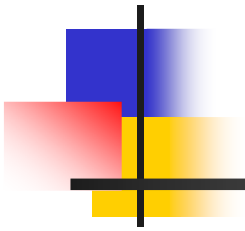
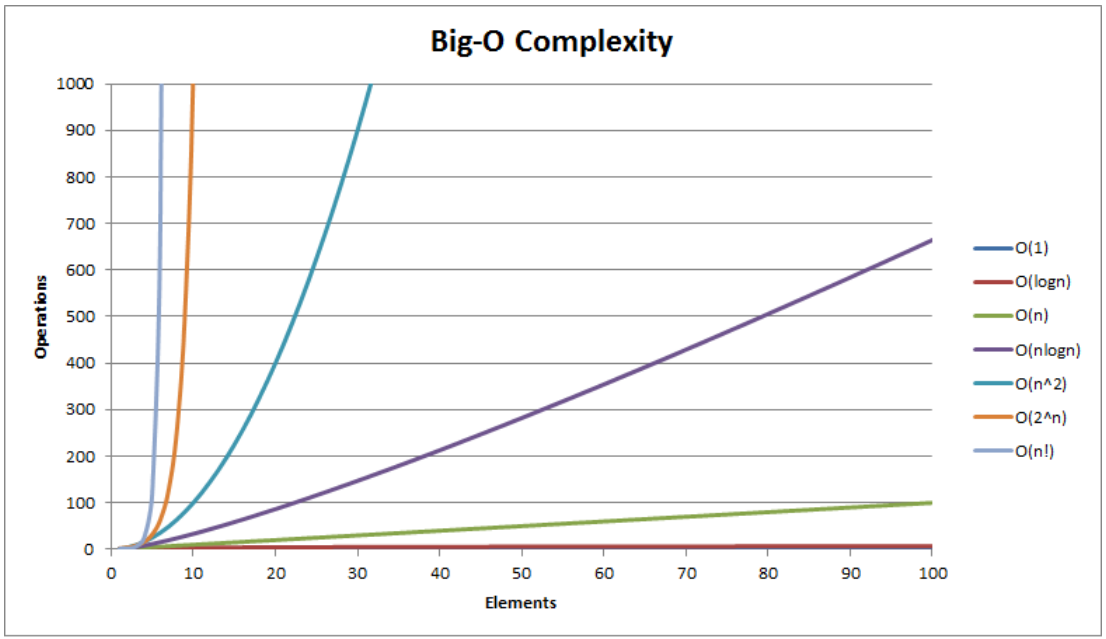


Chapter 01

Searching & Sorting

Complexity Analysis

- การวัดประสิทธิภาพ (ความซับซ้อน) ของอัลกอริธึม
  - วัดจากเวลาที่ใช้ประมวลผลจริง
  - วัดจากปริมาณหน่วยความจำที่ใช้
  - วัดจากจำนวนรอบของการประมวลผล
- Big-Oh O(g(n))
  - ขบวนการทางคณิตศาสตร์ ที่ให้คำตอบเป็นรูปฟังก์ชันพื้นฐาน
  - ใช้บอกความซับซ้อนของอัลกอริธึมที่ใช้ในการเขียนโปรแกรม
  - ใช้บอกความสัมพันธ์ระหว่าง ความเร็วของอัลกอริธึม กับจำนวนข้อมูล ว่ามีแนวโน้มเป็นอย่างไร เช่น O(1) , O(log n) , O(n) , O(n log n) , $O(n^2)$
  - อัลกอริธึมที่มี Big-Oh เท่ากัน ยังไม่สามารถบอกได้ว่าตัวใดดีกว่ากัน



Example

- ตัวอย่างการประเมินประสิทธิภาพอัลกอริธึม จากจำนวนรอบของคำสั่ง

• z = x; x = y; y = z; ...	// จำนวนรอบ = 1	O(1)
• for (i=1, i<= n, i++) {.....}	// จำนวนรอบ = c*n	O(n)
• for (i=1, i<= n, i=i+2) {.....}	//จำนวนรอบ = c*(n/2)	O(n)
• for (i=1, i<= n, i=i*2) {.....}	//จำนวนรอบ = c*$\log_2 n$	O($\log_2 n$)
• for (i=1; i<= n; i++) for (j=1; j<=n; j=j*2) {.....}	//จำนวนรอบ = c*(n$\log_2 n$)	O(n $\log_2 n$)
• for (i=1; i<= n; i++) for (j=1; j<=n; j++) {.....}	//จำนวนรอบ = c*(n*n)	O(n^2)
• for (i=1; i<= n; i++) for (j=i+1; j<=n; j++) {.....}	//จำนวนรอบ = c*n*(n-1)/2	O(n^2)

ตัวนับลดขนาดลงครึ่งหนึ่ง

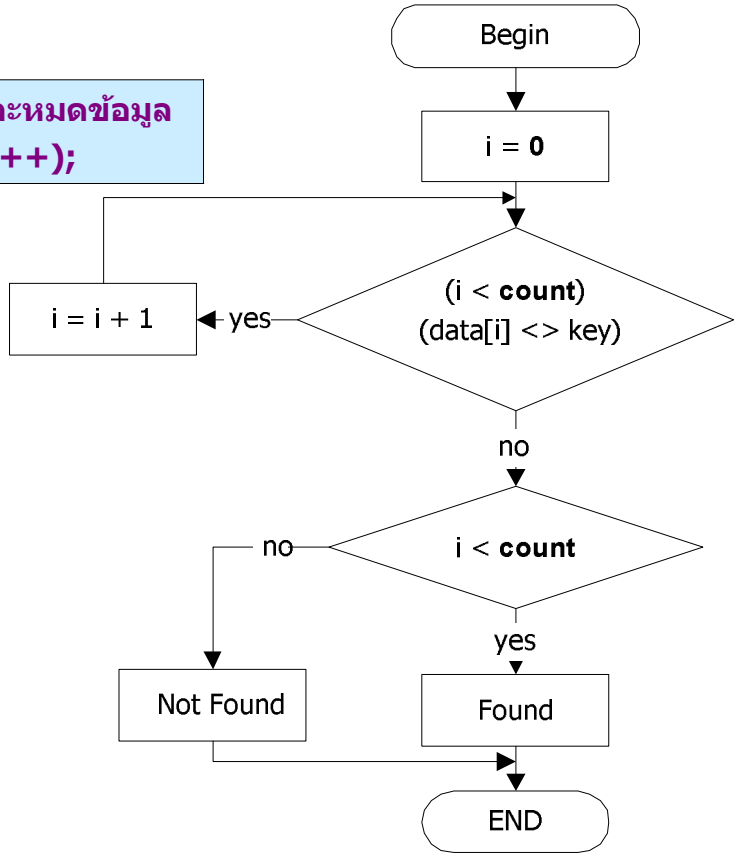
1. Linear Search(Unsorted)

- การค้นหาข้อมูลในอาร์เรย์ของตัวเลขจำนวนเต็ม ที่ไม่ได้เรียงลำดับ
  - สมมุติเก็บข้อมูลที่ตำแหน่ง 0 .. count -1
  - ให้ return ตำแหน่งที่เจอ และ return -1 ถ้าค้นไม่เจอ

```
int linear_search (int data[], int count, int key)
{ int i;
  i = 0;
  while ((i<count)&&(data[i] != key))
          i++;
   if (i < count)
       return i;
   else
       return -1;
}
```

