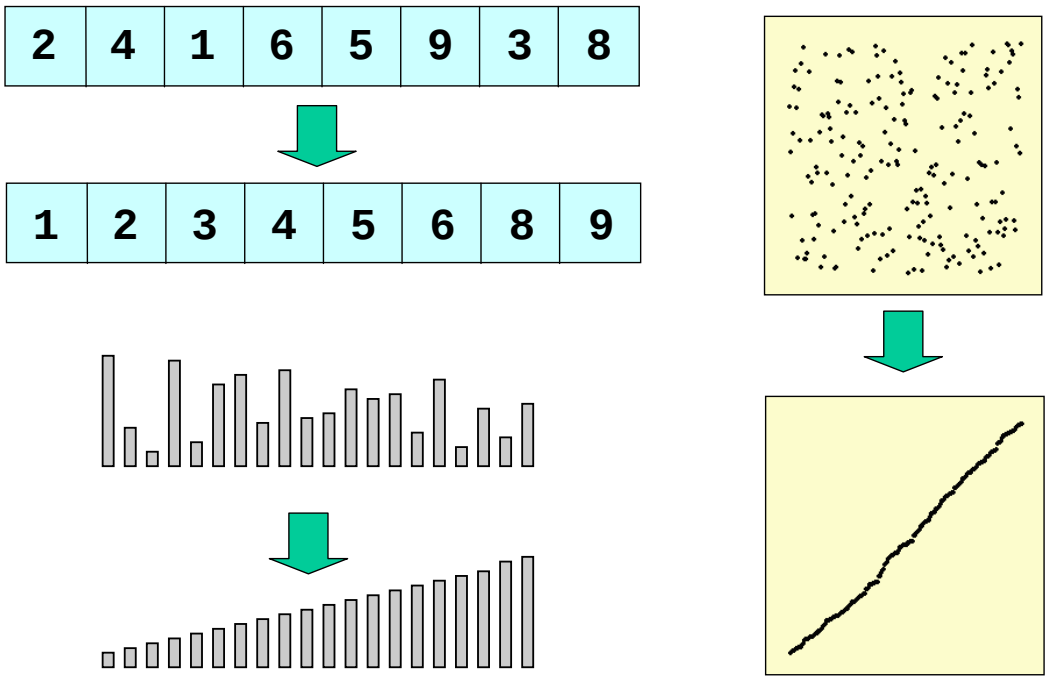
การเรียงลำดับข้อมูล

(Sorting)

หัวข้อ

- การเรียงลำดับแบบต่าง ๆ
  - เลือก, ฟอง, แทรก, เชลล์, ฮีป, ผสาน, เร็ว
- ประสิทธิภาพการเรียงลำดับ
  - เวลาการทำงาน
  - ปริมาณหน่วยความจำ
- การเปรียบเทียบการเรียงลำดับแบบต่าง ๆ

การเรียงลำดับ (Sorting)



วิธีการเรียงลำดับ

- เลือก : Selection sort $\Theta(n^2)$
- ฟอง : Bubble sort $O(n^2)$
- แทรก : Insertion sort $O(n^2)$
- เชลล์ : Shell sort $O(n^{1.xx})$
- ฮีป : Heap sort $O(n \log n)$
- ผสาน : Merge sort $O(n \log n)$
- เร็ว : Quick sort $O(n \log n)$ avg.

อาศัยการนำข้อมูลมาเปรียบเทียบกันทีละคู่

lessThan, swap

```
public class ArrayUtil {
  ...
  private static boolean lessThan(Object a, Object b) {
    return ((Comparable)a).compareTo(b) < 0;
  }

  private static void swap(Object[] d, int i, int j) {
    Object t = d[i]; d[i] = d[j]; d[j] = t;
  }
  ...
}
```


การเรียงลำดับแบบเลือก (Selection Sort)

- เรียงลำดับข้อมูล d[0] ถึง d[k]
- หาตัวมากสุด
- สลับตัวมากสุดกับตัวท้ายของกลุ่ม
- ขนาดของกลุ่มลดลงหนึ่ง
- ไปเรียงลำดับ d[0] ถึง d[k-1]
- ...
- ทำจนข้อมูลของกลุ่มเหลือตัวเดียว

การเรียงลำดับแบบเลือก : โปรแกรม

```
public static void selectionSort(Object[] d) {
  for (int k=d.length-1; k>0; k--) {
    int m = k;
    for (int j=0; j<k; j++)
      if (lessThan(d[m],d[j])) m = j;
    swap(d, m, k);  // d[m] ←→ d[k]
  }
}
```

