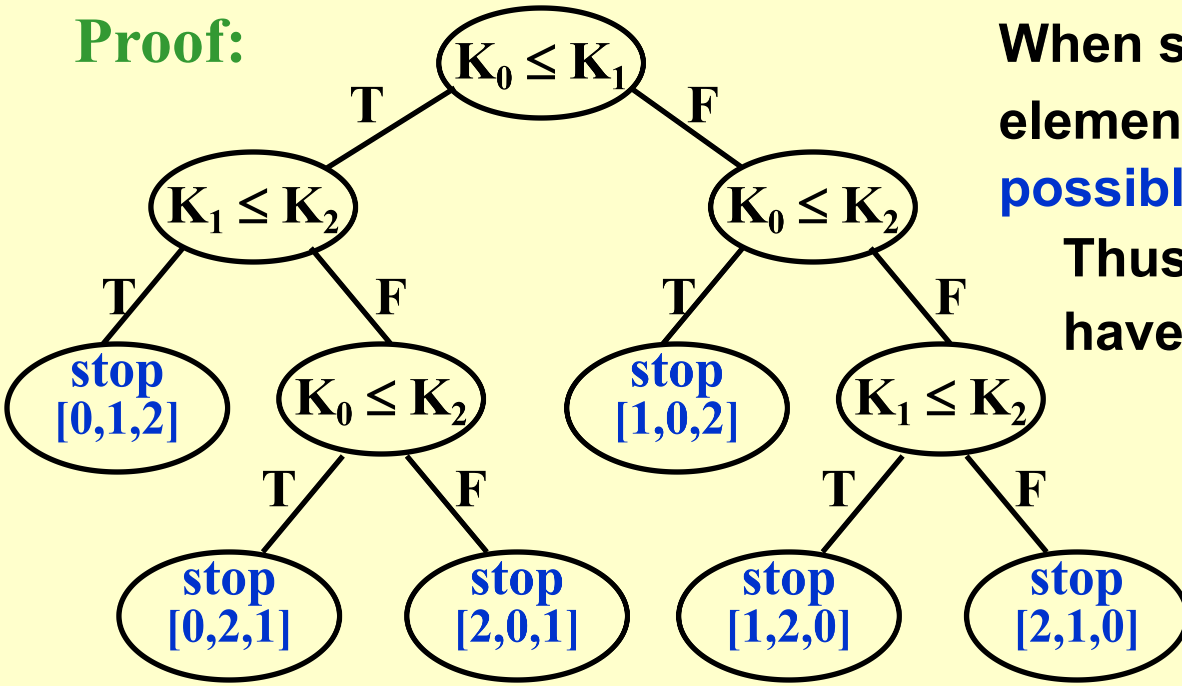
In the worst case there are $\lfloor N / 2 \rfloor$ cycles and requires $\lfloor 3N / 2 \rfloor$ record moves.

$T = O(\, m\, N\,)$ where m is the size of a structure.

§ 9 A General Lower Bound for Sorting

【Theorem】 Any algorithm that **sorts by comparisons only** must have a worst case computing time of $\Omega(N \log N)$.

Proof:



Decision tree for insertion sort on R_0, R_1, and R_2

When sorting N distinct elements, there are $N!$ **different possible results**.

Thus any decision tree must have at least $N!$ **leaves**.

If the height of the tree is k, then $N! \le 2^{k-1}$ (# of leaves in a complete binary tree)

$\Rightarrow \quad k \ge \log(N!) + 1$

Since $N! \ge (N/2)^{N/2}$ and $\log_2 N! \ge (N/2)\log_2(N/2) = \Theta (N \log_2 N)$

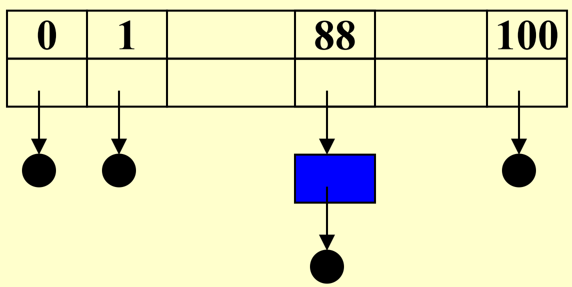
Therefore $T(N) = k \ge c \cdot N \log_2 N$. ■

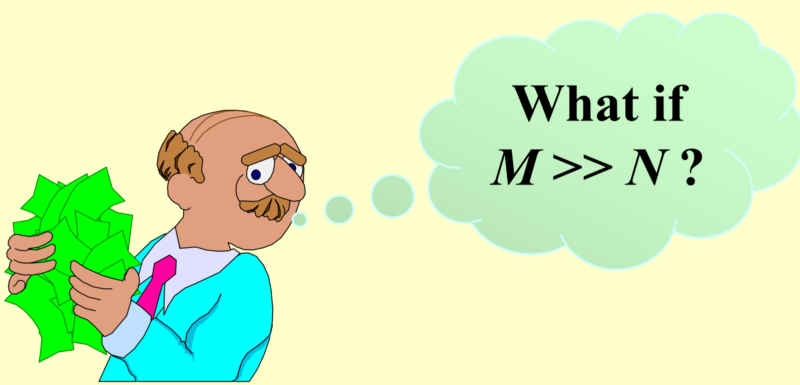
§ 10 Bucket Sort and Radix Sort

☞ Bucket Sort

〖Example〗 Suppose that we have N students, each has a grade record in the range 0 to 100 (thus there are M = 101 possible distinct grades). How to sort them according to their grades in linear time?