เปรียบเทียบจนกว่าจะเจอ หรือจนกว่าจะหมดข้อมูล
for(i=0;i<count&&data[i]!=key;i++);

การหลุดออกจากวนรอบจะมีได้ 2 กรณีคือ
- เจอข้อมูลในตำแหน่งที่ i
  (เมื่อ data[i] == key หรือ i < count)
- ไม่เจอ(เมื่อ i >= count)



1.1 Improve Algorithm

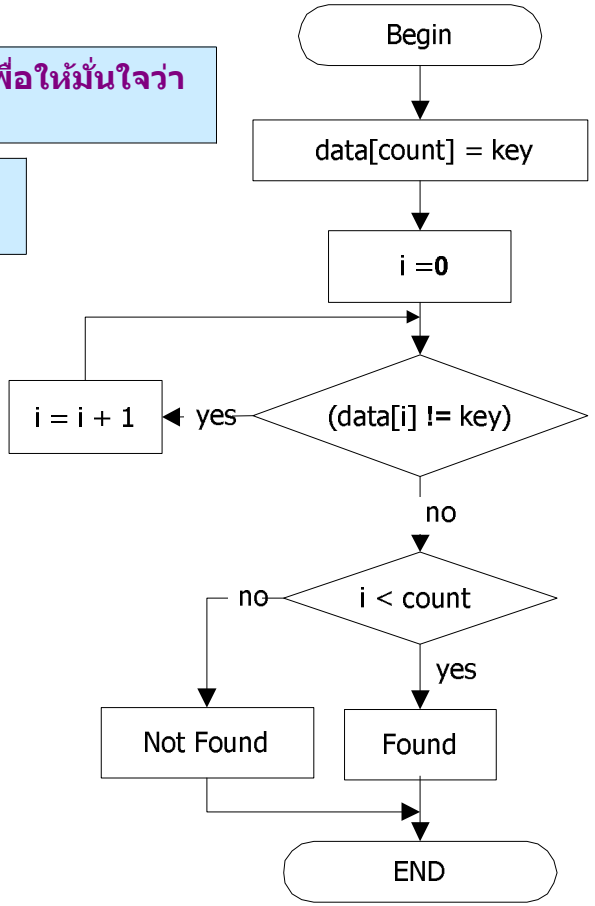
- ปรับปรุงเทคนิค เพื่อลดจำนวนเงื่อนไขที่ต้องเปรียบเทียบค้นหา

```
int dummy_search (int data[], int count, int key)
{ int i;
    data[count] = key;
    i = 0;
    while (data[i] != key)
            i++;
    if (i < count)
        return i;
    else
        return -1;
}
```

เพิ่มข้อมูลตัวหลอกเข้าต่อท้าย เพื่อให้มั่นใจว่าจะต้องค้นเจอ

เมื่อมั่นใจว่าต้องค้นเจอ จึงไม่ต้องตรวจสอบจำนวนข้อมูลล้นอาร์เรย์

ตรวจสอบว่าที่ค้นเจอเป็นตัวจริง หรือตัวหลอก



- ความซับซ้อนของอัลกอริธึม

ค่าเฉลี่ยเมื่อค้นเจอข้อมูล = (1+2+3+4+ ... + N)/N
= (N+1)/2
= *O(n)*

ค่าเฉลี่ยเมื่อค้นไม่เจอ = (N+1)
= *O(n)*

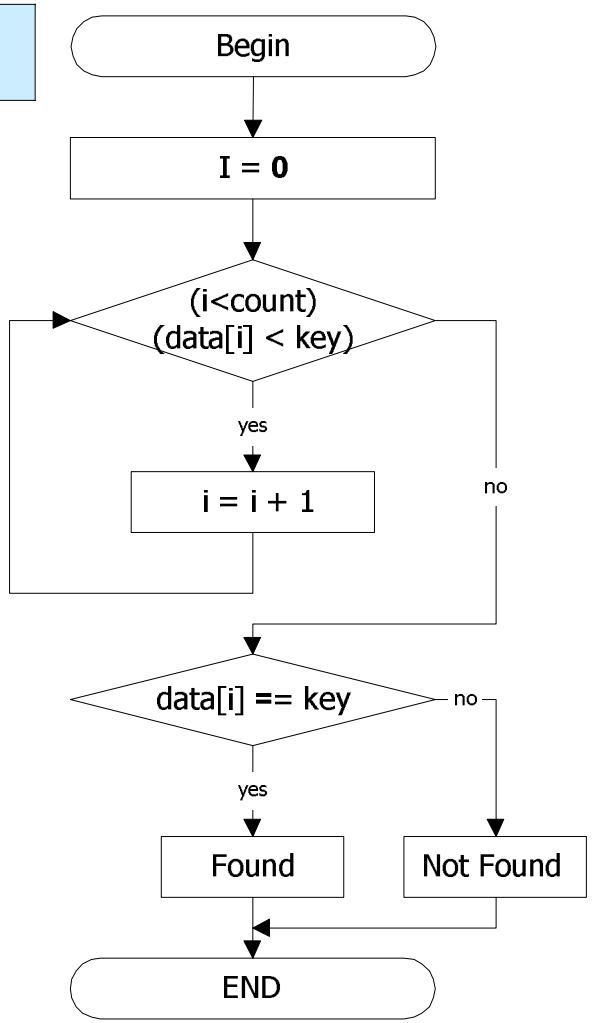
1.2 Linear Search (Sorted)

- ค้นหาข้อมูลในอาร์เรย์ของตัวเลขจำนวนเต็ม ที่เรียงลำดับแล้ว

```
int linear_search (int key)
{ int i;
  i = 0;
  while ( (i<count) && (data[i] < key) )
            i++;
  if (data[i]== key)
     return i ;
  else
     return -1;
}
```

เปรียบเทียบไปเรื่อย ๆ จนกว่าจะเจอ หรือแน่ใจว่าไม่เจอ

ถ้าค้นเจอ



- ความซับซ้อนของอัลกอริธึม

ค่าเฉลี่ยเมื่อค้นเจอข้อมูล = (1+2+3+4+ ... + N)/N
= (N+1)/2
= *O(n)*

ค่าเฉลี่ยเมื่อค้นไม่เจอ = (1+2+3+4+ ... + N)/N
= (N+1)/2
= *O(n)*

Analysis of algorithm

- การวิเคราะห์หาความซับซ้อนของอัลกอริธึม
  - กรณีที่ค้นเจอข้อมูล

    ถ้าข้อมูลที่ต้องการค้น มีโอกาสเจออยู่ในทุกตำแหน่งเท่ากัน

    (ค้นครั้งที่ 1 เจอในตำแหน่งที่ 1, ค้นครั้งที่ 2 เจอในตำแหน่งที่ 2, ...)

