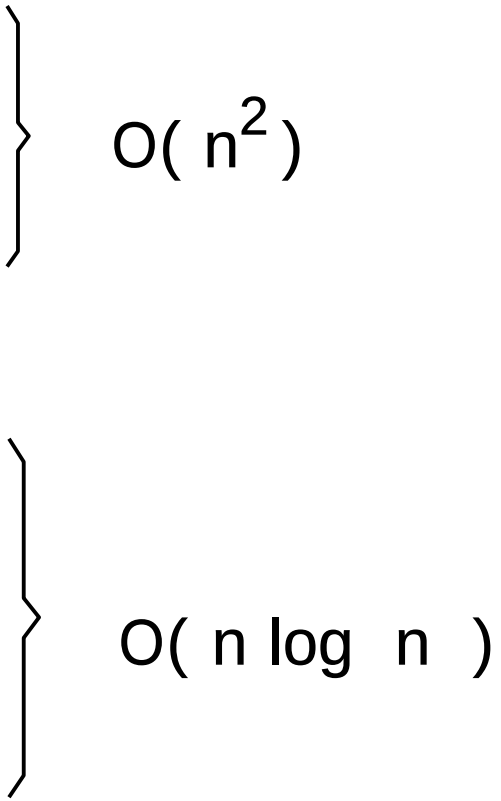
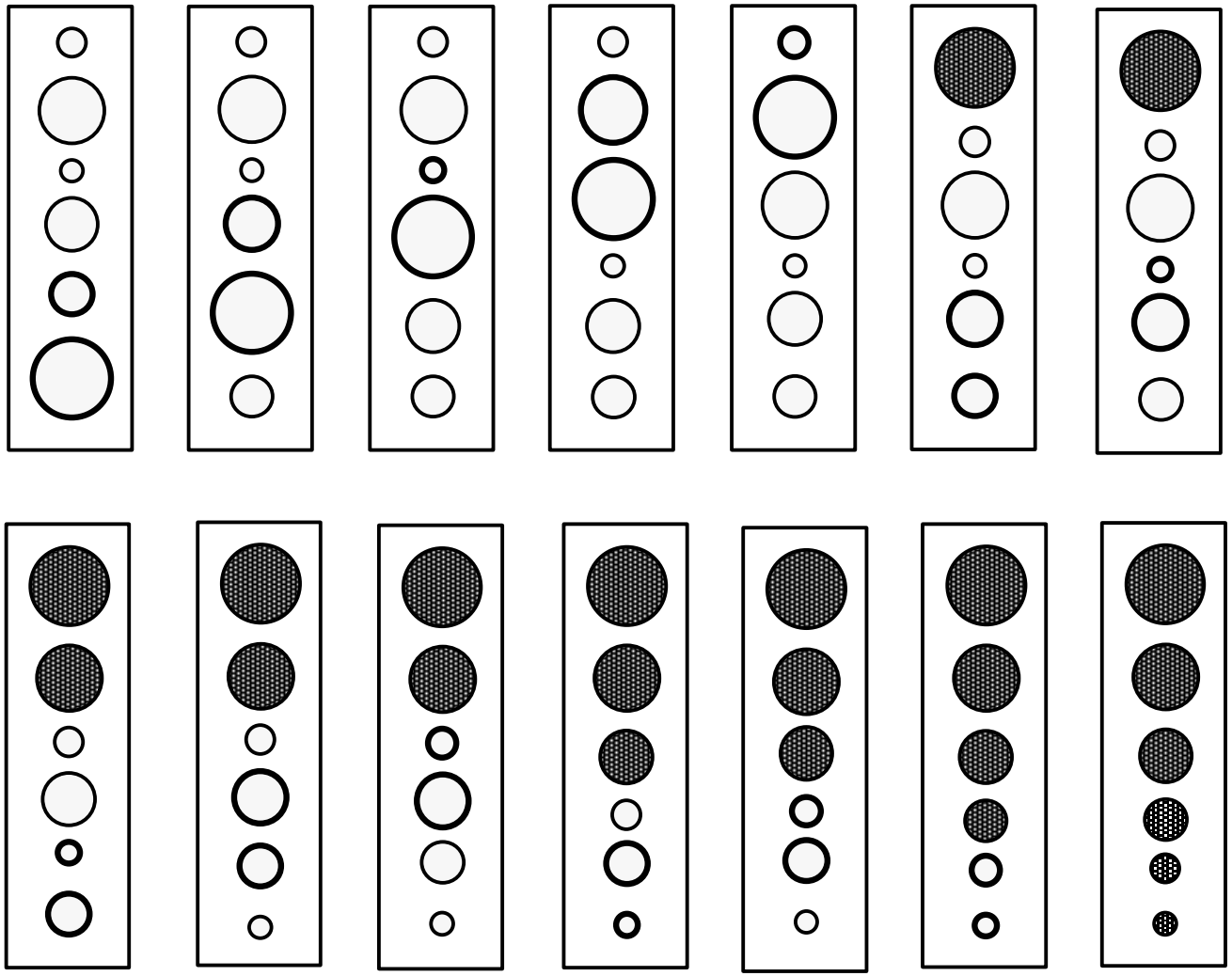
SORTING

การเรียงลำดับข้อมูล (Sorting)

- Bubble sort
- Insertion sort
- Selection sort
- Shell sort
- Mergesort
- Quicksort
- Treesort
- Heapsort