$\Theta(n^2)$

รอบที่	จำนวนข้อมูล	#(lessThan)	
1	n	n-1	Σ = n(n-1)/2
2	n-1	n-2	
...	...	...	
n-1	2	1	
			#swaps = n-1

การเรียงลำดับแบบฟอง (Bubble Sort)

- เรียงลำดับข้อมูล d[0] ถึง d[k]
- เปรียบเทียบคู่ที่ติดกัน จาก 0 ถึง k สลับกันถ้ากลับลำดับ
- ขนาดของกลุ่มลดลงหนึ่ง
- ไปเรียงลำดับ d[0] ถึง d[k-1]
- ...
- ทำจนข้อมูลของกลุ่มเหลือตัวเดียว

การเรียงลำดับแบบฟอง : โปรแกรม

```
public static void bubbleSort(Object[] d) {
  for (int k=d.length; k>1; k--)
    for (int j=1; j<k; j++)
      if (lessThan(d[j], d[j-1])) swap(d, j-1, j);
}
```

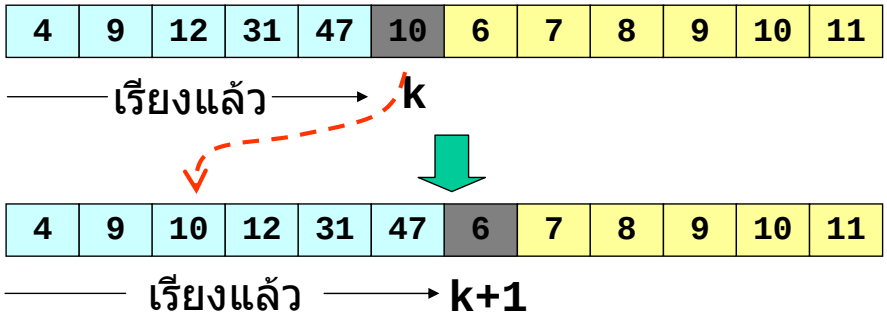
$\Theta(n^2)$

```
public static void bubbleSort(Object[] d)
  for (int k=d.length; k>1; k--) {
    boolean sorted = true;
    for (int j=1; j<k; j++) {
      if (lessThan(d[j], d[j-1])) {
        swap(d, j-1, j);
        sorted = false;
      }
    }
    if (sorted) break;
  }
}
```

$O(n^2)$

การเรียงลำดับแบบแทรก (Insertion Sort)

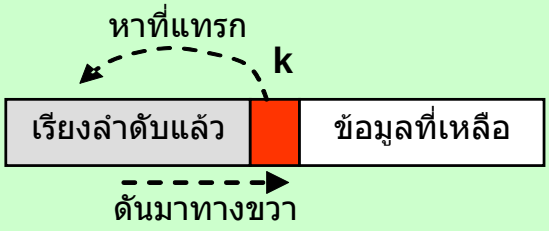
- ต้องการเรียงลำดับ d[0] ถึง d[m]
- พิจารณาข้อมูลตั้งแต่ตัวที่ 1 ถึง m
- เมื่อกำลังพิจารณาข้อมูลตัวที่ k
  - ข้อมูลทางซ้ายของ k : d[0] ถึง d[k-1] เรียงแล้ว
  - หาที่แทรกให้ d[k] ในช่วง 0 ถึง k



การเรียงลำดับแบบแทรก : โปรแกรม

```
public static void insertionSort(Object[] d) {
  for (int k=1; k<d.length; k++) {
    Object t = d[k];
    int j = k-1;
    while (j>=0 && lessThan(t, d[j])) {
      d[j+1] = d[j];
      j--;
    }
    d[j+1] = t;
  }
}
```

$O(n^2)$



รอบที่	จำนวนข้อมูลทางซ้าย	#(lessThan)
1	1	1
2	2	1 ถึง 2
...	...	...
n-1	n-1	1 ถึง n-1

$\sum$ = n-1 ถึง n(n-1)/2

การเรียงลำดับแบบเชลล์ : แนวคิด

- การเรียงลำดับแบบแทรก
  - ช้า เพราะเลื่อนข้อมูลไปช่องถัดไป ทีละตำแหน่ง
- การเรียงลำดับแบบเชลล์
  - แบ่งข้อมูลเป็น h ชุด แบบตัวเว้น h – 1 ตัว
  - sort แต่ละชุดด้วยการเรียงลำดับแบบแทรก
  - ถ้า h == 1 ก็เสร็จ ไม่เช่นนั้น ลดค่า h ลง กลับไปข้อ 1
- หมายเหตุ
  - รอบสุดท้ายเป็นการเรียงลำดับแบบแทรกชุดเดียว