    ค่าเฉลี่ยเมื่อค้นเจอข้อมูล n ครั้ง = (1+2+3+4+ ... + n)/n

    = (n+1)/2

    จะเห็นว่าเวลา(ความซับซ้อน)ที่ใช้ในการค้นหาข้อมูลแปรเปลี่ยนตามค่า n ดังนั้น

    ความซับซ้อนของอัลกอริธึม = *O(n)* *สมการเส้นตรง*

  - กรณีที่ค้นไม่เจอข้อมูล
    - กรณีไม่เรียงลำดับ โปรแกรมจะต้องค้นหาจนถึงตัวที่ n จึงจะรู้ว่าไม่เจอ

    เวลาเมื่อค้นไม่เจอข้อมูล = n

    ความซับซ้อนของอัลกอริธึม = *O(n)*
    - กรณีเรียงลำดับ โปรแกรมจะต้องค้นหาจนถึงตัวที่มากกว่า key จะรู้ว่าไม่เจอ

    เวลาเฉลี่ยเมื่อค้นไม่เจอข้อมูล = (1+2+3+4+ ... + n)/n

    = (n+1)/2

    ความซับซ้อนของอัลกอริธึม = *O(n)* *สมการเส้นตรง*

2. Binary Search

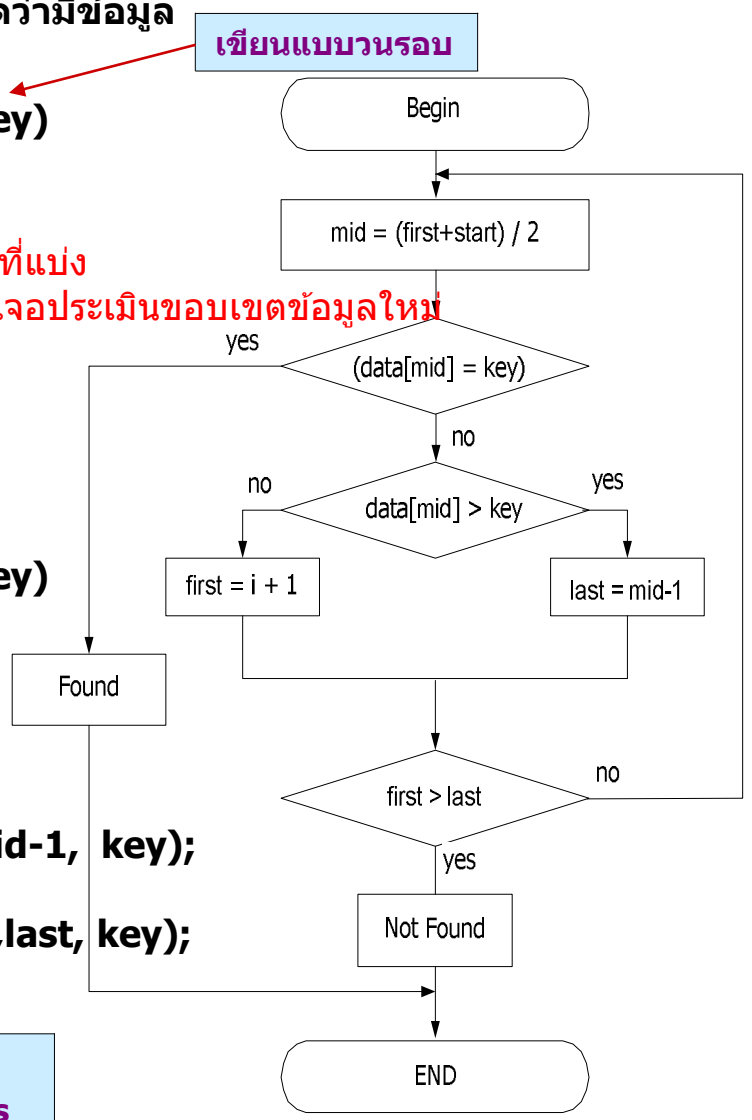
- ใช้เทคนิคแบ่งครึ่งข้อมูล แล้วเลือกค้นเฉพาะในกลุ่มที่คิดว่ามีข้อมูล
- การค้นข้อมูลแบบไบนารี ข้อมูลต้องเรียงลำดับเสมอ

เขียนแบบวนรอบ

```
int binary_search (int data[], int first, int last, int key)
{ int mid ;
  do {  mid = (first+last)/2; //แบ่งครึ่งข้อมูล
        if  (data[mid]==key) return mid; //ค้นตำแหน่งที่แบ่ง
        else if (data[mid]>key)  last = mid-1; //ถ้าไม่เจอประเมินขอบเขตข้อมูลใหม่
            else  first = mid+1;
     } while (first<=last);
  return -1;
}
```

เขียนแบบ recursion

```
int binary_search (int data[], int first, int last, int key)
{ int mid ;
  if (first > last)  return -1;  //Base Case
     else { mid  = (first + last) / 2 ; //Recursive Case
            if (data[mid] == key) return mid;
            else  if (data[mid] > key)
                    return binary_search(data, first, mid-1,  key);
                else
                    return binary_search (data, mid+1,last, key);
          }
}
```



C: มีฟังก์ชัน bsearch() อยู่ใน <stdlib.h>
Java : มีเมธอด Arrays.binarySearch() ใน Java.util.Arrays

Analysis of algorithm

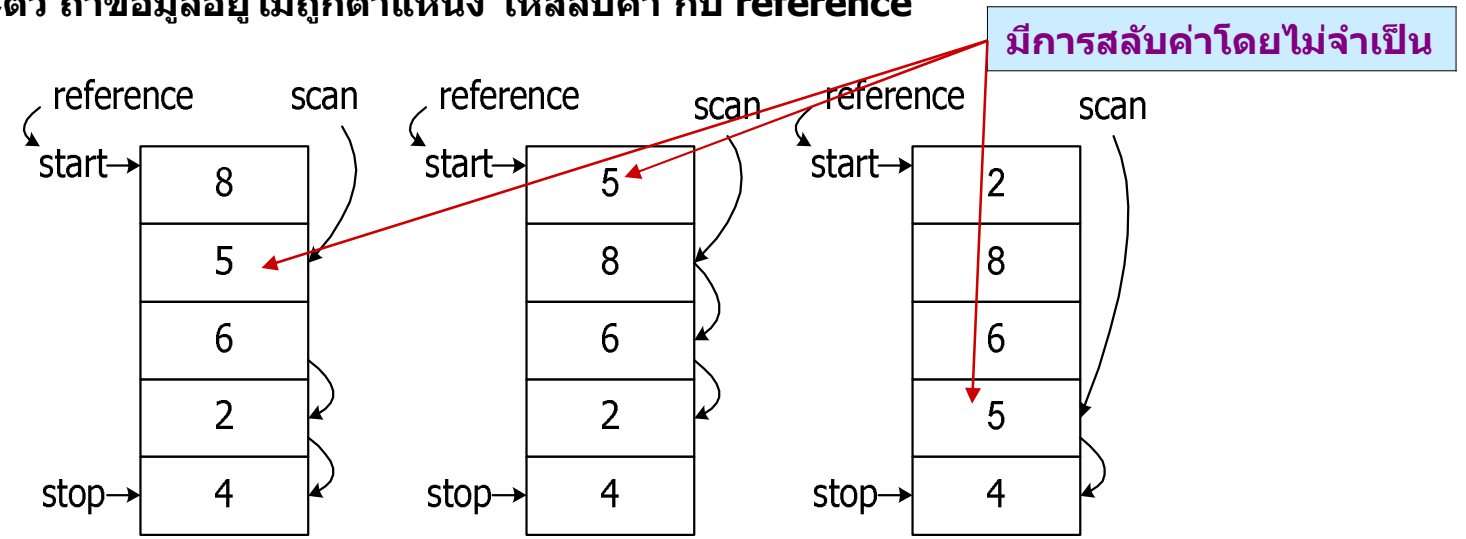
- การวิเคราะห์หาความซับซ้อนของอัลกอริธึม
  - ถ้าขนาดของข้อมูล n ลดลงครึ่งหนึ่งในทุกๆ รอบ
    - ขอบเขตในการค้นหาข้อมูล ครั้งที่ 1 = n
    - ขอบเขตในการค้นหาข้อมูล ครั้งที่ 2 = n/2
    - ขอบเขตในการค้นหาข้อมูล ครั้งที่ 3 = n/4
    - .....
    - ขอบเขตในการค้นหาข้อมูล ครั้งสุดท้าย = 1
    - จำนวนครั้งในการค้นหา = $(1 + 1 + 1 + 1 + ... + 1) = \log_2 n + 1$

