6.3 การเรียงลำดับแบบเร็ว (Quicksort)

กำหนดให้ข้อมูลที่ต้องการเรียงลำดับจัดเก็บไว้ในอาเรย์ A ขนาด n การทำงานเริ่มต้นด้วยการหาค่าไพวอท(pivot) ซึ่งค่าไพวอท(pivot) จะเป็นค่ากลาง (median) ของข้อมูล เมื่อได้ค่าpivot ก็จะแบ่งอาเรย์ A ออกเป็น 2 ส่วน

1. อาเรย์ย่อยด้านซ้ายเก็บข้อมูลที่มีค่าน้อยกว่าหรือเท่ากับค่าไพวอท
2. อาเรย์ย่อยด้านขวาเก็บข้อมูลที่มีค่ามากกว่าค่าไพวอท

โดยมีค่าไพวอทอยู่ตรงกลางระหว่างอาเรย์ย่อยด้านซ้ายและอาเรย์ย่อยด้านขวา เนื่องจากค่าในอาเรย์ย่อยด้านซ้ายมีค่าน้อยกว่าหรือเท่ากับค่าไพวอทและค่าในอาเรย์ย่อยด้านขวามีค่ามากกว่าค่าในไพวอท แสดงว่า ค่าไพวอทอยู่ในตำแหน่งที่ถูกต้องแล้ว หลังจากนั้นนำอัลกอริทึม ของการเรียงลำดับแบบควิกนี้ไปใช้ซ้ำในการเรียงลำดับอาเรย์ย่อยด้านซ้าย และอาเรย์ย่อยด้านขวา จนกระทั่งผลลัพธ์ที่ได้เรียงลำดับตามที่ต้องการ

≤ pivot	pivot	> pivot

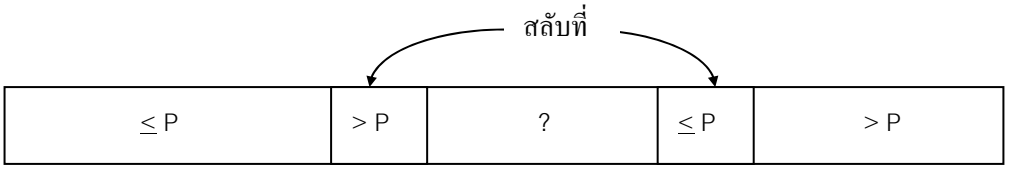
การแบ่งอาเรย์ออกเป็นอาเรย์ย่อยด้านซ้าย และอาเรย์ย่อยด้านขวา โดย **method partition** ซึ่งมีขั้นตอนวิธีดังนี้

ให้ P แทนดัชนีของค่าไพวอท
F แทนดัชนีของข้อมูลตัวแรกที่ต้องการพาร์ทิชัน
และ L แทนดัชนีของข้อมูลตัวสุดท้ายที่ต้องการพาร์ทิชัน

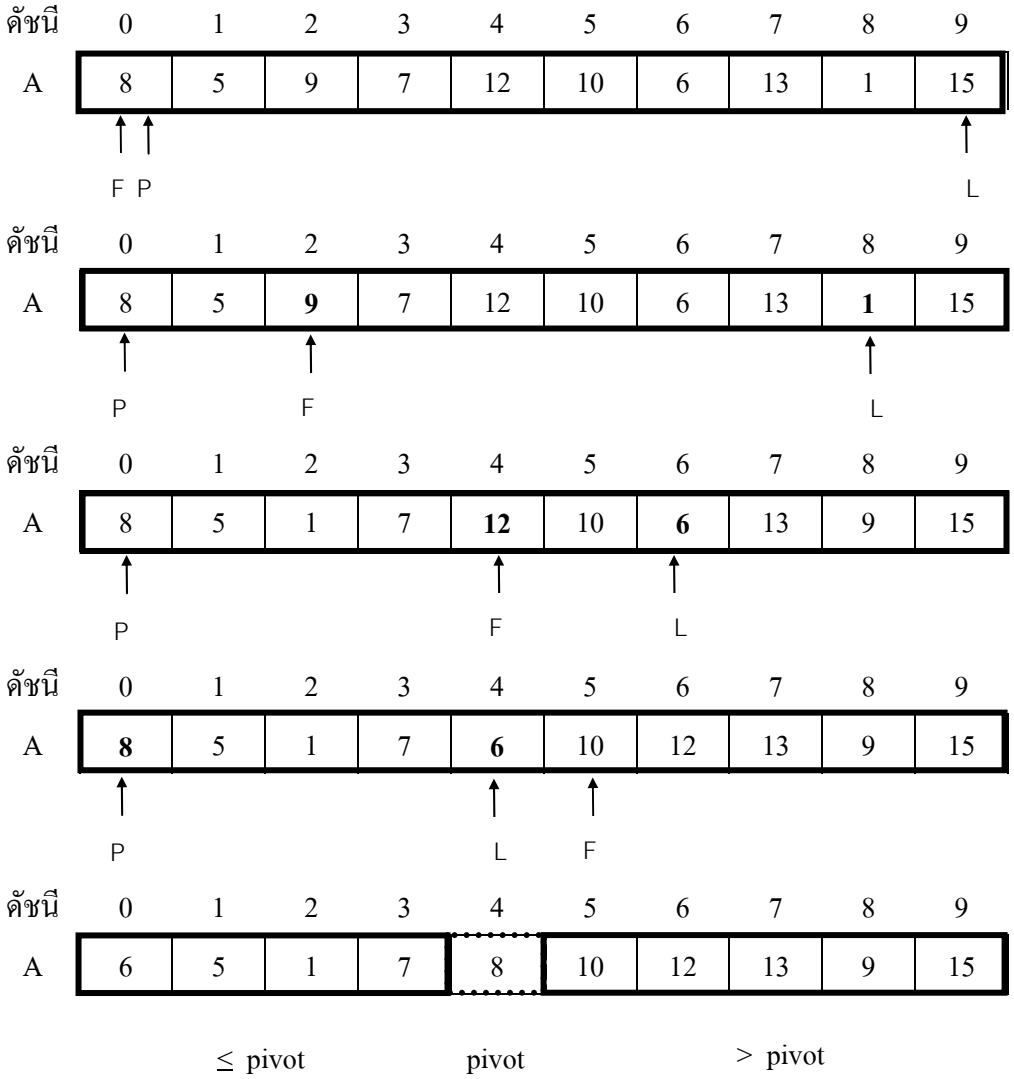
เริ่มต้นให้ F = 0 , L = n-1 หลังจากนั้นทำการกราดผ่านค่าในอาเรย์ดังนี้