Donald Shell, 1959

การเรียงลำดับแบบเชลล์ : ตัวอย่าง

- แบ่งเป็น 2 ชุด
- เรียงลำดับแต่ละชุดด้วยแบบแทรก
- รวมกับเป็นชุดเดียว แล้วเรียงลำดับทั้งชุด
- ถ้าเรียงลำดับครั้งเดียวเสร็จด้วยแบบแทรก

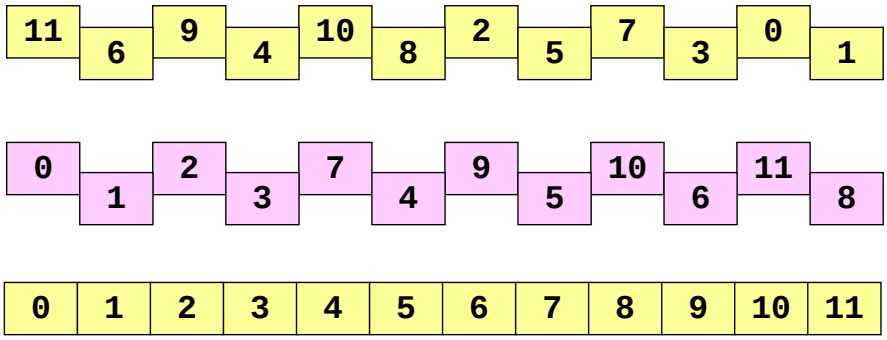
15+14

20

49

59

h=1,2



การเรียงลำดับแบบเชลล์ : โปรแกรม

```
public static void shellSort(Object[] d) {
  for (int h=d.length/2; h>0; h/=2) {
    for (int m = 0; m < h; m++) {
      for (int k = h+m  ; k < d.length; k += h) {
        Object t = d[k];  int j = k - h;
        while( j >= 0 && lessThan(t, d[j]) ) {
          d[j + h] = d[j];  j -= h;
        }
        d[j + h] = t;
      }
    }
  }
}
```

h-sequence อะไรดี ?

- ชุดที่ดีมักมีค่าลดลงเป็นเท่า ๆ
  - Shell : 1,2,4,8, ..., 2^m,... $O(n^2)$
  - Hibbard : 1,3,7,15, ..., $2^m - 1$, ... $O(n^{3/2})$
  - Knuth : 1,4,13,40,121, ..., $(3^m - 1)/2$, ... $O(n^{3/2})$
  - Sedgewick : 1,8,23,77, ..., $4^{m+1}+3\cdot 2^m+1$,... $O(n^{4/3})$
- ยังไม่มีผู้ใดพบ h-sequence ที่ดีสุด

```
public static void shellSort(Object[] d) {
  for (int h =d.length/2; h>0; h/=2) {
    for (int m = 0; m < h; m++) {
      ...
```

```
h = h==2 ? 1 : (int)(h/2.2)
```

ของ Gonnet

ทำไมแบบเชลล์ถึงเร็ว

- แบบแทรก : ข้อมูลย้ายตำแหน่งช้า
- แบบเชลล์
  - เริ่มด้วยค่า h ที่มาก แล้วลดลง ๆ
  - h มีค่ามาก : ข้อมูลย้ายตำแหน่งเร็ว
  - ข้อมูลเข้าใกล้ตำแหน่งที่ควรจะอยู่ได้อย่างรวดเร็ว
  - ใช้แบบแทรกเป็นเครื่องมือเพราะ "ยิ่งเรียงยิ่งเร็ว"
  - รอบหลัง ๆ, h น้อย ๆ, เรียงมาก, ก็ทำงานเร็ว

0	1	2	3	4	5	6	7	8	9	10	11
10	12	21	5	8	2	6	9	0	9	10	11

เปรียบเทียบเวลาการเรียงลำดับ

ข้อมูลเริ่มต้นเรียงลำดับอยู่แล้ว (เวลาเป็น ms)

n	เลือก	ฟอง	แทรก	เชลล์
200	0.44	0.01	0.01	0.03
400	1.80	0.01	0.02	0.06
800	7.01	0.02	0.03	0.18
1,600	28.59	0.04	0.06	0.29
3,200	112.64	0.07	0.11	0.73
6,400	459.64	0.14	0.23	1.54
12,800	1814.66	0.28	0.45	3.48
25,600	7240.25	0.57	1.00	12.10
51,200	29179.25	1.25	1.91	17.31
102,400	123337.25	2.64	3.91	61.43