  - ความซับซ้อนของอัลกอริธึม

$$T(n) = T(n/2) + O(1) = O(\log_2 n) + O(1)$$
$$= O(\log_2 n)$$

Sorting Data in Linear Arrays

- เรียงลำดับตัวเลขที่เก็บอยู่ในอาร์เรย์
  - เลือกข้อมูลตัวแรก(start) มาเป็น reference
    - เอาข้อมูลที่เหลือ (ตั้งแต่ reference+1 จนถึง stop) มาเปรียบเทียบกับ reference ทีละตัว ถ้าข้อมูลอยู่ไม่ถูกตำแหน่ง ให้สลับค่า กับ reference



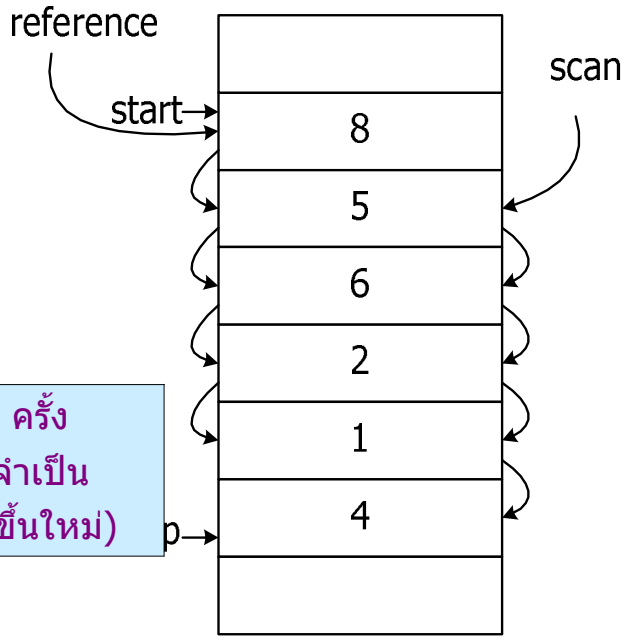
  - เลือกข้อมูลตัวถัดมา(ตัวที่สอง) มาเป็น reference แทนแล้วเปรียบเทียบกับข้อมูลตัวที่เหลือเช่นเดียวกับข้อมูลตัวแรก
- ทำเช่นเดิมจนกระทั่งครบทุกตัว ก็จะได้ข้อมูลที่เรียงลำดับ
- ความเร็วของการเรียงลำดับข้อมูลขึ้นอยู่กับ
  - จำนวนครั้งของการเปรียบเทียบข้อมูล
  - จำนวนครั้งของการย้ายข้อมูล (ข้อมูลที่มีจำนวนไบท์มากจะใช้เวลานานมากขึ้น)

Scan Sort

```
void scan_sort (int data[], int start, int stop)
{ int  i, j, x;
  for (i=start; i<=stop-1; i++) /* i = reference index */
    for (j=i+1; j<=stop; j++)   /* j = scan index */
      if (data[j] < data[i])          /* compare */
        swap(&data[i], &data[j]);     /* exchange */
}
```

```
void  swap(int *a, int *b)
{  int c;
       c = *a ; *a = *b ; *b = c ;
 }
```

มีการเปรียบเทียบ ∑(n-1) ครั้ง
มีโอกาสที่สลับค่าโดยไม่จำเป็น
(ข้อมูลเลื่อนลงแล้วเลื่อนขึ้นใหม่)



- **การวิเคราะห์หาความซับซ้อนของอัลกอริธึม**
  พิจารณาการวนรอบ ที่เกี่ยวกับจำนวนข้อมูล n
  มีรอบการวนรอบซ้อนกัน 2 ชั้น คือ
  วนรอบของ i ที่มีค่าตั้งแต่ 1 จนถึง n-1 มีจำนวน n - 1 ครั้ง
  ในแต่ละครั้งของ i จะมีวนรอบของ j จำนวน n-i รอบ
  จำนวนรอบที่ใช้เปรียบเทียบในการเรียงลำดับ
  = (n-1)+(n-2)+ … + 2 + 1 ที่ i=1, 2, 3, … , n-1
  = (n-1)*n/2
  = $O(n^2)$

Example 18 16 14 23 26 13 11 21 12 19

- ข้อมูล 10 ตัว ต้องเปรียบเทียบข้อมูล 45 ครั้ง มีการสลับข้อมูลทั้งที่ไม่จำเป็น

loop 1 (Change=4)

16 18 14 23 26 13 11 21 12 19
14 18 16 23 26 13 11 21 12 19
14 18 16 23 26 13 11 21 12 19
14 18 16 23 26 13 11 21 12 19
13 18 16 23 26 14 11 21 12 19
11 18 16 23 26 14 13 21 12 19
11 18 16 23 26 14 13 21 12 19
11 18 16 23 26 14 13 21 12 19
11 18 16 23 26 14 13 21 12 19

loop 2(Change=4)

11 16 18 23 26 14 13 21 12 19
11 16 18 23 26 14 13 21 12 19
11 16 18 23 26 14 13 21 12 19
11 14 18 23 26 16 13 21 12 19
11 13 18 23 26 16 14 21 12 19
11 13 18 23 26 16 14 21 12 19
11 12 18 23 26 16 14 21 13 19
11 12 18 23 26 26 14 21 13 19

ย้ายข้อมูลโดยไม่จำเป็น

loop 3 (Change=3)

11 12 18 23 26 16 14 21 13 19
11 12 18 23 26 16 14 21 13 19
11 12 16 23 26 18 14 21 13 19
11 12 14 23 26 18 16 21 13 19
11 12 14 23 26 18 16 21 13 19
11 12 13 23 26 18 16 21 14 19
11 12 13 23 26 18 16 21 14 19

loop 4 (Change=3)

11 12 13 18 26 23 16 21 14 19
11 12 13 16 26 23 18 21 14 19
11 12 13 14 26 23 18 21 16 19

loop 6 (Change=3)

11 12 13 14 16 23 26 21 18 19
11 12 13 14 16 21 26 23 18 19
11 12 13 14 16 18 26 23 21 19

loop 8(Change=2)