1. เคลื่อน Fไปทางขวาทีละหนึ่งตำแหน่งไปเรื่อย ๆ และหยุดเมื่อพบค่าที่มากกว่าค่าไพวอท และเคลื่อน L ไปทางซ้ายทีละหนึ่งตำแหน่งไปเรื่อย ๆ และหยุดเมื่อพบค่าที่น้อยกว่าหรือเท่ากับไพวอท
2. กรณี F < L ให้สลับที่ระหว่าง A[F] กับ A[L] ซึ่งหลังจากสลับที่เรียบร้อยแล้ว A[F] จะมีค่าน้อยกว่าหรือเท่ากับค่าไพวอท และ A[L] จะมีค่ามากกว่าค่าไพวอท แล้วไปทำขั้นตอนที่ 1

กรณี F > L ให้สลับที่ระหว่าง A[P] และ A[L] และจบการพาร์ทิชัน



ตัวอย่าง 6.3 แสดงการพาร์ทิชันอาร์เรย์ A รอบที่ 1 โดยกำหนดให้ข้อมูลตัวแรกของอาเรย์ที่ต้องการพาร์ทิชันเป็นค่าไพวอท



ตัวอย่าง 6.4 แสดงการเรียงลำดับแบบเร็ว ของส่วนสูง นักเรียนจำนวน 10 คน

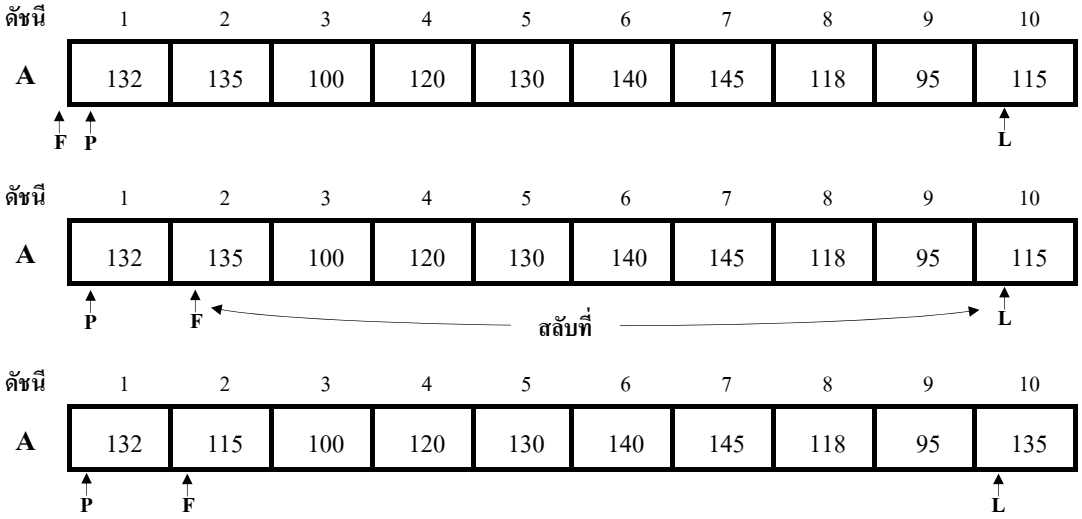
A = {132,135,100,120,130,140,145,118, 95 ,115}