count





Algorithm

```
{
    initialize count[ ];
    while (read in a student's record)
        insert to list count[stdnt.grade];
    for (i=0; i<M; i++) {
        if (count[i])
            output list count[i];
    }
}
```

$T(N, M) = O(M+N)$

〖Example〗 Given N = 10 integers in the range 0 to 999 (M = 1000)
Is it possible to sort them in linear time?

What if we sort according to the Most Significant Digit first?

☞ Radix Sort

Input: 64, 8, 216, 512, 27, 729, 0, 1, 343, 125

Sort according to the Least Significant Digit first.

Bucket	0	1	2	3	4	5	6	7	8	9
Pass 1	0	1	512	343	64	125	216	27	8	729
Pass 2	0	512	125		343		64			
	1	216	27							
	8		729							
Pass 3	0	125	216	343		512		729		
	1									
	8									
	27									
	64									

$T = O(P(N+B))$ where P is the number of passes, N is the number of elements to sort, and B is the number of buckets.

Output: 0, 1, 8, 27, 64, 125, 216, 343, 512, 729

Suppose that the record R_i has r keys.

- $K_i^{\,j}$ ::= the j-th key of record R_i
- $K_i^{\,0}$ ::= the **most** significant key of record R_i
- $K_i^{\,r-1}$ ::= the **least** significant key of record R_i
- A list of records $R_0, \ldots, R_{n-1}$ is **lexically sorted** with respect to the keys $K^0, K^1, \ldots, K^{r-1}$ iff

$$(K_i^0, K_i^1, \cdots, K_i^{r-1}) \le (K_{i+1}^0, K_{i+1}^1, \cdots, K_{i+1}^{r-1}),\ \ 0 \le i < n-1.$$

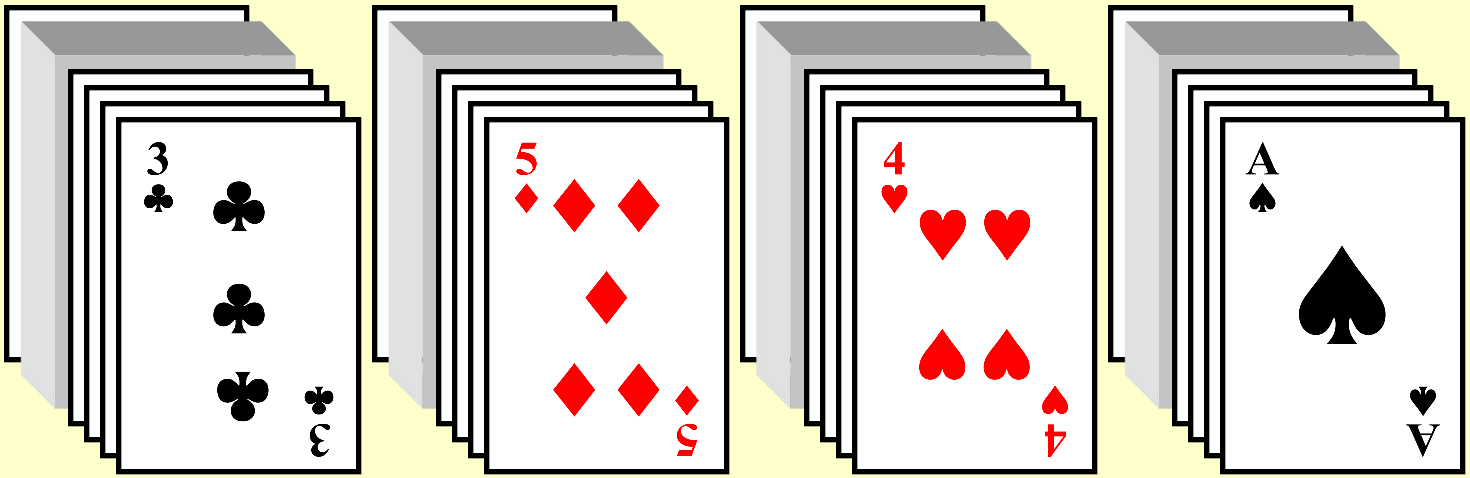
That is, $K_i^{\,0} = K_{i+1}^{\,0}$, ... , $K_i^{\,l} = K_{i+1}^{\,l}$, $K_i^{\,l+1} < K_{i+1}^{\,l+1}$ for some $l < r - 1$.