11 12 13 14 16 18 19 23 26 19
11 12 13 14 16 18 19 19 26 23

loop 5(Change=3)

11 12 13 14 23 26 18 21 16 19
11 12 13 14 18 26 23 21 16 19
11 12 13 14 16 26 23 21 18 19

loop 7(Change=3)

11 12 13 14 16 18 23 26 21 19
11 12 13 14 16 18 21 26 23 19
11 12 13 14 16 18 19 23 23 21

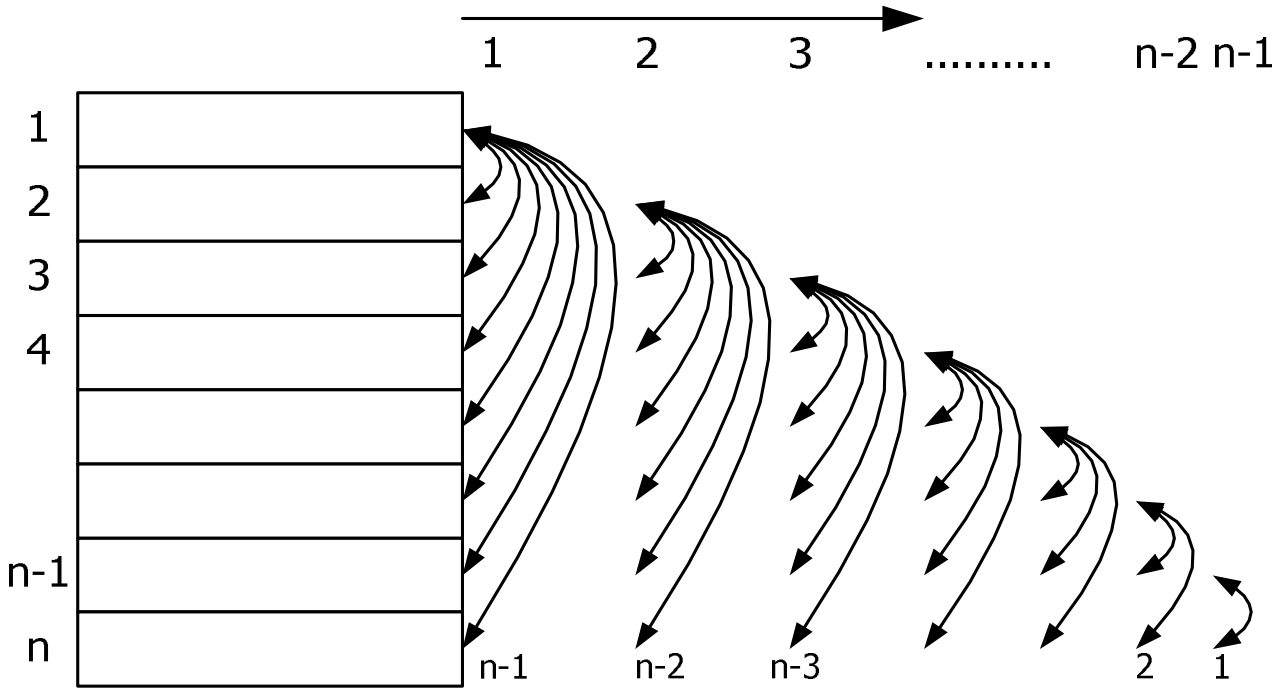
loop 9(Change=1)

11 12 13 14 16 18 19 21 23 26

เปรียบเทียบข้อมูล 45 ครั้ง
ย้ายข้อมูล 26*3=78 ครั้ง

1. Selection Sort

- ค้นหาข้อมูลตัวที่มีค่าน้อยที่สุดที่มีอยู่(กรณีต้องการเรียงลำดับจากน้อยไปมาก) ในแต่ละรอบออกมาเพื่อรอสลับค่า
- จำนวนครั้งในการเปรียบเทียบในแต่ละรอบเท่าเดิม แต่จำนวนครั้งในการสลับค่าเหลือรอบละครั้งเดียว จะได้ข้อมูลที่เรียงแล้วรอบละ 1 ตัว
- ทำซ้ำจนกว่าข้อมูลจะหมด



Selection Sort Method

- เรียงลำดับข้อมูลตัวเลขจำนวนเต็ม จากน้อยไปมากในอาร์เรย์ของ data[]

```
void  selection_sort (int data[], int start, int stop)
{ int i, j, min;
  for (i = start; i < stop; i++)          /* i = Reference */
      { min = i;                          /* k = smallest data position */
        for ( j = i+1; j <= stop; j++)
          { if (data[j] < data[min])
                min = j;  }     /* Keep smallest position */
        swap(&data[min], &data[i)] ; /* exchange data */
       }
}
```

จำตำแหน่งที่มีค่าน้อยที่สุดในแต่ละรอบไว้

มีการวนรอบเปรียบเทียบ ∑(n-1) รอบ
สลับค่า n-1 ครั้ง (รอบละ 1 ครั้ง)

สลับข้อมูลตัวอ้างอิงกับตัวที่มีค่าน้อยที่สุด
สลับค่ารอบละ 1 ครั้งเท่านั้น

- การวิเคราะห์หาความซับซ้อนของอัลกอริธึม
  - วนรอบของ i ที่มีค่าตั้งแต่ 1 จนถึง n-1 มีจำนวน n - 1 ครั้ง
    ในแต่ละครั้งของ i จะมีเปรียบเทียบข้อมูลโดยใช้ j ไม่เกิน n-i รอบ
  - จำนวนรอบเปรียบเทียบสูงสุด ที่ใช้ในการเรียงลำดับ
    = (n-1)+(n-2)+ … + 2 + 1 ที่ i=1, 2, 3, … , n-1
    = (n-1)*n/2
    = $O(n^2)$