Name = {“A”, “B”, “C”, “D”, “E”, “F”, “G”, “H”, “I”, “J”}

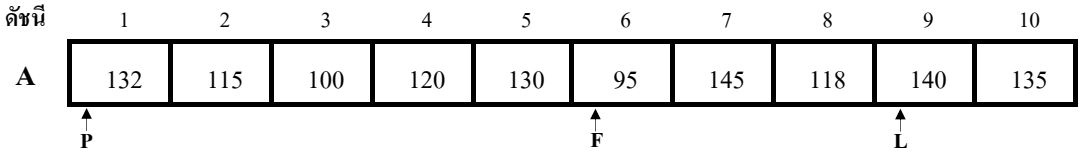
1	2	3	4	5	6	7	8	9	10
									
A-132	B-135	C-100	D-120	E-130	F-140	G-145	H-118	I-95	J-115

<u>วิธีทำ</u>

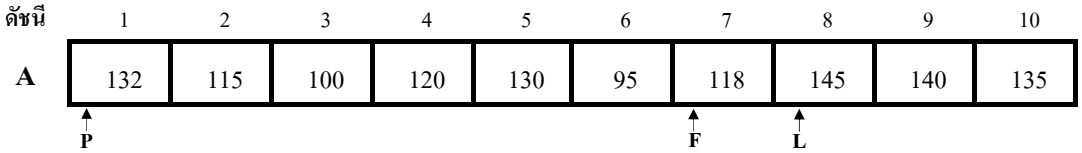
1. ให้ F=1, P=1, L=10, pivot = A[1] = 132 จากนั้นเลื่อน F ไปทางขวาจนกว่า A[F] > pivot และเลื่อน L มาทางซ้ายจนกว่า A[L] < pivot นั่นคือ F=2 และ L=10 จากนั้นสลับข้อมูล A[F] กับA[L]



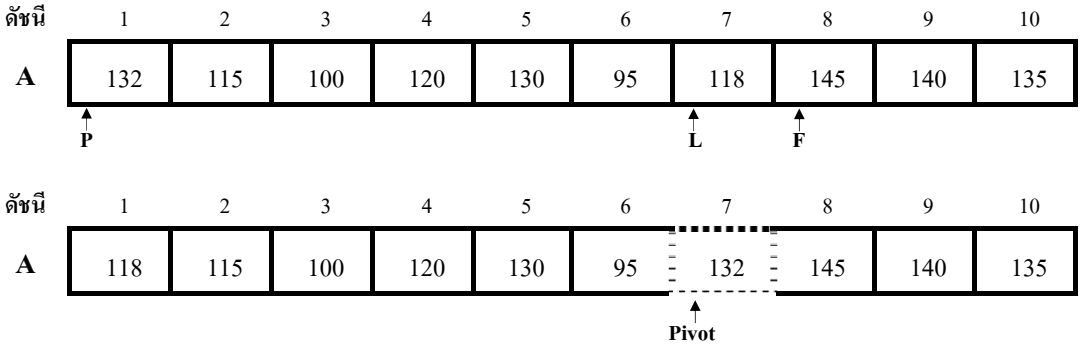
2. เลื่อน F กับ L อีก ได้ F=6 และ L = 9, สลับข้อมูล A[6] กับ A[9]



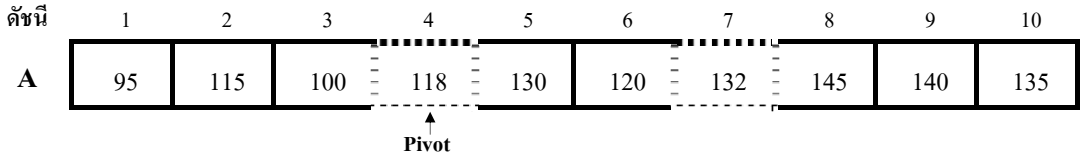
3. เลื่อน F กับ L อีก ได้ F=7 และ L =8, สลับข้อมูล A[7] กับ A[8]



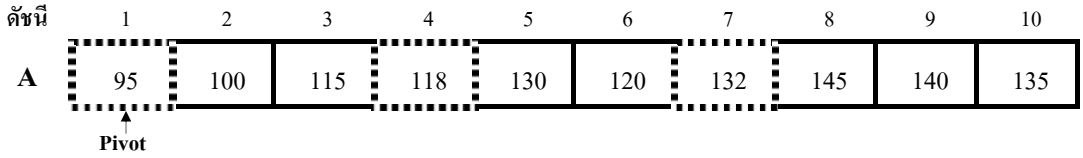
4. เลื่อน F กับ L อีก ได้ F=8 และ L =7, เพราะ F>L ให้สลับข้อมูล A[P] กับ A[L] (A[1] กับ A[8]