เปรียบเทียบเวลาการเรียงลำดับ

ข้อมูลเริ่มต้นเรียงกลับลำดับ (เวลาเป็น ms)

n	เลือก	ฟอง	แทรก	เชลล์
200	0.44	0.81	0.55	0.06
400	1.80	3.23	2.29	0.15
800	7.36	12.86	9.73	0.29
1,600	28.43	51.01	35.18	0.64
3,200	112.62	210.30	141.22	1.36
6,400	458.97	844.04	573.99	4.57
12,800	1816.83	3349.86	2266.79	6.55
25,600	7250.50	13449.50	8988.00	15.49
51,200	29249.75	53754.75	36625.25	31.10
102,400	124168.50	221318.25	153943.75	102.35

เปรียบเทียบเวลาการเรียงลำดับ

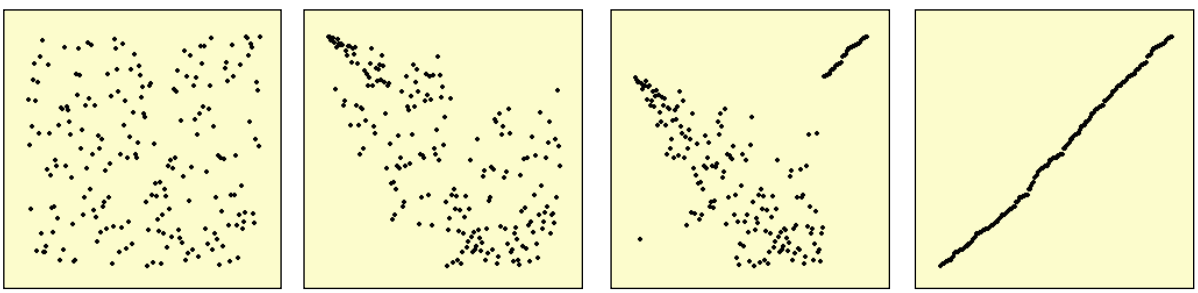
ข้อมูลเริ่มต้นเป็นแบบสุ่ม(เวลาเป็น ms)

n	เลือก	ฟอง	แทรก	เชลล์
200	0.45	0.19	0.30	0.08
400	1.82	3.13	1.10	0.19
800	7.02	15.38	4.38	0.45
1,600	31.01	49.71	16.93	1.10
3,200	112.86	207.47	74.17	2.49
6,400	452.57	823.90	284.56	5.83
12,800	1834.78	3318.25	1147.65	12.11
25,600	7273.00	13349.25	4639.25	27.83
51,200	30183.50	56241.00	20123.75	62.43
102,400	200588.50	332638.25	171231.25	184.83

การเรียงลำดับแบบฮีป (Heap Sort)

- รับอาเรย์มาสร้างให้เป็นฮีปมากสุด
- เข้าวงวน dequeue ข้อมูล (ตัวมากสุด) เพื่อนำไปไว้ ณ ตำแหน่งหลังสุดของกลุ่ม

0	1	2	3	4
25	12	10	11	27



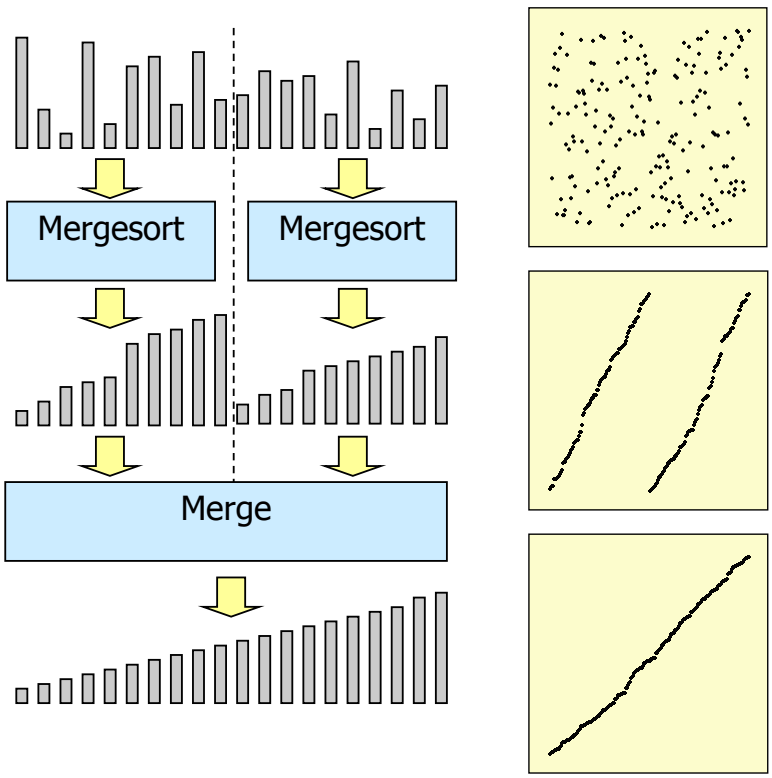
การเรียงลำดับแบบฮีป : โปรแกรม

```
public static void heapSort(Object[] d) {
  int size = d.length;
  for(int k=size/2-1; k>=0; k--)
    fixDown(d, size, k);
  for(int k=size-1; k>0; k--) {
    swap(d, 0, k);
    fixDown(d, --size, 0);
  }
}
static void fixDown(Object[] d, int size, int k) {
  int c;
  while ((c = 2 * k + 1) < size) {
    if (c < size-1 && lessThan(d[c], d[c+1])) c++;
    if (!lessThan(d[k], d[c])) break;
    swap(d, c, k);
    k = c;
  }
}
```