Bubble Sort

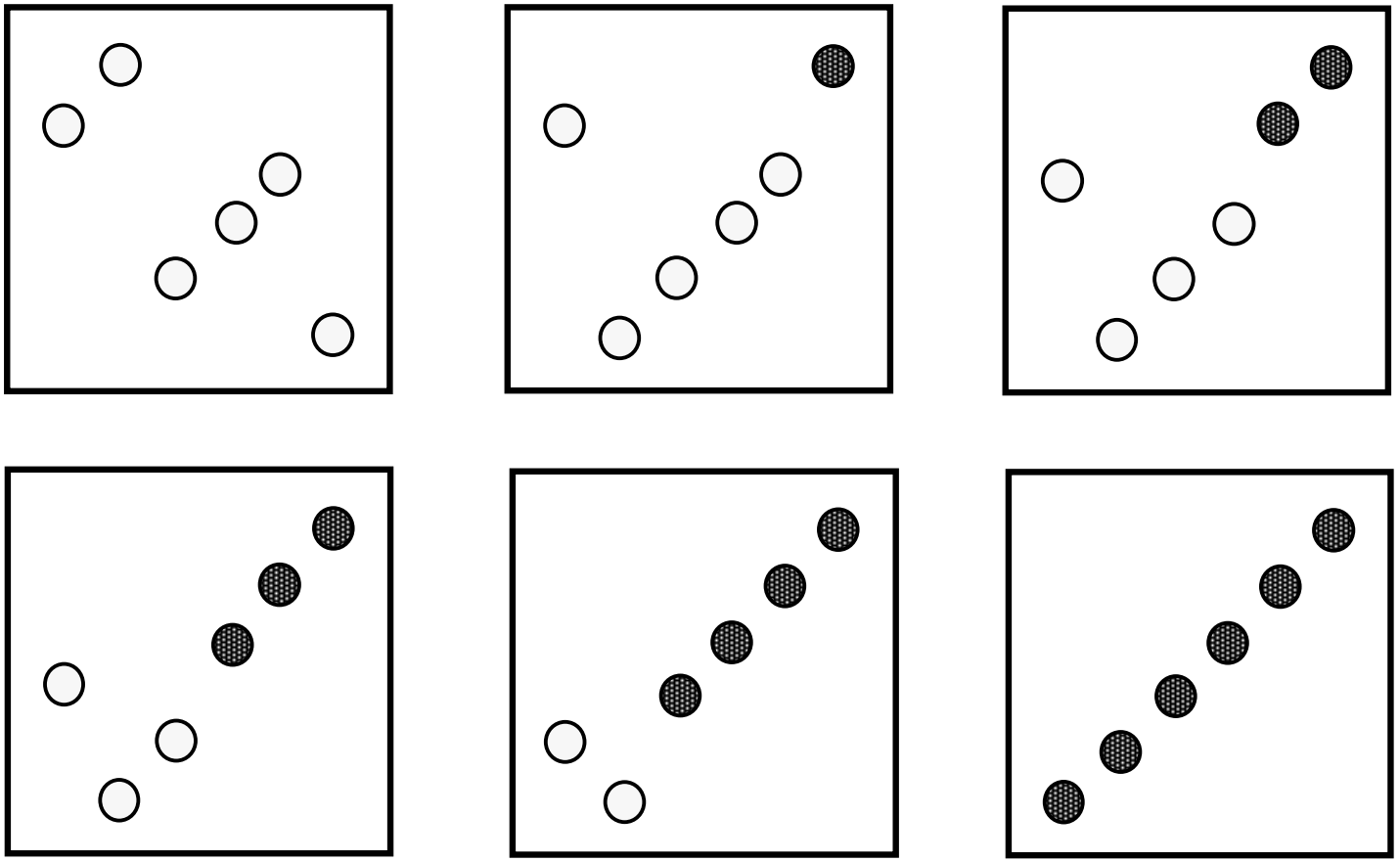


Bubble Sort

```
ListType  *BubbleSort( ListType *pList )
{
  int      i, k;

  for (k=1; k<=pList->count-1 k++) {
    for (i=pList->count-1; i>=k; i--) {
      if (pList->entry[i-1].key < pList->entry[i].key)
        Swap( i-1, i, pList );
    }
  }
  return( pList );
}
```

$$\sum_{k=1}^{n-1}(n-k) = \frac{n(n-1)}{2} = \frac{n^2}{2} - \frac{n}{2} = O(n^2)$$

Selection Sort



ในรอบที่ i, เลือกข้อมูลตัวที่คีย์มีค่ามากที่สุดจาก ข้อมูลตัวที่ 1 ถึงตัวที่ N-i+1 สลับข้อมูลตัวที่เลือกได้กับตัวที่ N-i+1

Selection Sort

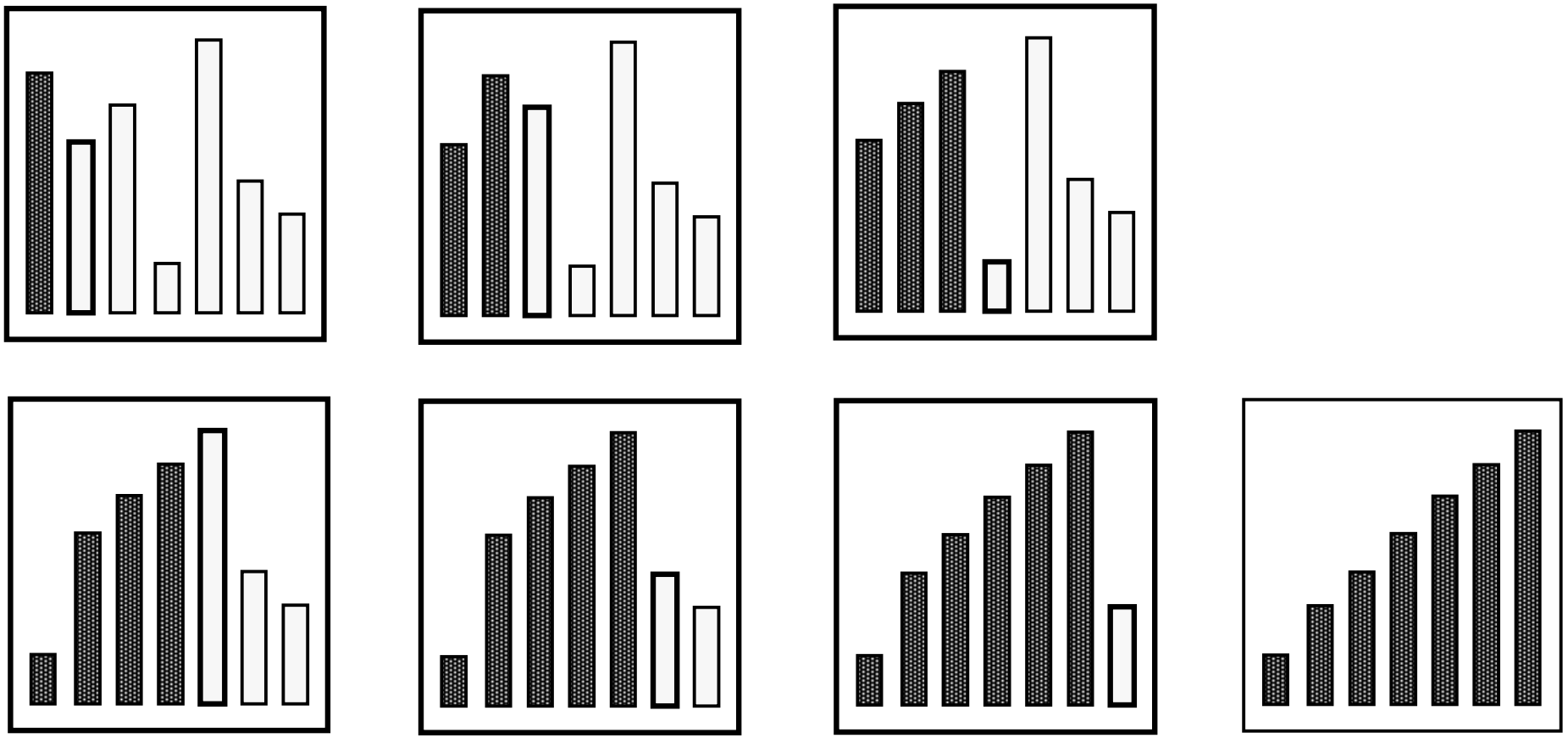
```
ListType *SelectionSort( ListType *pList )
{
  for (i=pList->count-1; i>0; i--) {
    j = MaxKey( 0, i, pList );
    Swap( i, j, pList );
  }
  return( pList );
}

int  MaxKey( int low, int high, ListType *pList )
{
  max = low;
  for (i=low+1; i<=high; i++) {
    if ( LT(pList->entry[max].key, pList->entry[i].key) )
      max = i;
  }
  return( max );
}
```