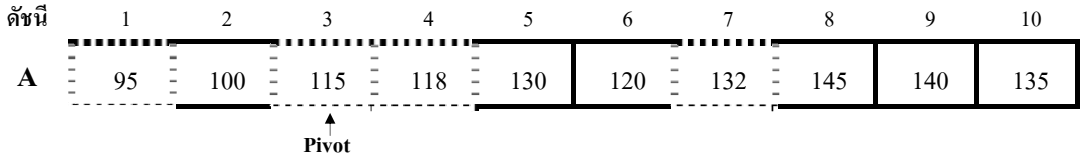
5. แบ่งอาเรย์ย่อย ครึ่งซ้ายเท่ากับ A[1]…A[6] และ ครึ่งขวาเท่ากับ A[8]…A[10] ทำการแบ่ง partition ครึ่งซ้ายอีก ตามขั้นตอนวิธี 1- 4 จะได้



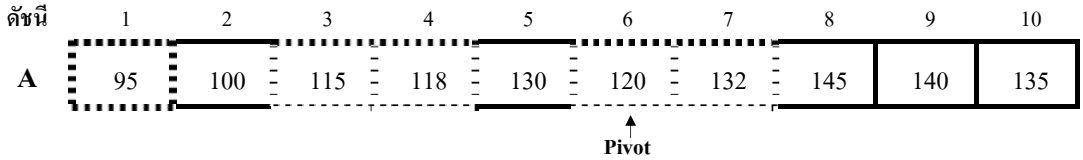
6. แบ่งอาเรย์ย่อย ครึ่งซ้ายเท่ากับ A[1]…A[3] และ ครึ่งขวาเท่ากับ A[5]…A[6] ทำการแบ่ง partition ครึ่งซ้ายอีก ตามขั้นตอนวิธี 1- 4 จะได้



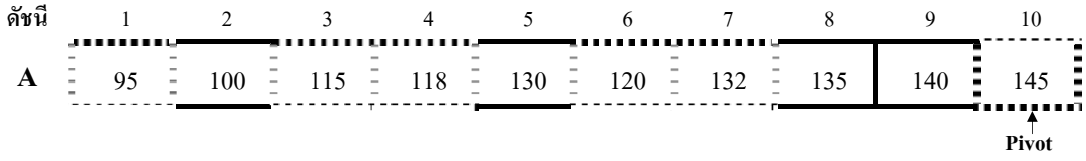
7. เพราะอาเรย์ย่อยครึ่งซ้ายเป็น 1 ไม่ต้องแบ่งอาเรย์ย่อย ย้อนกลับไปแบ่งอาเรย์ย่อยครึ่งขวา A[2]…A[3] ตามขั้นตอนวิธี 1- 4 จะได้



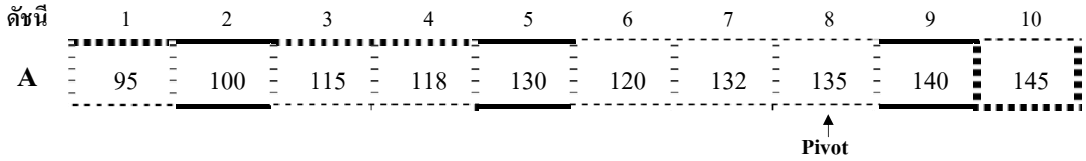
8. เพราะอาเรย์ย่อยครึ่งซ้ายเป็น 1 ไม่ต้องแบ่งอาเรย์ย่อย ย้อนกลับไปแบ่งอาเรย์ย่อยครึ่งขวา A[5]…A[6] ตามขั้นตอนวิธี 1- 4 จะได้