Example 18 16 14 23 26 13 11 21 12 19

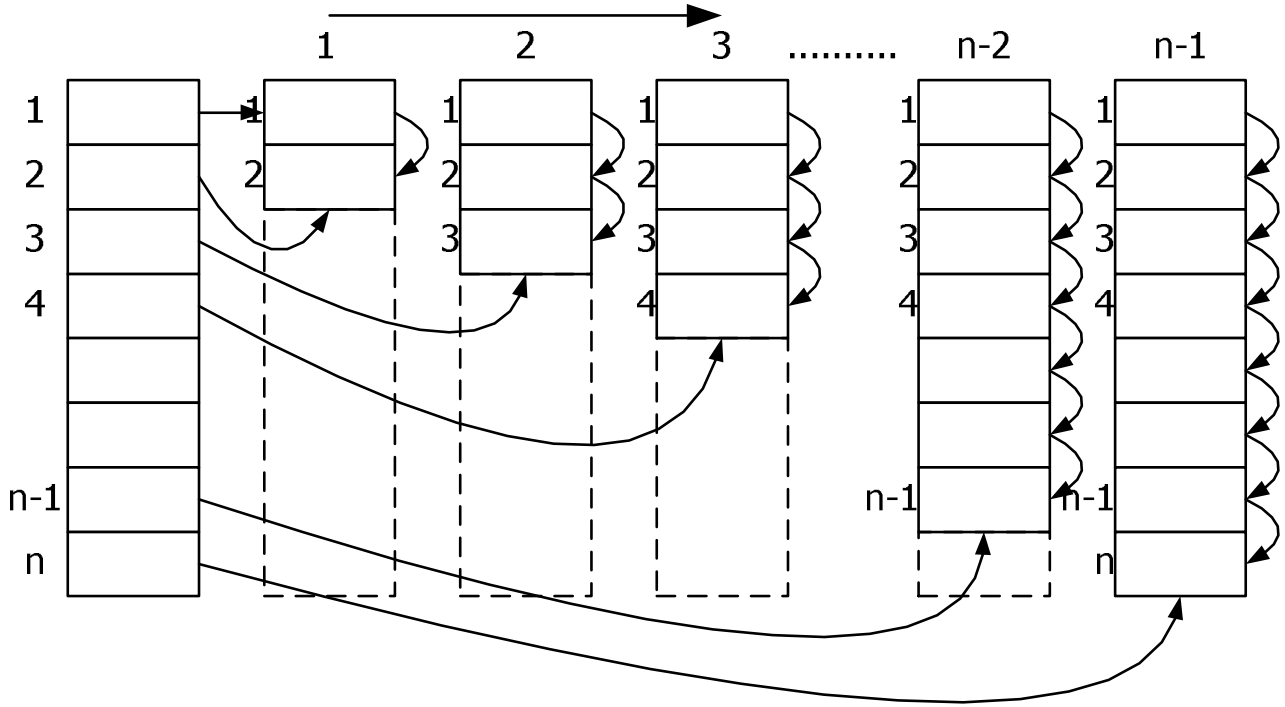
- เปรียบเทียบข้อมูล 45 ครั้ง แต่สลับข้อมูลเพียงรอบละ 1 ครั้ง

loop	start	Min	End
1	18 16 14 23 26 13 11 21 12 19	7	11 16 14 23 26 13 18 21 12 19
2	11 16 14 23 26 13 18 21 12 19	9	11 12 14 23 26 13 18 21 16 19
3	11 12 14 23 26 13 18 21 16 19	6	11 12 13 23 26 14 18 21 16 19
4	11 12 13 23 26 14 18 21 16 19	6	11 12 13 14 26 23 18 21 16 19
5	11 12 13 14 26 23 18 21 16 19	9	11 12 13 14 16 23 18 21 26 19
6	11 12 13 14 16 23 18 21 26 19	7	11 12 13 14 16 18 23 21 26 19
7	11 12 13 14 16 18 23 21 26 19	10	11 12 13 14 16 18 19 21 26 23
8	11 12 13 14 16 18 19 21 26 23	8	11 12 13 14 16 18 19 21 26 23
9	11 12 13 14 16 18 19 21 26 23	10	11 12 13 14 16 18 19 21 23 26

เปรียบเทียบ 45 ครั้ง ย้ายข้อมูล 9*3 =27 ครั้ง

2. Insertion Sort

- ดึงข้อมูลออกมาเพื่อเรียงลำดับทีละตัว โดยการค้นหาตำแหน่งที่เหมาะสม(ให้เกิดการเรียงลำดับ) สำหรับเก็บข้อมูลตัวนั้น แล้วนำข้อมูลตัวนั้นแทรกลงในตำแหน่งที่หาได้
- ใช้วิธีเลื่อนหาตำแหน่งข้อมูลที่เหมาะสมแทนการสลับค่า
- ทำซ้ำจนกว่าข้อมูลจะหมด



Insertion Sort Method

- เรียงลำดับข้อมูลตัวเลขจำนวนเต็ม จากน้อยไปมากในอาร์เรย์ของ data[]

```
void  insertion_sort (int data[], int start, int stop)
{ int i, j ;
  for (i = start+1; i <= stop; i++)    // เริ่มเปรียบเทียบตั้งแต่ตัวที่ 2 เป็นต้นไป
    { x = data[i];              // จำค่าตัวเลขที่เลือก
      for (j = i;((j>start)&&(x < data[j-1])); j--) // วนรอบเพื่อหาตำแหน่งที่เหมาะสม
          data[j] = data[j-1]; // เลื่อนข้อมูลลงมาทีละตัว
      data[j] = x; // เจอตำแหน่งที่เหมาะสมให้ใส่ค่าตัวเลขที่เลือกไว้
    }
}
```

- การวิเคราะห์หาความซับซ้อนของอัลกอริธึม
  - วนรอบของ i ที่มีค่าตั้งแต่ 2 จนถึง n มีจำนวน n - 1 ครั้ง
    ในแต่ละครั้งของ i จะมีเปรียบเทียบข้อมูลโดยใช้ j ไม่เกิน i-1 รอบ
  - จำนวนรอบเปรียบเทียบสูงสุด ที่ใช้ในการเรียงลำดับ
    = 1 + 2 + ... + (n-2)+(n-1) ที่ i = 2, 3, 4, ... , n
    = (n-1)*n/2
    = $O(n^2)$

มีการวนรอบเปรียบเทียบ ∑(n-1) รอบ
มีการย้ายค่า(Insert) ไม่เกิน ∑(n-1) ครั้ง

Example 18 16 14 23 26 13 11 21 12 19

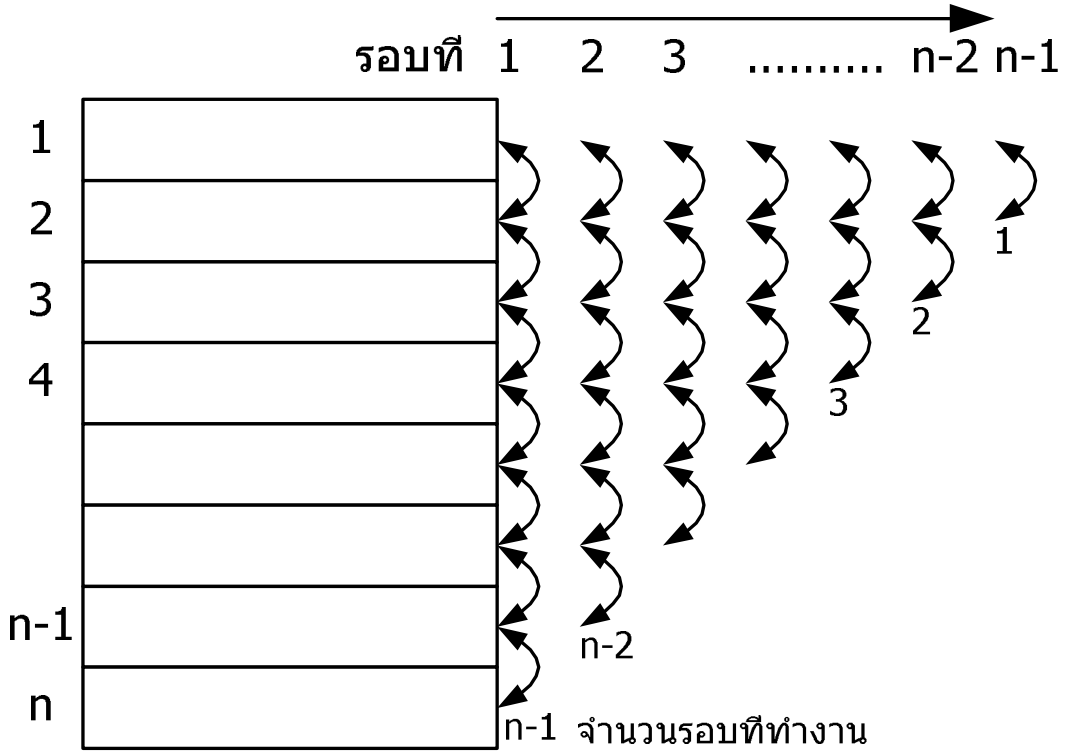
- ย้ายข้อมูลเท่าที่จำเป็น แต่ขยับทีละ 1 ตำแหน่ง