Analysis of Selection Sort

- ถ้ามีข้อมูลอยู่ n ตัว
- จะมีการเรียกฟังก์ชัน Swap เป็นจำนวน n-1 ครั้ง
  นั่นคือจะมีการย้ายข้อมูลจำนวน 3(n-1) ครั้ง
- ฟังก์ชัน MaxKey ถูกเรียก n-1 ครั้ง ดังนั้นจึงมีการเปรียบเทียบคีย์เป็นจำนวน

$$\sum_{k=1}^{n-1}(n-k) = \frac{n(n-1)}{2} = \frac{n^2}{2} - \frac{n}{2} = O(n^2)$$

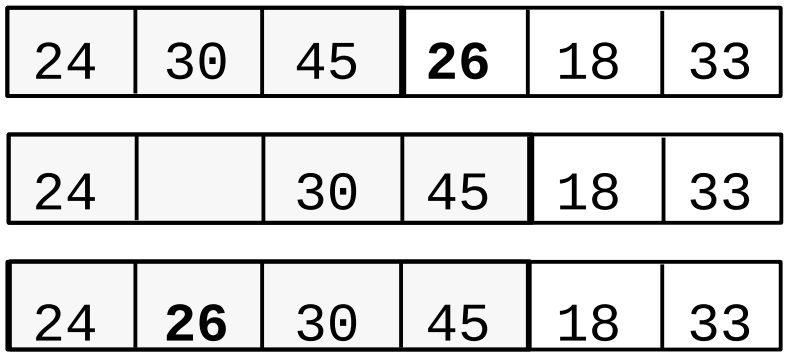
Insertion Sort



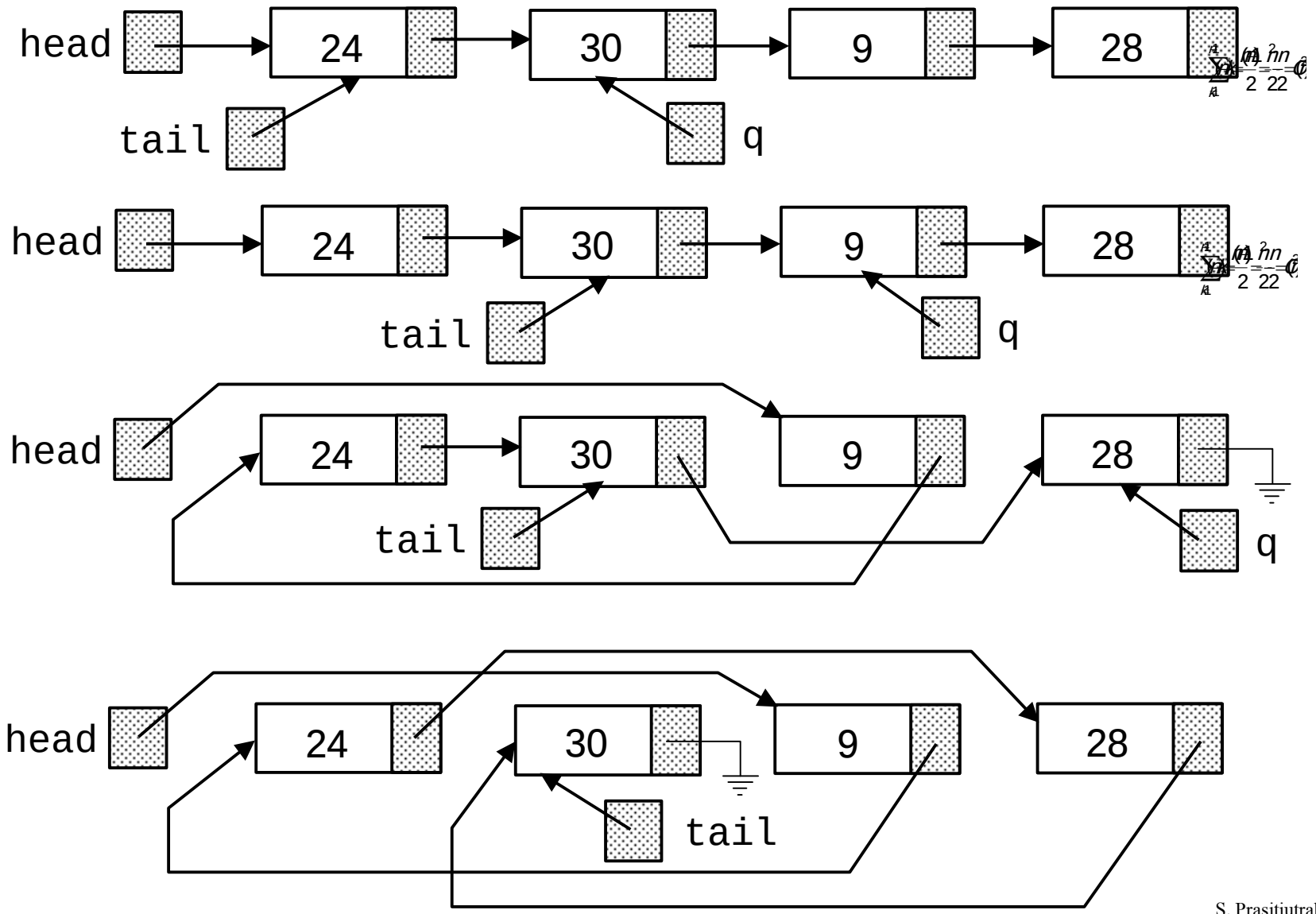
ถ้าข้อมูล *i*-1 ตัวแรกเรียงลำดับแล้ว นำข้อมูลตัวที่ *i* มาแทรกไว้ ในตำแหน่ง ที่เหมาะสม เพื่อทำให้ข้อมูล *i* ตัวแรกเรียงลำดับถูกต้อง