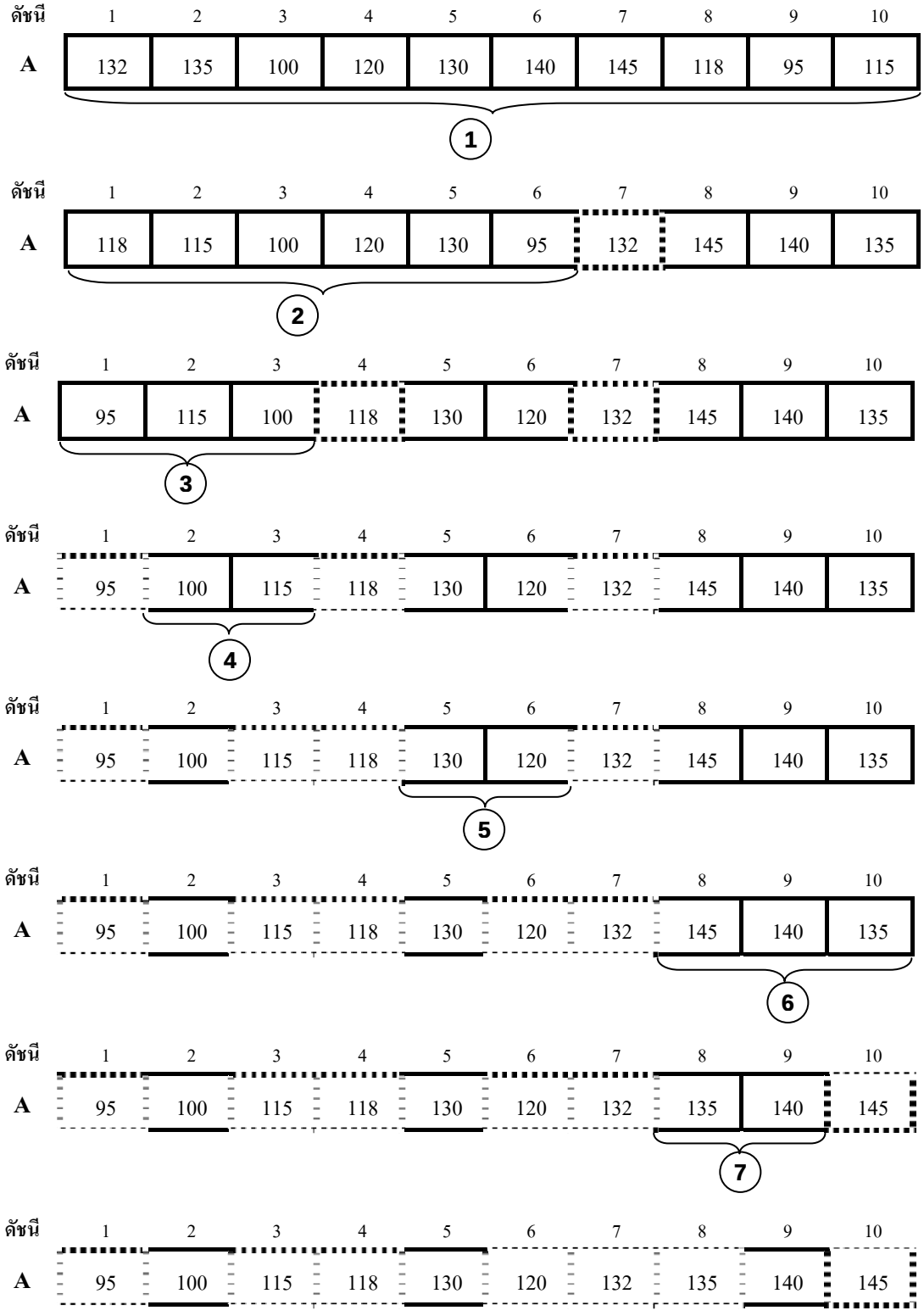
9. เพราะอาเรย์ย่อยครึ่งซ้ายเป็น 1 ไม่ต้องแบ่งอาเรย์ย่อย ย้อนกลับไปแบ่งอาเรย์ย่อยครึ่งขวา A[8]…A[10] ตามขั้นตอนวิธี 1- 4 จะได้



10. แบ่งอาเรย์ย่อย ครึ่งซ้ายเท่ากับ A[8]…A[9] และ ครึ่งขวาเท่ากับ A[10] แบ่ง partition อีก จะได้



11. สรุปรวมด้วยรูป 6. 7



รูป 6.7 ลำดับการแบ่ง partition ในวิธีเรียงลำดับแบบเร็ว

โปรแกรม QSort.java ซึ่งแสดงการเรียงลำดับแบบเร็ว

```
public class QSort
{ //public static int pivot;
public static void main(String argv[])
{   int aa[]={0, 132,135,100,120,130,140,145,118, 95 ,115};
   System.out.println("The original data before sorting");
   print(aa);
   System.out.println("Being sorting process");
   quicksort(aa, 1 ,10 );
   System.out.println("The data after sorting");
   print(aa);
}
//-----------------------------------------------------------------------------------------------//
public static void print(int[] a)
{for(int i=1;i< a.length;i++)
      System.out.print(a[i]+" ");
       System.out.println();
}
//-----------------------------------------------------------------------------------------------
public static void Swap(int[] a, int e1, int e2)
{    int tmp = a[e1];
    a[e1]= a[e2];  a[e2]= tmp;
    print(a);
}
//-----------------------------------------------------------------------------------------------
public static void quicksort( int[] a, int low, int high )
{     int pivot;
       if ( high > low )
       {
       pivot = partition( a, low, high );
       quicksort( a, low, pivot-1 );
       quicksort( a, pivot+1, high );
      }
}
//  เมท็อด partition  โดยกำหนดให้ข้อมูลค่าแรกเป็นค่าไพวอท
//  โดยทำการพาร์ทิชันอาเรย์ออกเป็น 2 ส่วน  โดยมีค่าไพวอทอยู่ตรงกลาง
public  static int partition( int[] a,  int  low, int  high )
{   int  left,  right , pivot;
   left = pivot = low;
   right = high;
   int   pivot_item = a[pivot];
   while (  left  <  right  )  {
         /*  เคลื่อนไปทางซ้ายขณะที่มีค่าน้อยกว่า หรือเท่ากับ pivot_item  */
```

```
            while ( ( a[left]  <=  pivot_item) &&( left < high) )   left++;
            /*  เคลื่อนไปทางขวาขณะที่มีค่ามากกว่า pivot_item  */
           while ( ( a[right]  >  pivot_item) && (right > low)  )    right --;
           if (left < right) Swap( a,  left, right );
            }
     a[pivot]  =  a[right];
     a[right]  =  pivot_item;
     print(a);
             System.out.println("pivot ="+right);
    return  right;
     }
}
```