$O(n)$

$O(n \log n)$

การเรียงลำดับแบบผสาน (Merge Sort)



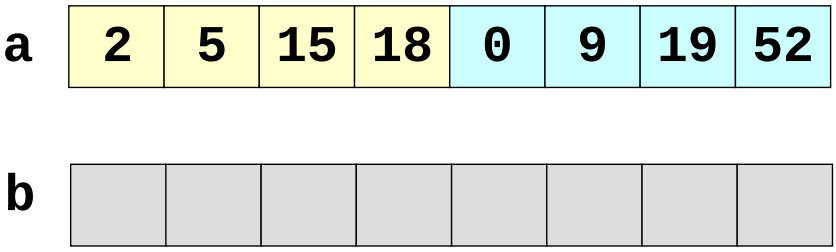
การเรียงลำดับแบบผสาน : โปรแกรม

```
public static void mergeSort(Object[] d) {
  mSortR(d, 0, d.length-1,(Object[]) d.clone());
}

private static
void mSortR(Object[] d,
            int left, int right, Object[] t) {
  if (left < right) {
    int m = (left + right)/2;
    mSortR(t, left, m, d);
    mSortR(t, m + 1, right, d);
    merge(t, left, m, right, d);

  }
}
```

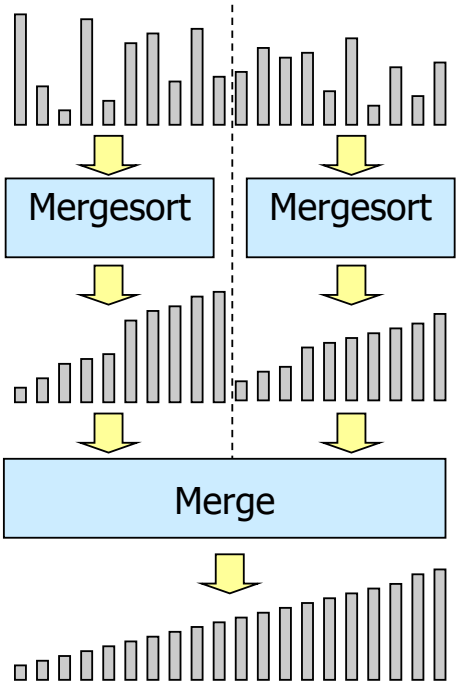

การผสาน (merge)




```
static void merge(Object[] a, int left, int mid,
                  int right, Object[] b) {
  int i = left, j = mid+1;
  for (int k = left; k <= right; k++) {
    if (i > mid)   {b[k] = a[j++]; continue;}
    if (j > right) {b[k] = a[i++]; continue;}
    b[k] = lessThan(a[i], a[j]) ? a[i++] : a[j++];
  }
}
```

การเรียงลำดับแบบผสาน : วิเคราะห์

ให้ $c(n)$ คือจำนวนครั้งของการเปรียบเทียบเพื่อเรียงลำดับข้อมูล n ตัว



$c(n/2) + c(n/2)$

$$c(n) \geq 2c(n/2) + n/2$$
$$c(n) \leq 2c(n/2) + n - 1$$

$n/2$ ถึง $n - 1$

ขอบเขตบนของจำนวนการเปรียบเทียบ

$$\begin{aligned}
c(n) &\leq 2c(n/2) + n - 1 \\
&\leq 2(\ 2c(n/4) + n/2 - 1\ ) + n - 1 \\
&= 4c(n/4) + n - 2 + n - 1 \\
&\leq 4(\ 2c(n/8) + n/4 - 1\ ) + n - 2 + n - 1 \\
&= 8c(n/8) + n - 4 + n - 2 + n - 1 \\
&\ \ \ \ \ldots \\
&\leq 2^k c(n/2^k) + n - 2^{k-1} + \ldots + n - 2 + n - 1 \\
&= 2^k c(n/2^k) + nk - (2^k - 1) \\
&= n \log_2 n - n + 1 \\
&= O(n \log n)
\end{aligned}$$