Insertion Sort : Contiguous Version

```
ListType *InsertionSort( ListType *pList )
{
  for (k=1; k<=pList->count-1; k++) {
    if ( LT(pList->entry[k].key, pList->entry[k-1].key) ) {
      tmp = pList->entry[k];
      for (j=k-1; j>=0; j--) {
        pList->entry[j+1] = pList->entry[j];
        if ( j==0 ||
             LE( pList->entry[j-1].key, tmp.key) )  break;
      }
      pList->entry[j] = tmp;
    }
  }
  return( pList );
}
```

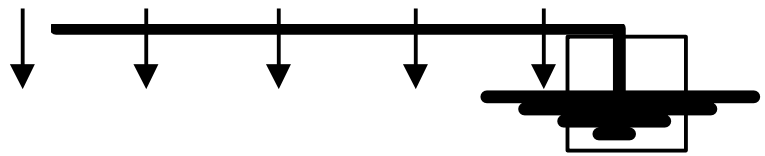


Insertion Sort : Linked Version



Analysis of Insertion Sort

- ขณะที่พิจารณาข้อมูลตัวที่ k นั้น ตำแหน่งที่ข้อมูลตัวที่ k จะแทรกได้มีอยู่ k ตำแหน่ง



- กำหนดให้ โอกาสที่จะแทรกข้อมูลตัวที่ k ที่ตำแหน่งใดๆมี เท่าๆกัน ดังนั้น จำนวนรอบภายในเฉลี่ย เท่ากับ

Analysis of Insertion Sort

- เมื่อพิจารณาข้อมูลตัวที่ k
  - ความน่าจะเป็นที่ข้อมูลตัวที่ k จะอยู่ตำแหน่งเดิม = $1 / k$
  - ความน่าจะเป็นที่ข้อมูลตัวที่ k จะไม่อยู่ตำแหน่งเดิม = $(k\text{-}1) / k$
- จำนวนการเปรียบเทียบเฉลี่ย เมื่อพิจารณาข้อมูลตัวที่ k

- จำนวนการย้ายข้อมูลเฉลี่ย เมื่อพิจารณาข้อมูลตัวที่ k

- จำนวนการเปรียบเทียบ / การย้ายข้อมูล ทั้งหมด

Selection Sort vs. Insertion Sort

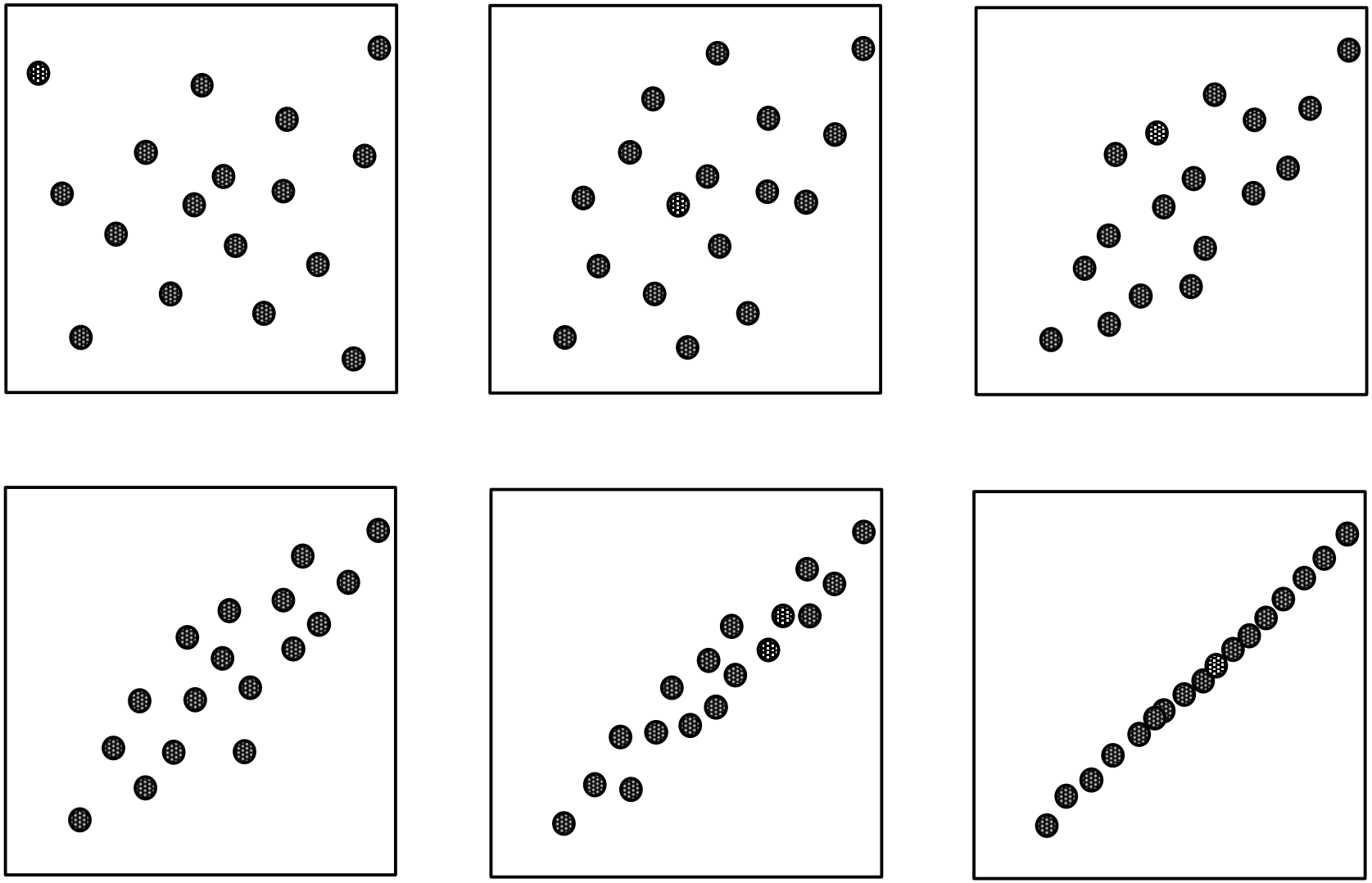
	การย้ายข้อมูล	การเปรียบเทียบคีย์
Selection Sort	$3.0n + O(1)$	$0.5n + O(n)$ 2
Insertion Sort	$0.25n + O(n)$	$0.25n + O(n)$
	2	2