outputs

```
The original data before sorting
132 135 100 120 130 140 145 118 95 115
Being sorting process
132 115 100 120 130 140 145 118 95 135
132 115 100 120 130 95 145 118 140 135
132 115 100 120 130 95 118 145 140 135
118 115 100 120 130 95 132 145 140 135
pivot =7
118 115 100 95 130 120 132 145 140 135
95 115 100 118 130 120 132 145 140 135
pivot =4
95 115 100 118 130 120 132 145 140 135
pivot =1
95 100 115 118 130 120 132 145 140 135
pivot =3
95 100 115 118 120 130 132 145 140 135
pivot =6
95 100 115 118 120 130 132 135 140 145
pivot =10
95 100 115 118 120 130 132 135 140 145
pivot =8
The data after sorting
95 100 115 118 120 130 132 135 140 145
```

ประสิทธิภาพของการเรียงลำดับแบบควิก

ในการเรียงลำดับแบบควิก จำนวนครั้งในการในการสลับที่จะมีจำนวนน้อยกว่าจำนวนครั้งในการเปรียบเทียบมาก จึงพิจารณาเฉพาะจำนวนครั้งของการเปรียบเทียบ

จำนวนครั้งของการเปรียบเทียบในการเรียงลำดับแบบควิกขึ้นอยู่กับการเลือกค่าไพวอท

<u>กรณีที่ดีที่สุด</u> คือ อาเรย์ A มีขนาดเป็น n ซึ่ง $n = 2^m$ หรือ $m = \log_2 n$ และทุกครั้งที่หาค่าไพวอทจะได้ว่าค่าไพวอทอยู่ ณ ตำแหน่งกึ่งกลางของอาเรย์ A เสมอ ดังนั้น หลังจากผ่านไปหนึ่งรอบ อาเรย์จะถูกแบ่งออกเป็น 2 ส่วน โดยแต่ละส่วนมีขนาดเป็น $\frac{n}{2}$ และ ในแต่ละอาเรย์ย่อยขนาด $\frac{n}{2}$ ทั้ง 2 ส่วนนี้จะถูกแบ่งครึ่งไปอีกเป็น 4 อาเรย์ย่อย โดยที่แต่ละส่วนย่อยมีขนาดเป็น $\frac{n}{4}$ และเป็นเช่นนี้ไปเรื่อย ๆ จนไม่สามารถแบ่งย่อยได้อีก นั่นคือ จะได้ว่าจำนวนครั้ง ของการแบ่งครึ่งอาเรย์ออกเป็นอาเรย์ย่อย ๆ จะทำได้ทั้งหมด m ครั้ง

ดังนั้นจำนวนครั้งของการเปรียบเทียบทั้งหมด

$$= n + 2(\frac{n}{2}) + 4(\frac{n}{4}) + ... + n(\frac{n}{n})$$

หรือ $= n + n + n + ... + n$ (ทั้งหมด m ครั้ง)

$$= nm = n \log_2 n = O(n \log n)$$

<u>กรณีที่แย่ที่สุด</u> คือ ค่าไพวอทที่ได้ทุกครั้งเป็นค่าที่น้อยที่สุดหรือเป็นค่าที่มากที่สุด ดังนั้นเมื่อผ่านการทำไปหนึ่งรอบ อาเรย์จะไม่ถูกแบ่งเป็น 2 ส่วน แต่จะลดขนาดลงเป็น n-1 และเมื่อผ่านการทำไป 2 รอบ อาเรย์จะมีขนาดลดลงเป็น n-2 ดังนั้นจึงทำเป็นจำนวน n รอบ

ดังนั้นจำนวนครั้งของการเปรียบเทียบทั้งหมด

$$= n + (n-1) + (n-2) + (n-3) + ... + 2 + 1 = \frac{n(n+1)}{2} = O(n^2)$$

<u>กรณีเฉลี่ย</u> จำนวนครั้งของการเปรียบเทียบทั้งหมด $= 2n \log_2 n = O(n \log n)$

6.4 การเรียงลำดับแบบเรดิก (Radix Sort)