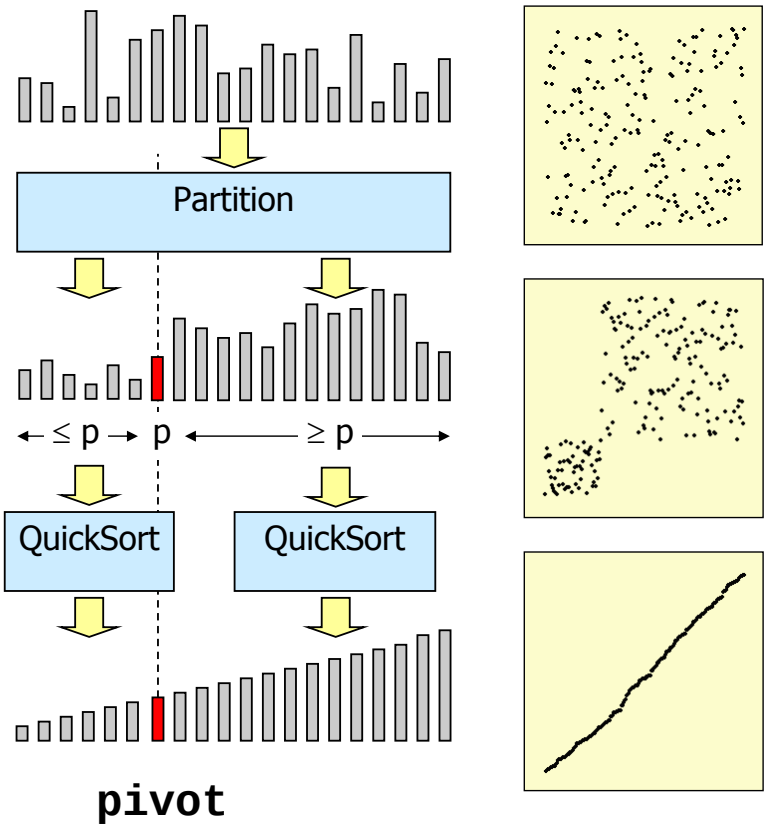
$c(1) = 0$

ให้ $n = 2^k$, $k = \log_2 n$

เวลาการทำงาน

- ขอบเขตบน : $c(n) \leq 2c(n/2) + (n-1) = O(n \log n)$
- ขอบเขตล่าง : $c(n) \geq 2c(n/2) + n/2 = \Omega(n \log n)$
- จำนวนการเปรียบเทียบ = $\Theta(n \log n)$
- จำนวนการย้าย $g(n) = 2g(n/2) + n = \Theta(n \log n)$
- เวลาการทำงาน = $\Theta(n \log n)$

การเรียงลำดับแบบเร็ว (Quick Sort)



การเรียงลำดับแบบเร็ว : โปรแกรม

```
public static void quickSort(Object[] d) {
  qSortR(d, 0, d.length-1);
}
static void qSortR(Object[] d, int left, int right){
  if (left < right) {
    int j = partition(d, left, right);
    qSortR(d, left, j - 1);
    qSortR(d, j + 1, right);
  }
}
```

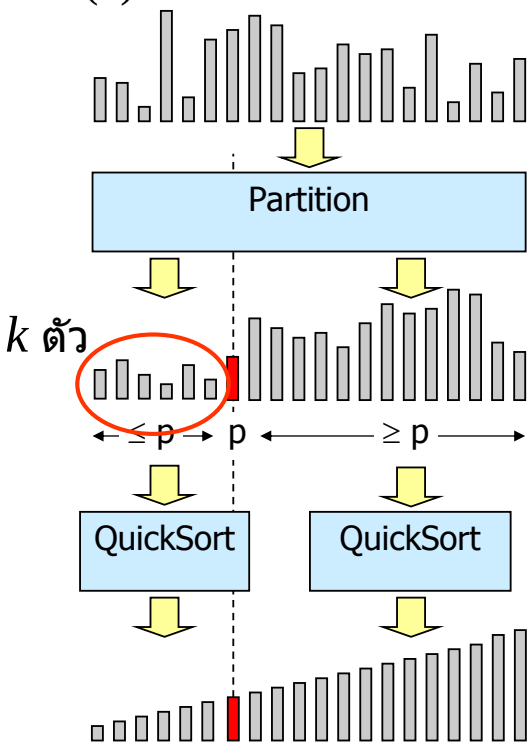

การแบ่งส่วน (partition)