ปัญหา: ลองหาลักษณะลำดับเริ่มต้นของคีย์ที่ทำให้การเรียงลำดับโดยวิธี selection sort และ insertion sort เร็วที่สุด และสำหรับกรณีที่ช้าที่สุด

Shell Sort

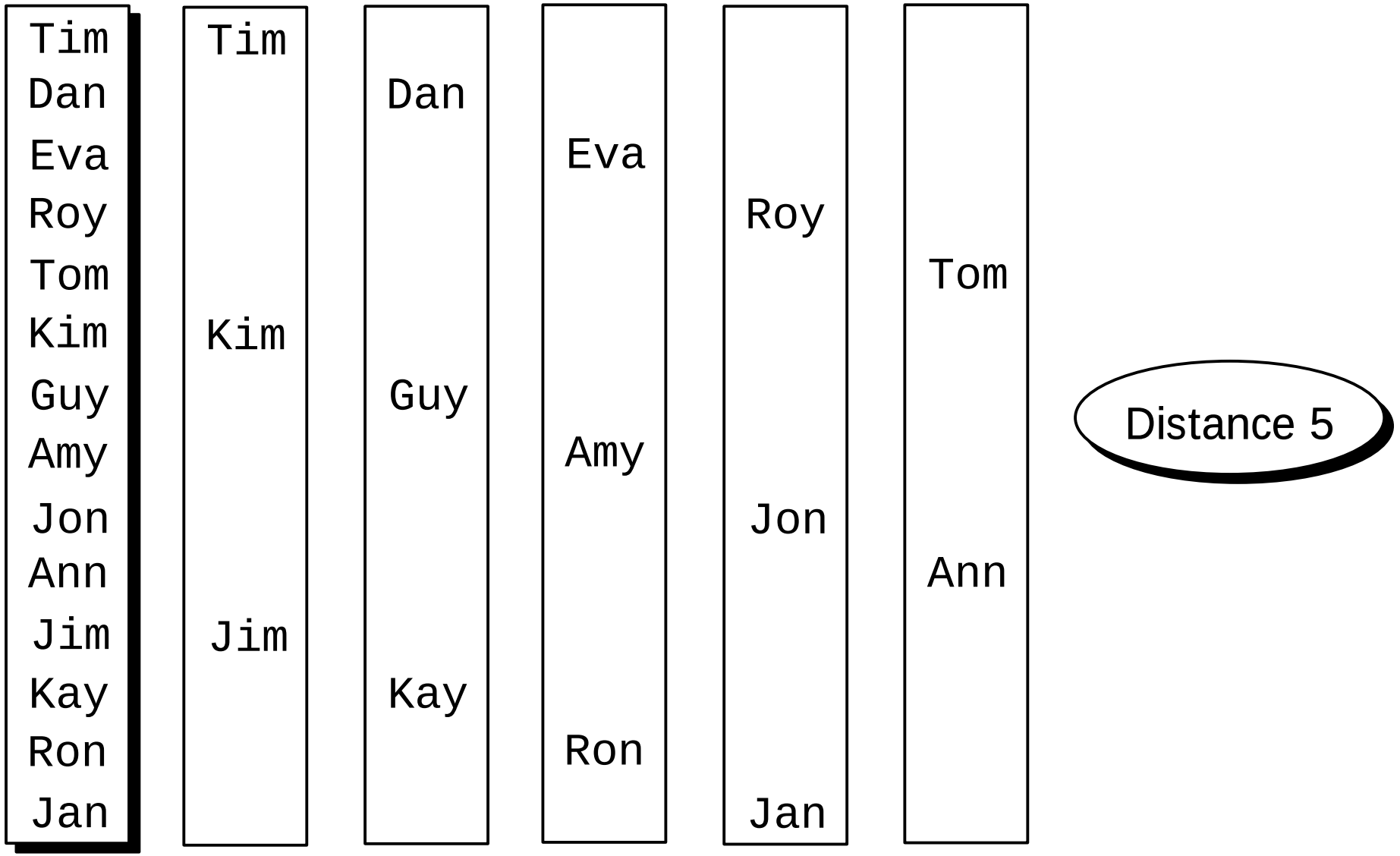
- ระหว่างการเรียงลำดับโดยวิธี Insertion sort นั้น จะต้องมีการเคลื่อนย้ายข้อมูล เพื่อให้การแทรกข้อมูลถูกตำแหน่ง
- โดยทั่วไปลำดับของข้อมูลยิ่งเรียงมาก ก็ทำให้การเรียงลำดับโดยวิธี Insertion sort เร็วขึ้นตาม
- ลักษณะเด่นของ Shell sort ก็คือการค่อยๆทำให้ลำดับของข้อมูลเรียง มากขึ้นเรื่อยๆ จนกระทั่งเรียงลำดับทั้งหมดในที่สุด โดยการใช้ Insertion sort ในระหว่างการเรียงลำดับ