การเรียงลำดับแบบเรดิกใช้แนวคิดที่แตกต่างจากการเรียงลำดับแบบอื่น ๆ ที่ได้กล่าวมาแล้วที่ผ่านมาเราเรียงลำดับข้อมูลโดยการเปรียบเทียบค่าของข้อมูล การเรียงลำดับแบบเรดิก จะทำการแยกค่าที่ต้องการเรียงลำดับออกเป็นตัวเลข และทำการจัดเรียงข้อมูลใหม่ตามค่าของตัวเลขที่แยกออกมา ซึ่งเป็นวิธีที่แปลกกว่าวิธีอื่นตรงที่ไม่มีการเปรียบเทียบค่าแต่อย่างใด

คำว่า “เรดิก” หมายถึง ฐานของระบบตัวเลข สิบเป็นเรดิกของเลขฐานสิบ

การเรียงลำดับแบบเรดิก เริ่มจากการนำข้อมูลทั้งหมดมาแยกเป็นคิว จากคิวที่ 0 ถึงคิวที่ 9 เป็นจำนวน 10 คิว ตามค่าของตัวเลขในหลักหน่วยและตามลำดับข้อมูลที่อ่านเข้ามา จากนั้นนำข้อมูลทั้งหมดมาเรียงต่อกันจากคิวที่ 0 ไปจนถึงคิวที่ 9 แล้วนำข้อมูลทั้งหมดไปแยกใส่คิวที่ 0 ถึงคิว

ที่ 9 อีกต่อหนึ่งตามลำดับผลลัพธ์ที่ได้ใหม่ โดยแยกตามค่าของตัวเลขในหลักสิบ และนำข้อมูลทั้งหมดมาเรียงต่อกันจากคิวที่ 0 ถึงคิวที่ 9 และทำซ้ำเช่นนี้ไปเรื่อย ๆ จนกระทั่งพิจารณาครบทุกหลัก

ตัวอย่างที่ 6.5 แสดงการเรียงลำดับแบบเรดิก โดยมีข้อมูลดังนี้

A = {432,135,100,520,130,140,145,818, 95 ,115}

<u>รอบที่ 1</u> หยิบตัวเลขใส่คิว(enqueue) ตามตัวเลขในหลักหน่วย

คิวที่	0	1	2	3	4	5	6	7	8	9
	100		432			135			818	
	520					145				
	130					95				
	140					115				

จากนั้นดึงตัวเลขออกจากคิว (dequeue) เพื่อเป็นข้อมูลอีกครั้ง คือ

{100, 520, 130,140, 432,135,145,95,115,818}

<u>รอบที่ 2</u> หยิบตัวเลขใส่คิว(enqueue) ตามตัวเลขในหลักร้อยสิบ

คิวที่	0	1	2	3	4	5	6	7	8	9
	100	115	520	130	140					95
		818		432	145					
				135						

จากนั้นดึงตัวเลขออกจากคิว (dequeue) เพื่อเป็นข้อมูลอีกครั้ง คือ

{100, 115,118,520,130,432,135,140,145,95}

<u>รอบที่ 3</u> หยิบตัวเลขใส่คิว(enqueue) ตามตัวเลขในหลักร้อย

คิวที่	0	1	2	3	4	5	6	7	8	9
	95	100			432	520			818	
		115								
		130								
		135								
		140								
		145								

จากนั้นดึงตัวเลขออกจากคิว (dequeue) คือ {95,100, 115,130,135,140,145,432,520,818}

หมดหลักร้อยแล้วก็จะเป็นการเรียงลำดับแล้ว

โปรแกรม 6.4 เมท็อด RadixSortApp ซึ่งใช้ในการเรียงลำดับแบบเรดิก

```
import  java.util.*;
public class RadixSortApp
{     public static int radix = 10;
      public static int digits = 3;
      public static void main(String argv[])
          {   int aa[]={0, 432,135,100,520,130,140,145,818, 95 ,115} ;
              System.out.println("The original data before sorting");
               print(aa);
               System.out.println("Being sorting process");
               radixsort(aa);
              System.out.println("The data after sorting");
              print(aa);
            }
//-------------------------------------------------------------------------------------------------------------------
      public static void  radixsort (int[] data )
      {    int  i,d, j, k, factor;
           Queue[]  queues = new Queue[20];
           for ( d = 0;  d < radix;  d++)
                queues[d] =  new  Queue(10);
           d =1;
           for (  factor = 1;  d <=  digits ; d++)
           {
                System.out.println( " at Factor :"+factor);
                 for ( j = 1;  j < 11; j++)
                 {
                    queues[ (data[j] / factor) % radix].enQueue (data[j] );
                    System.out.println("data = "+data[j]+ " \tQueue no \t"+((data[j] / factor) % radix));
                 } // next j
                System.out.println("Dequeue to be data for next Factor");
                k=0;
                 for ( i =  0;  i < 10;  i++)
                 {   while (!queues[i].isEmpty() )
                     {  k=k+1;
                          data[ k] =  queues[ i].deQueue() ;
                        System.out.print(data[k]+"\t");}
                   }// next  i
                 System.out.println();
                 factor = factor* radix;
            }  // next digit
      }  //  end radixsort ()
}    // end RadixSortApp.java
```