d	12	5	15	18	0	9	11	52

```
static int partition(Object[] d, int left, int right) {
  Object p = d[left];
  int i = left, j = right + 1;
  while (i < j) {
    while (lessThan(p, d[--j]));
    while (lessThan(d[++i], p)) if (i == right) break;
    if (i < j) swap(d, i, j);
  }
  swap(d, left, j);
  return j;
}
```

การเรียงลำดับแบบเร็ว : วิเคราะห์

ให้ $c(n)$ คือจำนวนครั้งของการเปรียบเทียบเพื่อเรียงลำดับข้อมูล n ตัว



$n - 1$

$$c(n) = c(k) + c(n - k - 1) + n - 1$$

$c(k) + c(n - k - 1)$

จำนวนการเปรียบเทียบ : กรณีเร็วสุด

$$c(n) = c(k) + c(n - k - 1) + n - 1$$

$$c_{min}(n) = c_{min}(\lfloor n/2 \rfloor) + c_{min}(n - \lfloor n/2 \rfloor - 1) + n - 1$$

$$\leq 2c_{min}(n/2) + n - 1$$

$$= n \log_2 n - n + 1$$

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

จำนวนการเปรียบเทียบ : กรณีช้าสุด

$$c(n) = c(k) + c(n-k-1) + n - 1$$

$$\begin{aligned} c_{max}(n) &= c(0) + c_{max}(n-1) + n - 1 \\ &= c_{max}(n-1) + n - 1 \\ &= c_{max}(n-2) + n - 2 + n - 1 \\ &\ldots \\ &= c_{max}(1) + 1 + 2 + \ldots + n - 2 + n - 1 \\ &= n(n-1)/2 \\ &= \Theta(n^2) \end{aligned}$$

จำนวนการเปรียบเทียบ : กรณีเฉลี่ย

$$c(n) = c(k) + c(n-k-1) + n - 1$$

$$c_{\text{avg}}(n) = \frac{1}{n}\sum_{k=0}^{n-1}\Big(c_{\text{avg}}(k) + c_{\text{avg}}(n-k-1)\Big) + (n-1)$$

$$= \frac{2}{n}\sum_{k=0}^{n-1} c_{\text{avg}}(k) + (n-1)$$

$$nc_{\text{avg}}(n) = 2\sum_{k=0}^{n-1} c_{\text{avg}}(k) + n(n-1)$$

$$(n-1)c_{\text{avg}}(n-1) = 2\sum_{k=0}^{n-2} c_{\text{avg}}(k) + (n-1)(n-2)$$

$$nc_{\text{avg}}(n) = (n+1)c_{\text{avg}}(n-1) + 2(n-1)$$

จำนวนการเปรียบเทียบ : กรณีเฉลี่ย

$$nc_{\text{avg}}(n) = (n+1)c_{\text{avg}}(n-1) + 2(n-1)$$

$$\frac{c_{\text{avg}}(n)}{n+1} = \frac{c_{\text{avg}}(n-1)}{n} + \frac{2(n-1)}{n(n+1)}$$

$$= \frac{c_{\text{avg}}(1)}{2} + 2\sum_{i=2}^{n} \frac{(i-1)}{i(i+1)}$$

$$\approx 2\sum_{i=2}^{n} \frac{1}{(i+2)}$$

$$= 2(\ln n + \mathrm{O}(1))$$

$$c_{\text{avg}}(n) = 2n\ln n + \mathrm{O}(n)$$

$$\approx 1.39n\log_2 n + \mathrm{O}(n)$$

$$= \mathrm{O}(n\log n)$$

การเลือกตัวหลัก (pivot)