	Select data[i]	Sorted	
2	18 16 14 23 26 13 11 21 12 19	16 18 14 23 26 13 11 21 12 19	1+2
3	16 18 14 23 26 13 11 21 12 19	14 16 18 23 26 13 11 21 12 19	2+3
4	14 16 18 23 26 13 11 21 12 19	14 16 18 23 26 13 11 21 12 19	1+0
5	14 16 18 23 26 13 11 21 12 19	14 16 18 23 26 13 11 21 12 19	1+0
6	14 16 18 23 26 13 11 21 12 19	13 14 16 18 23 26 11 21 12 19	5+6
7	13 14 16 18 23 26 11 21 12 19	11 13 14 16 18 23 26 21 12 19	6+7
8	11 13 14 16 18 23 26 21 12 19	11 13 14 16 18 21 23 26 12 19	3+4
9	11 13 14 16 18 21 23 26 12 19	11 12 13 14 16 18 21 23 26 19	8+9
10	11 12 13 14 16 18 21 23 26 19	11 12 13 14 16 18 19 21 23 26	4+5

เปรียบเทียบ 31 ครั้ง ย้ายข้อมูล 26+18=44 ครั้ง

3. Bubble Sort

- เปรียบเทียบข้อมูลคู่ที่อยู่ติดกันไปเรื่อยๆ ถ้าพบว่าข้อมูลคู่ใดมีลำดับที่อยู่ไม่ถูกต้อง ให้สลับตำแหน่งกัน ทำซ้ำจนกว่าจะเรียงลำดับเสร็จ
- การสลับข้อมูลไม่เสียเปล่า (ไม่มีโอกาสที่ข้อมูลเที่เลื่อนลงแล้ว จะเลื่อนขึ้น)
- อาจเรียงลำดับเสร็จก่อนถึงการวนรอบสุดท้าย



Bubble Sort Method

- เรียงลำดับข้อมูลตัวเลขจำนวนเต็ม จากน้อยไปมากในอาร์เรย์ของ data[]

เปรียบเทียบเพื่อสลับตำแหน่งข้อมูลตัวที่อยู่ติดกัน

```
void  bubbleSort (int data[], int start, int stop)
{ int i, j;
  for (i = start; i <= stop-1; i++)
    for (j = stop; j > i; j--)
        if (data[j] < data[j-1])
            swap(&data[j], &dat[j-1]);
}
```

```
void  swap(int *a, int *b)
{  int c;
       c = *a; *a = *b; *b = c; }
```

ถ้าต้องการให้หลุดออกเมื่อไม่มีการสลับค่า

```
int doMore = 1;
  for (i = start; i <= stop-1 && doMore; i++) {
       doMore = 0;
       for (j = stop; j > i; j--)
            if (data[j] < data[j-1]) {
                swap(&data[j] ,&data[j-1]);
                doMore = 1;  } //end if
  }   // end for
}
```

- การวิเคราะห์หาความซับซ้อนของอัลกอริธึม

พิจารณาการวนรอบ ที่เกี่ยวกับจำนวนข้อมูล n
มีรอบการวนรอบซ้อนกัน 2 ชั้น คือ
วนรอบของ i ที่มีค่าตั้งแต่ 1 จนถึง n-1 มีจำนวน n - 1 ครั้ง
ในแต่ละครั้งของ i จะมีวนรอบของ j จำนวน n-i รอบ
จำนวนรอบที่ใช้เปรียบเทียบในการเรียงลำดับ
= (n-1)+(n-2)+ … + 2 + 1 ที่ i=1, 2, 3, … , n-1
= (n-1)*n/2
= $O(n^2)$

Example 18 16 14 23 26 13 11 21 12 19

- ขยับข้อมูลทีละ 1 ตำแหน่ง ทำเสร็จก่อนกำหนดได้

loop 1 (Change=7)

18 16 14 23 26 13 11 21 12 19
18 16 14 23 26 13 11 12 21 19
18 16 14 23 26 13 11 12 21 19
18 16 14 23 26 11 13 12 21 19
18 16 14 23 11 26 13 12 21 19
18 16 14 11 23 26 13 12 21 19
18 16 11 14 23 26 13 12 21 19
18 11 16 14 23 26 13 12 21 19
11 18 16 14 23 26 13 12 21 19

loop 2(Change=7)

11 18 16 14 23 26 13 12 19 21
11 18 16 14 23 26 13 12 19 21
11 18 16 14 23 26 12 13 19 21
11 18 16 14 23 12 26 13 19 21
11 18 16 14 12 23 26 13 19 21
11 18 16 12 14 23 26 13 19 21
11 18 12 16 14 23 26 13 19 21
11 12 18 16 14 23 26 13 19 21

loop 3 (Change=5)

11 12 18 16 14 23 26 13 19 21
11 12 18 16 14 23 26 13 19 21
11 12 18 16 14 23 13 26 19 21
11 12 18 16 14 13 23 26 19 21
11 12 18 16 13 14 23 26 19 21
11 12 18 13 16 14 23 26 19 21
11 12 13 18 16 14 23 26 19 21

loop 4 (Change=4)

11 12 13 18 16 14 23 26 19 21
11 12 13 18 16 14 23 19 26 21
11 12 13 18 16 14 19 23 26 21
11 12 13 18 16 14 19 23 26 21
11 12 13 18 14 16 19 23 26 21
11 12 13 14 18 16 19 23 26 21

loop 5(Change=3)

11 12 13 14 18 16 19 23 21 26
11 12 13 14 18 16 19 21 23 26
11 12 13 14 18 16 19 21 23 26
11 12 13 14 18 16 19 21 23 26
11 12 13 14 16 18 19 21 23 26

loop 6 (Unchange)

11 12 13 14 16 18 19 21 23 26

loop 7 (Unchange)

11 12 13 14 16 18 19 21 23 26

loop 8(Unchange)

11 12 13 14 16 18 19 21 23 26

loop 9 (Unchange)