หมายเหตุ ในการเรียงลำดับแบบเรดิก มีการเรียกใช้คิว จึงต้องทำการนำเข้าคลาส Queue โดยใช้คำสั่ง import java.util.Queue;

Outputs

```
The original data before sorting
432 135 100 520 130 140 145 818 95 115
Being sorting process
 at Factor :1
data = 432     Queue no      2
data = 135     Queue no      5
data = 100     Queue no      0
data = 520     Queue no      0
data = 130     Queue no      0
data = 140     Queue no      0
data = 145     Queue no      5
data = 818     Queue no      8
data = 95      Queue no      5
data = 115     Queue no      5
Dequeue to be data for next Factor
100    520    130    140    432    135    145    95     115    818

 at Factor :10
data = 100     Queue no      0
data = 520     Queue no      2
data = 130     Queue no      3
data = 140     Queue no      4
data = 432     Queue no      3
data = 135     Queue no      3
data = 145     Queue no      4
data = 95      Queue no      9
data = 115     Queue no      1
data = 818     Queue no      1
Dequeue to be data for next Factor
100    115    818    520    130    432    135    140    145    95

 at Factor :100
data = 100     Queue no      1
data = 115     Queue no      1
data = 818     Queue no      8
data = 520     Queue no      5
data = 130     Queue no      1
data = 432     Queue no      4
data = 135     Queue no      1
```

```
data = 140    Queue no    1
data = 145    Queue no    1
data = 95     Queue no    0
Dequeue to be data for next Factor
95   100   115   130   135   140   145   432   520   818
The data after sorting
95 100 115 130 135 140 145 432 520 818
```

ประสิทธิภาพของการเรียงลำดับแบบเรดิก

ในการเรียงลำดับแบบเรดิก จะไม่มีการเปรียบเทียบแต่จะมีการคำนวณเพื่อหาค่าในหลักต่าง ๆ และมีการคัดลอกค่าไปใส่ในคิว และนำกลับมาใส่อาเรย์ โดยในแต่ละหลักของตัวเลขจะมีการคำนวณ n ครั้ง คัดลอกค่า 2n ครั้ง รวมเป็น 3n ครั้ง หากข้อมูลมี c หลัก จะมีการคำนวณและคัดลอกทั้งหมด $3cn = O(n)$ และมีข้อเสีย คือ ต้องการเนื้อที่เพิ่มเติมในการสร้างคิว ซึ่งเนื้อที่ที่ต้องการนี้มีขนาดเท่ากับ $O(n)$

ในความเป็นจริง เมื่อ n มีขนาดใหญ่ ข้อมูลก็มีจำนวนหลักมากขึ้น ซึ่งจำนวนครั้งของการคำนวณและการคัดลอกข้อมูลเท่ากับ [3n× จำนวนหลัก] ซึ่งจำนวนหลัก ก็คือ ค่า ($\log_{radix}$ ของข้อมูลที่มีค่ามากที่สุด) ซึ่งนั่นก็คือ $O(n \log n)$ เช่นเดียวกับการเรียงลำดับแบบควิกและแบบเมริดจ์

แบบฝึกหัด

1. ให้แสดงขั้นตอนการเรียงลำดับข้อมูลในแต่ละรอบของวิธีการเรียงลำดับข้อมูลแบบเรดิก เพื่อเรียงลำดับข้อมูลจากน้อยไปหามากของข้อมูลต่อไปนี้

   424, 887, 807, 709, 883, 616, 573, 413, 257, 313,
   679, 180, 975, 264

2. ให้แสดงขั้นตอนการเรียงลำดับข้อมูลอย่างละเอียด โดยวิธีการเรียงลำดับแบบควิก และแบบเมริดจ์ ในการเรียงลำดับข้อมูลจากน้อยไปหามากของข้อมูลต่อไปนี้

   23, 43, 2, 4, 26, 8, 34, 16, 9, 20, 45, 79

3. จงเปรียบเทียบข้อดี ข้อเสียของการเรียงลำดับข้อมูลแบบควิก แบบเมริดจ์ และแบบเรดิก

4. จงอธิบายวิธีการเรียงลำดับข้อมูลแบบเรดิก ในการเรียงลำดับข้อมูลที่เป็นคำ เช่น คำศัพท์ภาษาอังกฤษ คำศัพท์ภาษาไทย เป็นต้น

5. จงเขียนโปรแกรมเพื่อทำการเรียงลำดับข้อมูล วัน เดือน ปี ด้วยวิธีการเรียงลำดับข้อมูลแบบเรดิก