การจัดเรียงข้อมูล

Sorting

การเรียงลำดับข้อมูล

การจัดเรียงหรือเรียงลำดับข้อมูล (Sorting) คือ การจัดเรียงข้อมูลให้เรียงลำดับตามเงื่อนไขที่กำหนดไว้ โดยอาจเรียงจากน้อยไปมาก หรือค่ามากไปน้อยก็ได้ การเรียงลำดับข้อมูลในระบบคอมพิวเตอร์ จะแบ่งเป็น 2 ลักษณะใหญ่ ๆ คือ

1. การจัดเรียงลำดับข้อมูลภายใน (Internal sorting)

2. การเรียงลำดับข้อมูลภายนอก (External sorting)

การจัดเรียงข้อมูล

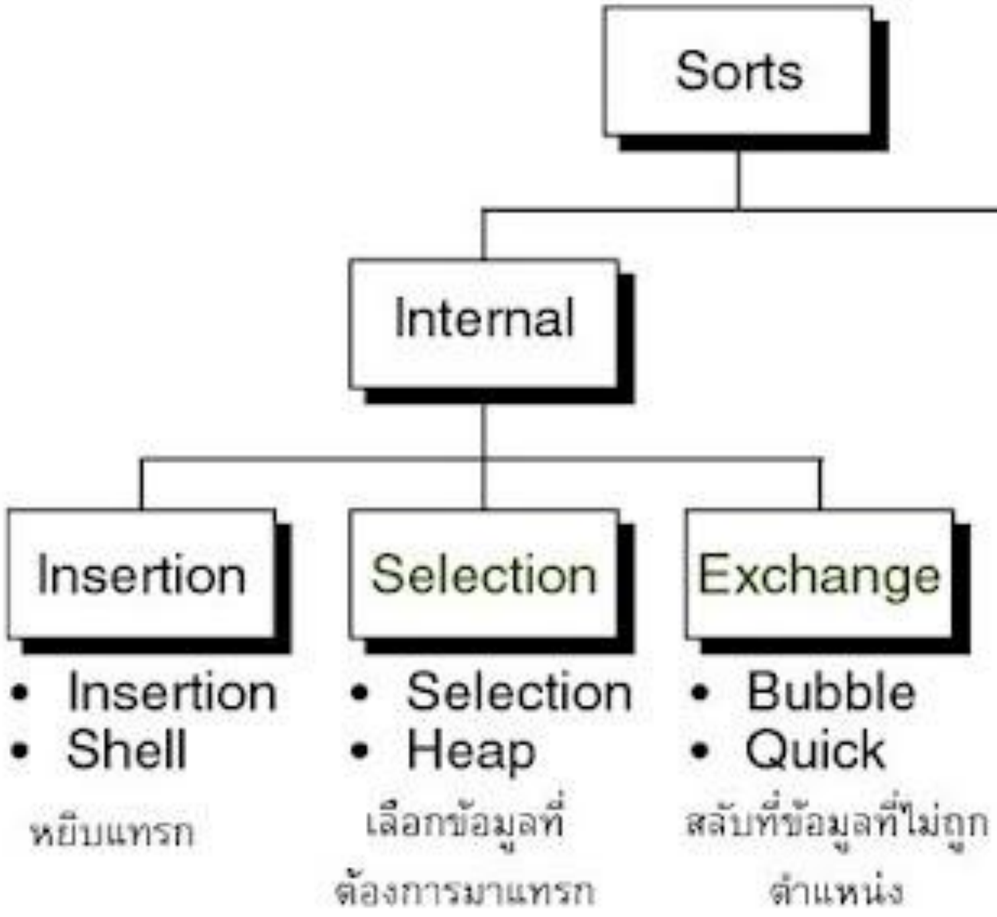
1. การจัดเรียงลำดับข้อมูลภายใน (Internal sorting)

- ใช้กับข้อมูลที่มีจำนวนไม่ใหญ่กว่าเนื้อที่ในหน่วยความจำ (main memory)
- ไม่ต้องใช้หน่วยความจำสำรอง เช่น ดิสก์, เทป เป็นต้น

2. การเรียงลำดับข้อมูลภายนอก (External sorting)

- ใช้กับข้อมูลที่มีจำนวนใหญ่เกินกว่าที่จะเก็บลงในหน่วยความจำได้หมดภายในครั้งเดียว
- จะใช้หน่วยความจำภายนอก เช่น ดิสก์, เทป สำหรับเก็บข้อมูลบางส่วนที่ได้รับการเรียงลำดับข้อมูลแล้ว แล้วจึงค่อยจัดการเรียงลำดับข้อมูลในส่วนต่อไป

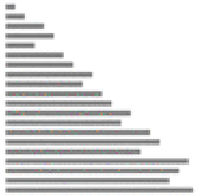
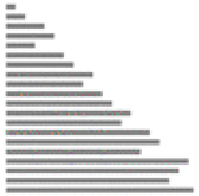
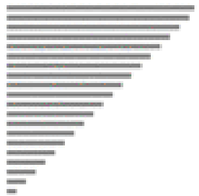
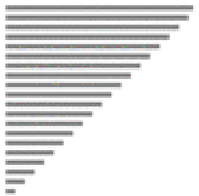
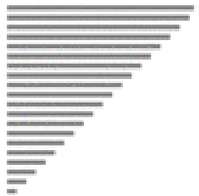
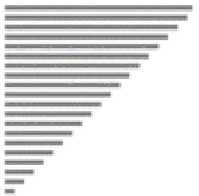
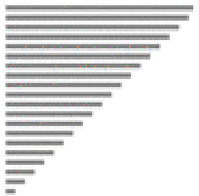
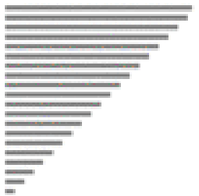
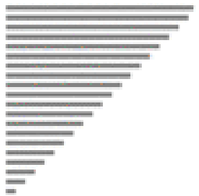
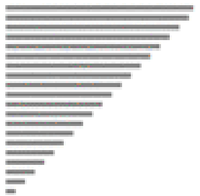
การจัดเรียงข้อมูล