Shell Sort

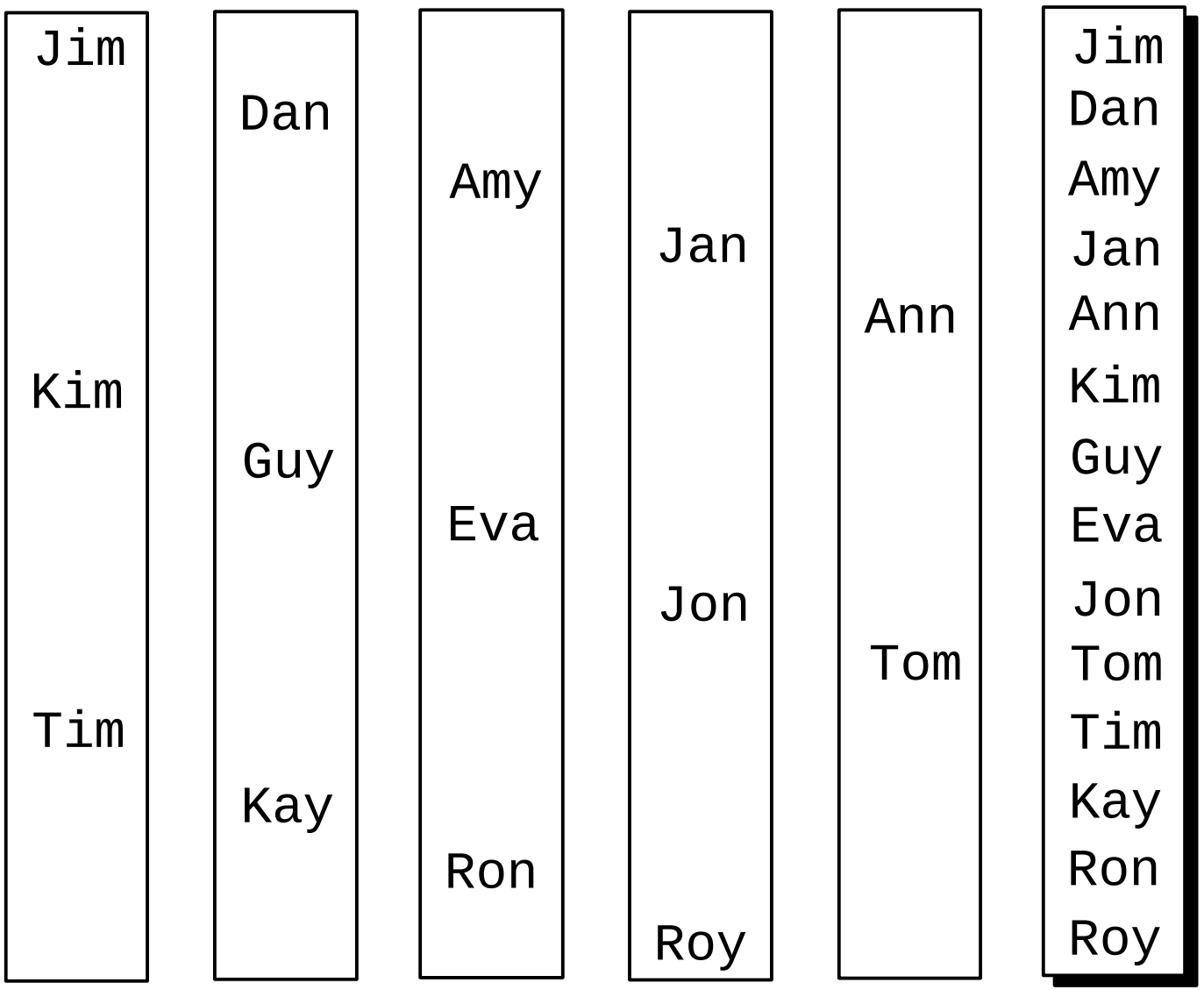


S. Prasitjutrakul 1994

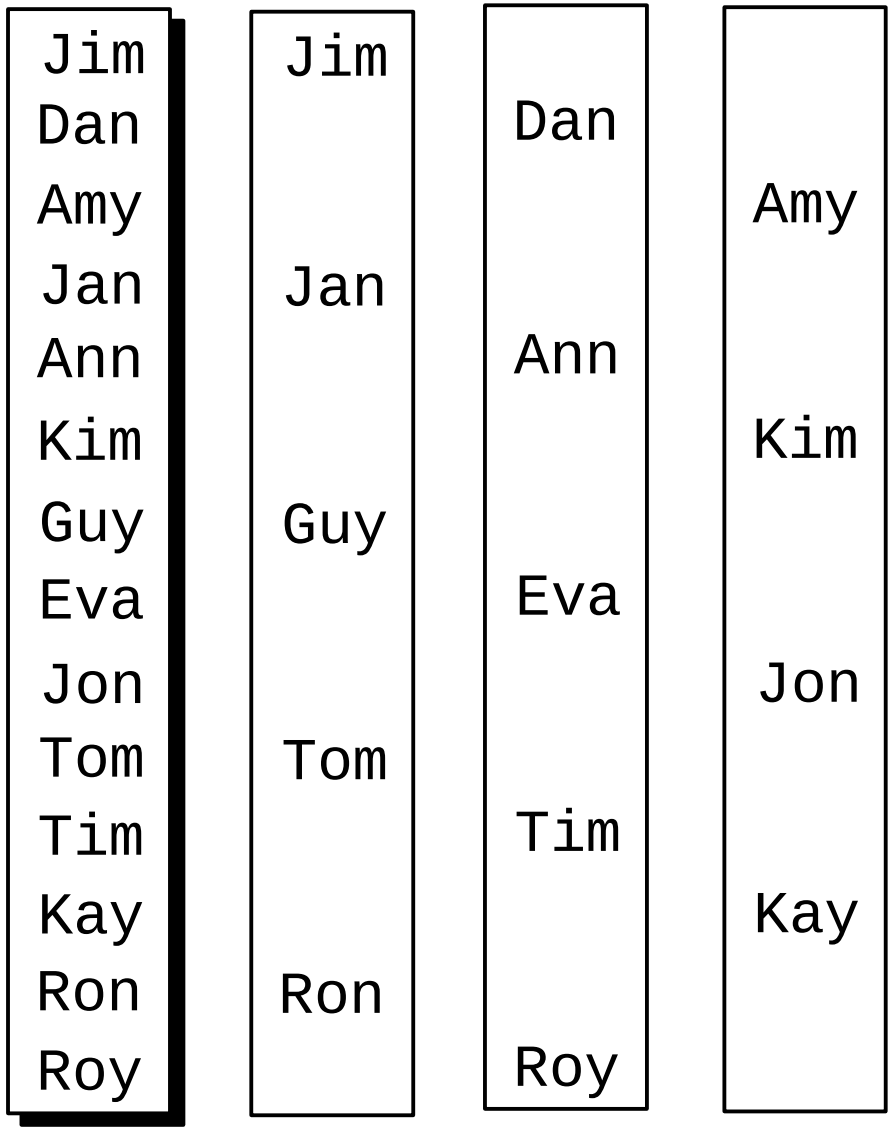
Shell Sort : An Example



Shell Sort : An Example

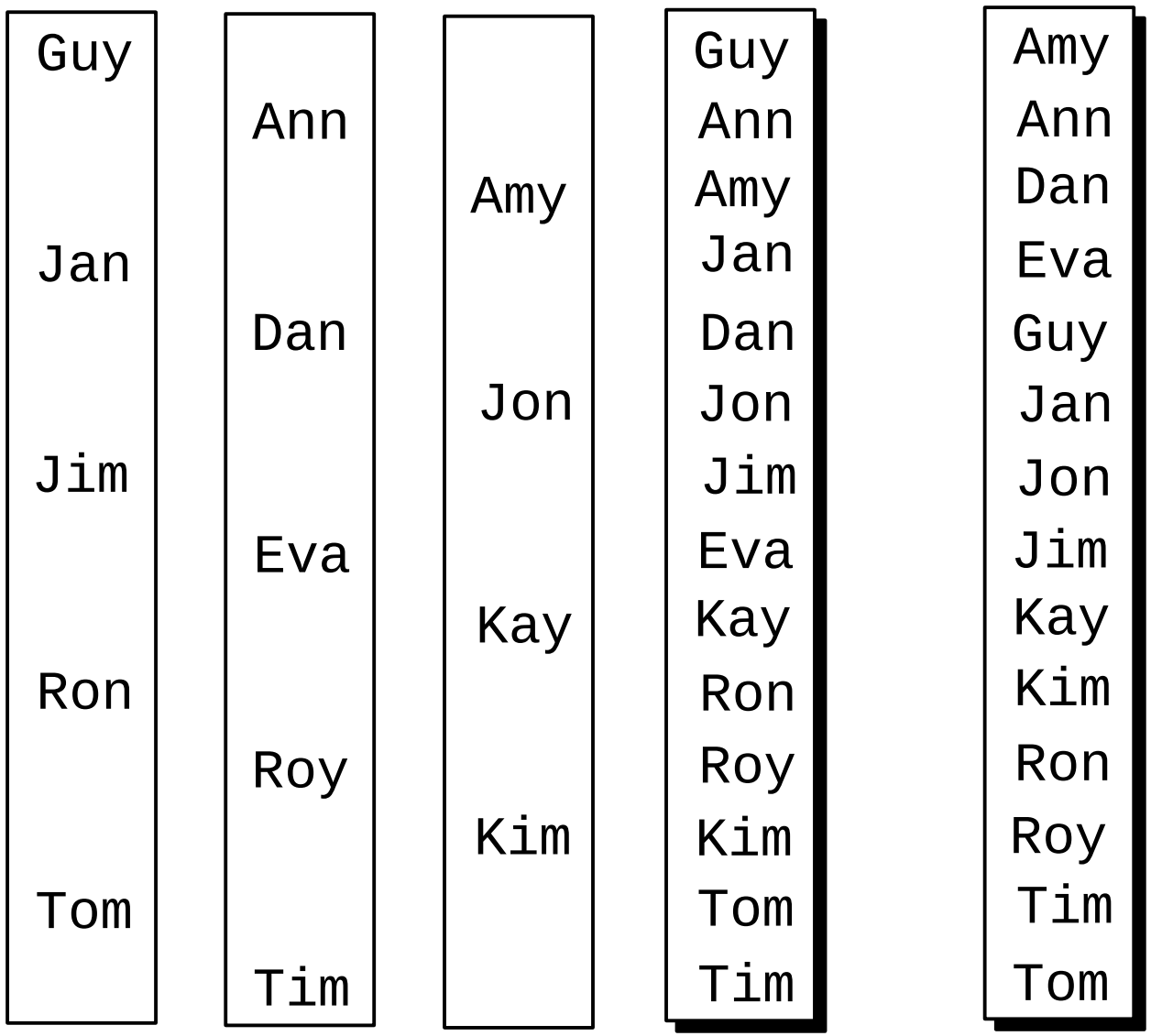


Shell Sort : An Example



Distance 3

Shell Sort : An Example



Distance 1

Shell Sort

```