〖Example〗 A deck of cards sorted on 2 keys

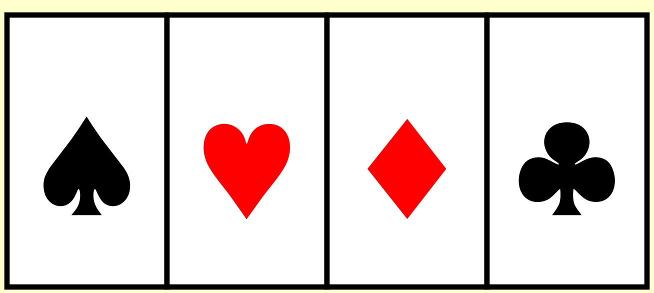
K^0 [Suit]	♣ < ♦ < ♥ < ♠
K^1 [Face value]	2 < 3 < 4 < 5 < 6 < 7 < 8 < 9 < 10 < J < Q < K < A
Sorting result :	2♣ ... A♣ 2♦ ... A♦ 2♥ ... A♥ 2♠ ... A♠

☞ MSD (Most Significant Digit) Sort

① Sort on K^0: for example, create 4 buckets for the suits

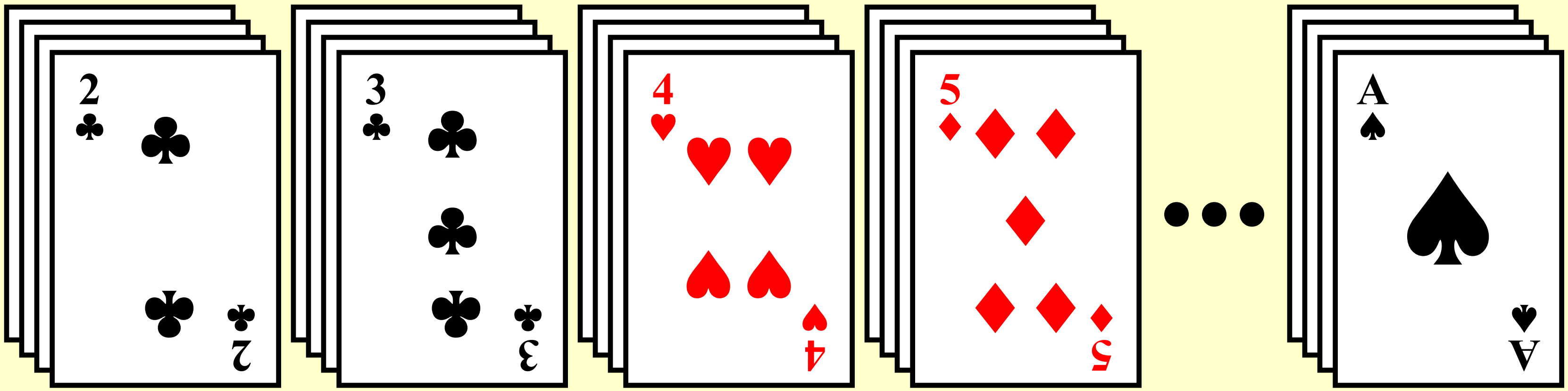


② Sort each bucket independently (using any sorting technique)



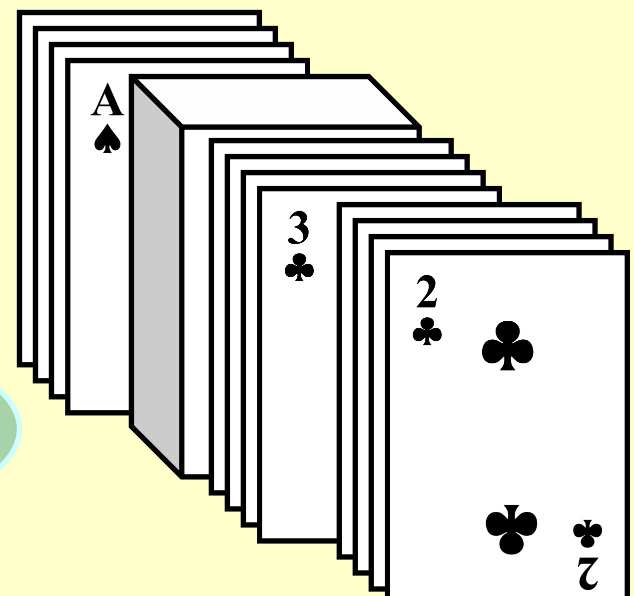
☞ LSD (Least Significant Digit) Sort

① Sort on K^1: for example, create 13 buckets for the face values



② Reform them into a single pile

③ Create 4 buckets and resort



Question:
Is LSD always faster than MSD?

Bonus Problem 2

Replacement Selection

(2 points)

Due: Monday, June 17th, 2024 at 10:00pm

The problem can be found and submitted at https://pintia.cn/