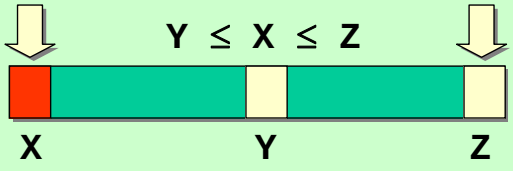
- ที่ผ่านมา : เลือกตัวซ้ายสุดเป็นตัวหลัก
- ถ้าข้อมูลเริ่มต้นเรียงลำดับอยู่แล้ว
  - หลังการแบ่งส่วน ชุดซ้ายมี 0 ตัวเสมอ
  - เกิดกรณีช้าสุด : $\Theta(n^2)$
  - ถ้าใช้โปรแกรมที่เขียนแบบเวียนเกิด
  - เกิดการเรียกเวียนเกิดซ้อน ๆ กันจำนวน n – 1 ครั้ง
  - มีโอกาสเกิด StackOverflowException
- สุ่มเลือกตัวหลัก
  - โอกาสเกิดกรณีช้าสุดมีน้อยมาก ๆ

```
static int partition(Object[] d, int left, int right) {
  int m = left + (int)(Math.random()*(right-left+1));
  swap(d, left, m);
  Object p = d[left];
  ...
```

การเลือกมัธยฐานสามเป็นตัวหลัก

- ใช้ตัวหลักที่ได้มาจากมัธยฐานของตัวซ้าย ขวา และกลางของข้อมูล

```
static int partition(Object[] d, int left, int right) {
  int c = (left + right)/2;
  if (lessThan(d[left],d[c])) swap(d, left, c);
  if (lessThan(d[right],d[c])) swap(d, c, right);
  if (lessThan(d[right],d[left])) swap(d, left, right);
  Object p = d[left];
  int i = left, j = right + 1;
  while (i < j) {
    while (lessThan(p, d[--j]));
    while (lessThan(d[++i], p)) if (i==right) break;
    if (i < j) swap(d, i, j);
  }
  swap(d, left, j);
  return j;
}
```



เปรียบเทียบเวลาเรียงลำดับ

ข้อมูลเริ่มต้นเรียงลำดับ (เวลาเป็น ms)

n	เชลล์	ฮีป	ผสาน	เร็ว	เร็ว (สุ่ม)	เร็ว (M3)
1000	0.20	0.96	0.38	11.66	0.48	0.29
10000	2.96	16.74	4.77	-	5.94	4.38
100000	64.67	168.72	58.99	-	67.60	46.87
1000000	1153.70	2005.57	724.59	-	787.78	503.23

เปรียบเทียบเวลาเรียงลำดับ

ข้อมูลเริ่มต้นเรียงกลับลำดับ (เวลาเป็น ms)

n	เชลล์	ฮีป	ผสาน	เร็ว	เร็ว (สุ่ม)	เร็ว (M3)
1000	0.40	0.77	0.39	10.51	0.49	0.30
10000	6.45	11.51	9.86	-	6.23	5.95
100000	113.26	154.61	60.88	-	71.68	48.66
1000000	1692.68	1997.14	745.68	-	809.86	541.56

เปรียบเทียบเวลาเรียงลำดับ

ข้อมูลเริ่มต้นแบบสุ่ม (เวลาเป็น ms)

n	เชลล์	ฮีป	ผสาน	เร็ว	เร็ว (สุ่ม)	เร็ว (M3)
1000	0.97	0.96	0.56	0.58	0.94	0.52
10000	12.82	15.53	10.27	6.88	15.37	8.89
100000	199.27	224.27	111.66	106.92	121.83	102.17
1000000	3143.03	4303.59	1682.07	1591.25	1810.50	1600.21

เปรียบเทียบเนื้อที่เสริม

- ใช้เนื้อที่เสริมเพื่อการเรียงลำดับ
  - $\Theta(1)$: selection, bubble, insertion, Shell, heap, quick
  - $\Theta(n)$: merge
- ใช้เนื้อที่เสริมเพราะเขียนแบบเวียนเกิด
  - quick : น้อยสุด $\Theta(\log n)$, มากสุด $\Theta(n)$
  - merge : $\Theta(\log n)$
  - สามารถเขียน quicksort และ mergesort แบบวนวน (ไม่ต้อง recursive) ได้

สรุป

- ไม่ควรใช้การเรียงลำดับแบบฟอง
- การเรียงลำดับแบบเลือกย้ายข้อมูลน้อยครั้งสุด
- ถ้าข้อมูลไม่มาก น่าใช้การเรียงลำดับแบบแทรก
- การเรียงลำดับแบบเชลล์เร็วมาก
- การเรียงลำดับแบบผสานเร็วกว่า ถ้าข้อมูลมาก แต่ต้องการเนื้อที่เสริม
- การเรียงลำดับแบบฮีปเป็น O(n log n) แต่ช้า
- การเรียงลำดับแบบเร็ว ต้องเขียนดี ๆ ถึงเร็ว และเร็วมาก ๆ