ListType  *ShellSort( ListType *pList )
{
  inc  = pList->count;
  do {
    inc = inc / 3 + 1;
    for (i=0; i<inc; i++)
      pList = InsertionSortX( i, inc, pList );
  } while ( inc > 1 );
  return( pList );
}
```

ไม่มีกฎตายตัวในการเลือกค่า `inc`

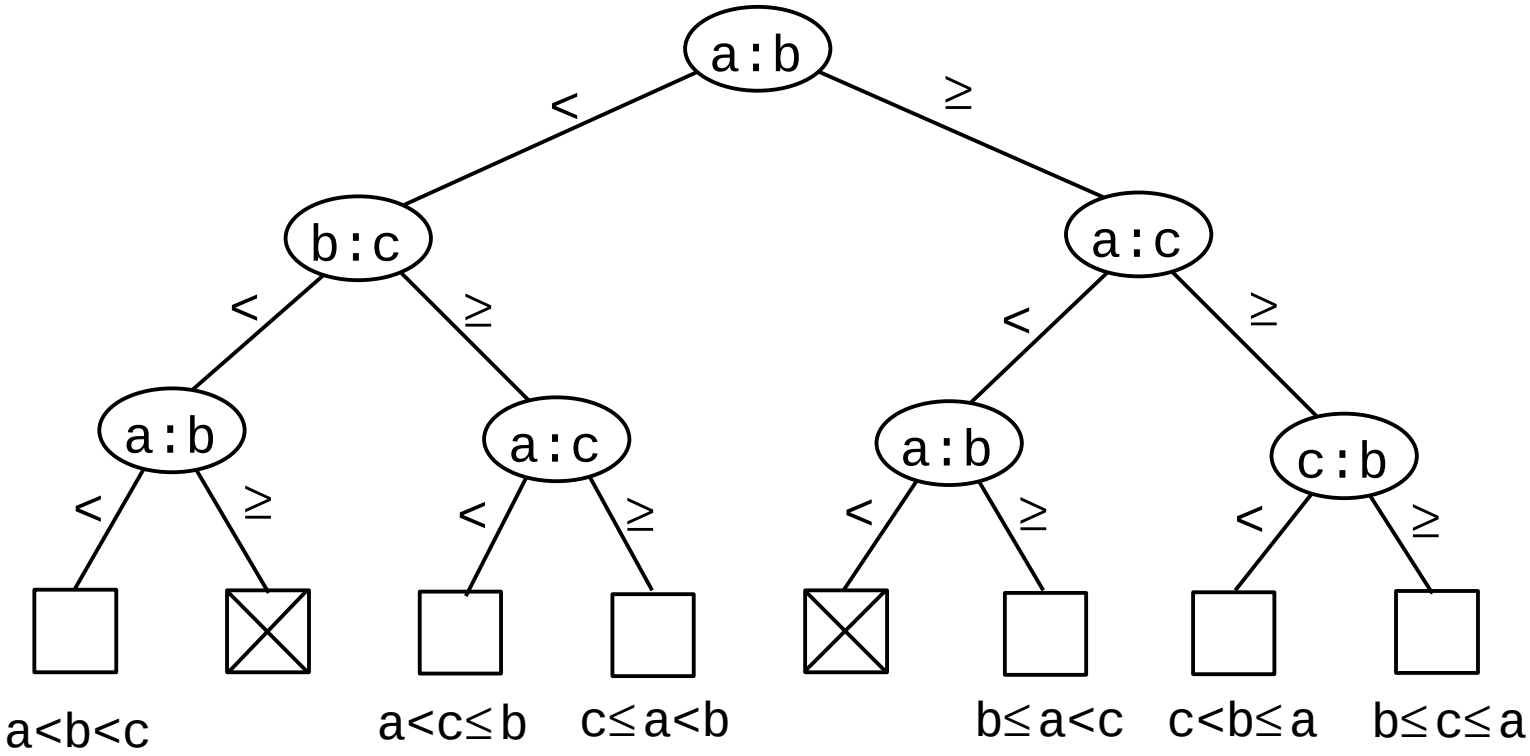
แต่ควรพยายามทำให้การเปรียบเทียบ คีย์ชุดซ้ำกันมีน้อยๆ