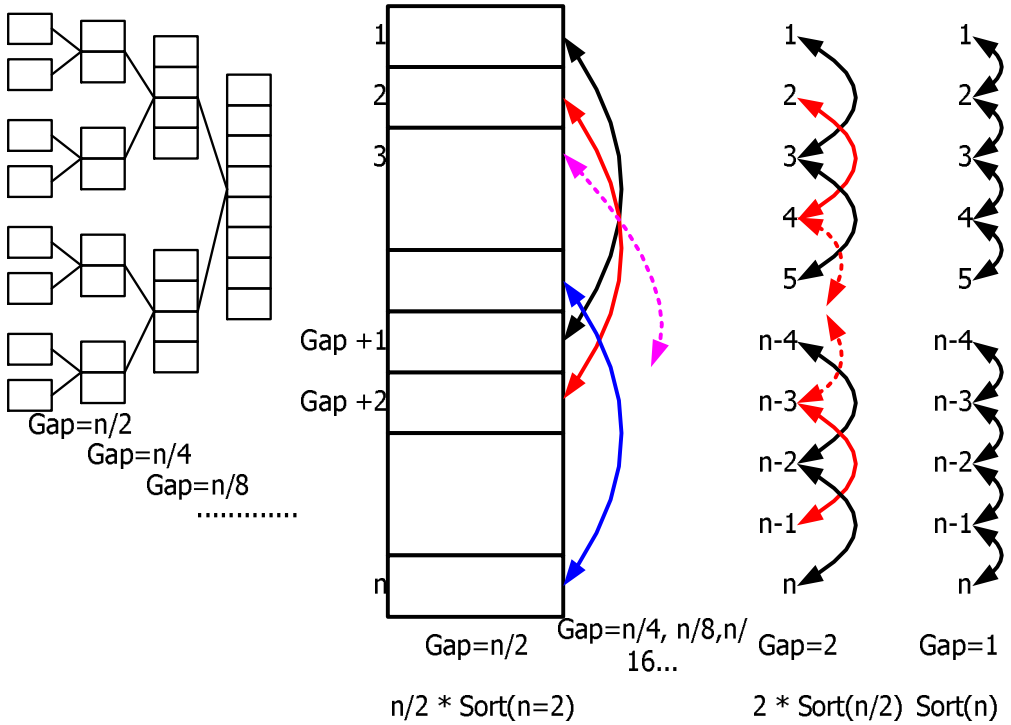
11 12 13 14 16 18 19 21 23 26

เปรียบเทียบ 45 ครั้ง
มีการย้ายข้อมูล 26*3 = 78 ครั้ง

- ข้อสังเกต โปรแกรมเริ่มไม่มีการสลับค่าตั้งแต่รอบที่ 6*(Unchange)*

4. Shell Sort

- เปรียบเทียบข้อมูลคู่ที่อยู่ห่างกันช่วงหนึ่งไปเรื่อยๆ แล้วสลับค่า เพื่อให้ข้อมูลกระโดดไปยังตำแหน่งที่เหมาะสมได้เร็วกว่า **Bubble Sort**
- ลดช่วงห่างการกระโดดลงครึ่งหนึ่งไปเรื่อยๆ จนกว่าจะเรียงลำดับเสร็จ
- รอบแรกจะมีข้อมูล **n/2** กลุ่ม(เท่ากับ **Gap**) กลุ่มละ **2** ตัว รอบถัดๆไป จำนวนกลุ่มจะลดลงครึ่งหนึ่ง ขณะที่จำนวนข้อมูลในกลุ่มเพิ่มขึ้นเป็น **2** เท่า
- รอบสุดท้ายจะเหลือกลุ่มเดียว (เหมือน **bubble Sort**) แต่จะเรียงลำดับเสร็จก่อนที่จะวนรอบเสร็จ



Shell Sort Method

```
void shellSort (int data[], int start, int stop)
{ int gap, changed, i;
  gap = stop-start+1;
  do { gap = gap/2;
       do { changed = 0;
            for (i =start; i < stop-gap+1; i++)
              if (data[i] > data[i+gap])
                { swap (&data[i], &data[i+gap])
                  changed = 1; }
          } while(changed == 1);
     } while(gap >1);
}
```

กำหนดระยะห่างของข้อมูล (จำนวนกลุ่ม)

ใช้ตัวแปร changed เพื่อตรวจสอบว่ามีการสลับค่า

ถ้ามีการสลับค่า แสดงว่ายังเรียงไม่เสร็จ ให้วนรอบทำจนกว่าจะไม่มีการสลับ

วนรอบทำจนกว่าจะเหลือกลุ่มเดียว

```
void  swap(int *a, int *b)
{  int c;
       c = *a; *a = *b; *b = c;
}
```

Example 18 16 14 23 26 13 11 21 12 19

- Gap = 5 เปรียบเทียบข้อมูล 5+5 =10 ครั้ง สลับค่า 4 ครั้ง

Set Gap = h = n/2 = 5, changed=0										
18	16	14	23	26	13	11	21	12	19	
1	2	3	4	5	h+1	h+2	h+3	h+4	h+5	
13	16	14	23	26	18	11	21	12	19	changed=1
13	11	14	23	26	18	16	21	12	19	changed=1
13	11	14	23	26	18	16	21	12	19	
13	11	14	12	26	18	16	21	23	19	changed=1
13	11	14	12	19	18	16	21	23	26	changed=1
Resume, Gap = 5, changed =0										
13	11	14	12	19	18	16	21	23	26	changed=0, Done

Gap = 2

- Gap = 2 เปรียบเทียบข้อมูล 8+8 = 16 ครั้ง สลับค่า 1 ครั้ง

Set Gap = 5/2 = 2, changed =0										
13	11	14	12	19	18	16	21	23	26	
13	11	14	12	19	18	16	21	23	26	
13	11	14	12	19	18	16	21	23	26	
13	11	14	12	19	18	16	21	23	26	
13	11	14	12	16	18	19	21	23	26	changed=1
13	11	14	12	16	18	19	21	23	26	
13	11	14	12	16	18	19	21	23	26	
13	11	14	12	16	18	19	21	23	26	
Resume, Gap = 2, changed =0										
13	11	14	12	16	18	19	21	23	26	changed=0, Done

Gap = 1

- Gap = 1 เปรียบเทียบข้อมูล 9+9 = 18 ครั้ง สลับค่า 2 ครั้ง

Gap =2/2 = 1 , changed =0										
11	13	14	12	16	18	19	21	23	26	changed=1
11	13	14	12	16	18	19	21	23	26	
11	13	12	14	16	18	19	21	23	26	changed=1
11	13	12	14	16	18	19	21	23	26	
11	13	12	14	16	18	19	21	23	26	
11	13	12	14	16	18	19	21	23	26	
11	13	12	14	16	18	19	21	23	26	
11	13	12	14	16	18	19	21	23	26	
11	13	12	14	16	18	19	21	23	26	

Changed = 0

- ทำต่อ Gap = 1 เปรียบเทียบข้อมูล 9+9 =18 ครั้ง สลับค่า 1 ครั้ง

Resume, Gap = 1, changed =0										
11	13	12	14	16	18	19	21	23	26	
11	12	13	14	16	18	19	21	23	26	changed=1
11	12	13	14	16	18	19	21	23	26	
11	12	13	14	16	18	19	21	23	26	
11	12	13	14	16	18	19	21	23	26	
11	12	13	14	16	18	19	21	23	26	
11	12	13	14	16	18	19	21	23	26	
11	12	13	14	16	18	19	21	23	26	
11	12	13	14	16	18	19	21	23	26	
Resume, Gap = 1, changed =0										
11	12	13	14	16	18	19	21	23	26	changed=0, Done

ตัวเลขชุดนี้มีเปรียบเทียบ 62 ครั้ง ย้ายข้อมูล 8*3 = 24 ครั้ง