จากการทดลองพบว่า เมื่อมีข้อมูลจำนวนมาก จำนวนการย้ายข้อมูลจะอยู่ในช่วง

$$n \quad \text{ถึง} \quad 1.6n^{1.25}$$

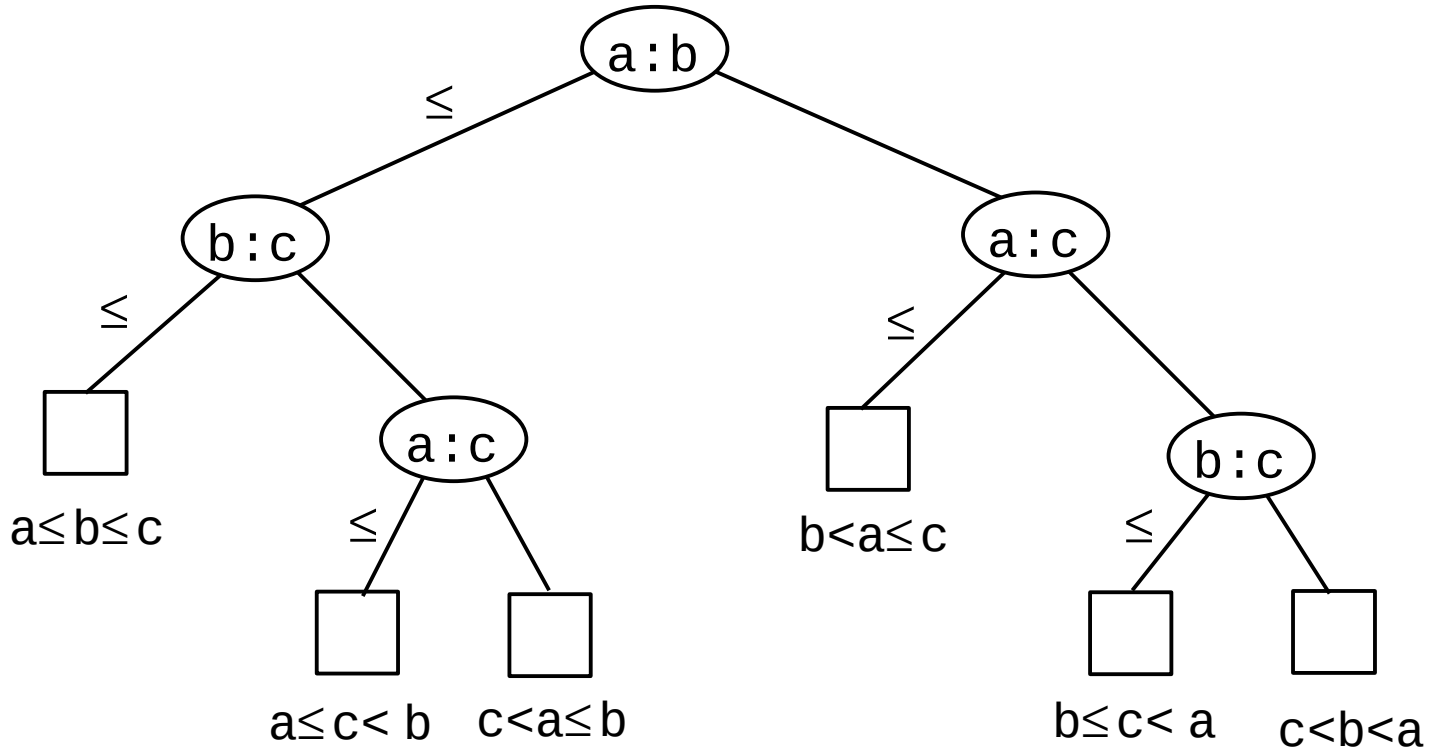
($n^{1.25}$ คือ n คือจำนวนข้อมูล)

Comparison Trees : Selection Sort



a b c

Comparison Trees : Insertion Sort



a b c

ขอบเขตล่างของการเรียงลำดับ

- พิจารณาเฉพาะวิธีการเรียงลำดับที่ใช้การเปรียบเทียบข้อมูลที่ละคู่
- ถ้ามีข้อมูล n ตัว ต้นไม้เปรียบเทียบของการเรียงลำดับจะมีอย่างน้อย $n!$ ใบ
- ดังนั้น การเรียงลำดับวิธีใดๆ
  จำนวนการเปรียบเทียบมากสุดต้องมีอย่างน้อย (ความสูงของต้นไม้ที่สมดุล)

$$\lceil \log_2 n! \rceil \text{ ครั้ง}$$

และ โดยเฉลี่ยแล้วจะต้องเปรียบเทียบอย่างน้อย

$$\log_2 n! \text{ ครั้ง}$$

Stirling's Formula

$$n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$$

$$\log_2 n! = \left(n+\frac{1}{2}\right)\log_2 n - \log_2 e^n + O(1)$$

$$= n\log_2 n - 1.44n + O(1)$$

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

สรุปได้ว่า

ไม่มีวิธีการเรียงลำดับใดๆ (ที่ใช้การเปรียบเทียบคีย์ทีละคู่) ที่มี worst-case time complexity ที่ดีกว่า $O(n \log n)$