Sort Order

1. Ascending Order เรียงจากน้อย –> มาก
2. Descending Order เรียงจากมาก –> น้อย

การจัดเรียงข้อมูล

	Insertion	Selection	Bubble	Shell	Merge	Heap	Quick	Quick3
Random								
Nearly Sorted								
Reversed								
								

การจัดเรียงข้อมูล

1. การจัดเรียงลำดับข้อมูลภายใน (Internal sorting)

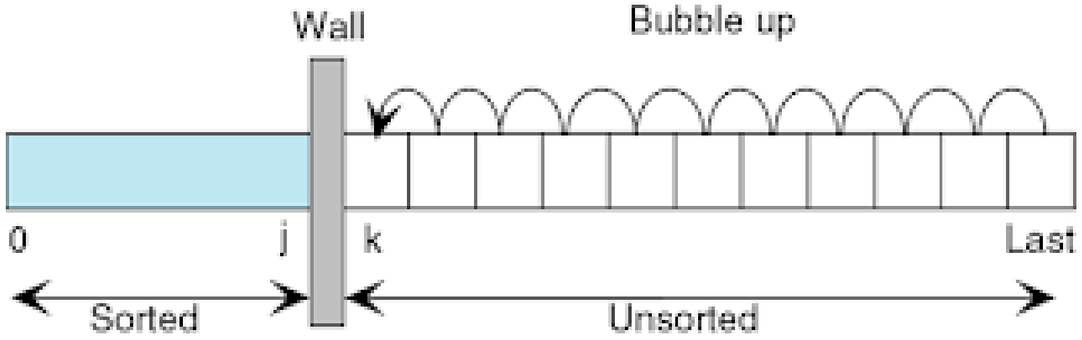
- ใช้กับข้อมูลที่มีจำนวนไม่ใหญ่กว่าเนื้อที่ในหน่วยความจำ (main memory)
- ไม่ต้องใช้หน่วยความจำสำรอง เช่น ดิสก์, เทป เป็นต้น

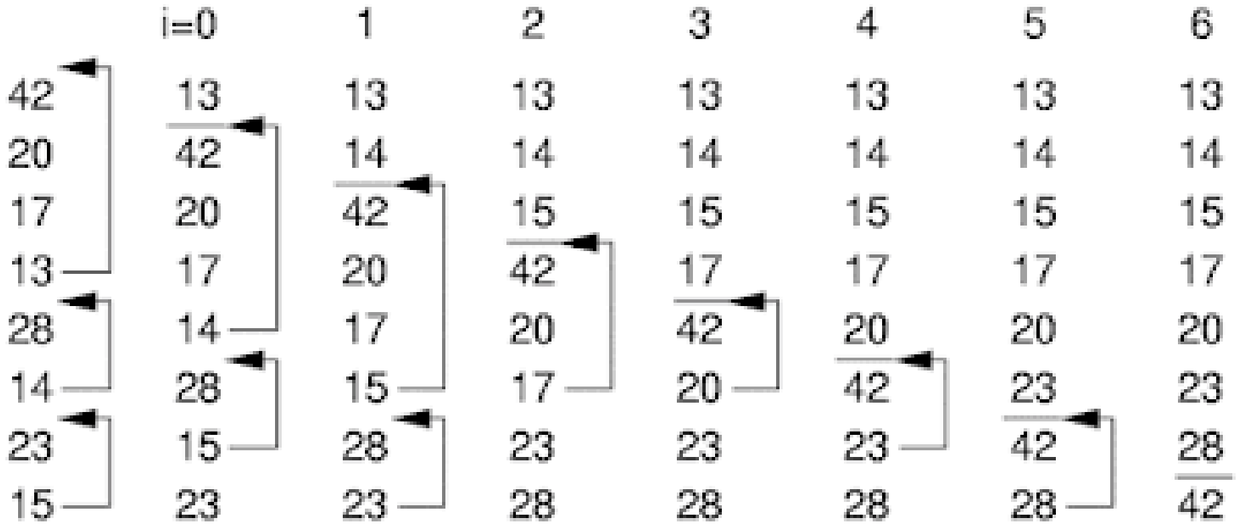
2. การเรียงลำดับข้อมูลภายนอก (External sorting)

- ใช้กับข้อมูลที่มีจำนวนใหญ่เกินกว่าที่จะเก็บลงในหน่วยความจำได้หมดภายในครั้งเดียว
- จะใช้หน่วยความจำภายนอก เช่น ดิสก์, เทป สำหรับเก็บข้อมูลบางส่วนที่ได้รับการเรียงลำดับข้อมูลแล้ว แล้วจึงค่อยจัดการเรียงลำดับข้อมูลในส่วนต่อไป

1. Bubble Sort

การจัดเรียงแบบบับเบิล เป็นการจัดเรียงโดยการเปรียบเทียบค่า 2 ค่าที่ติดกัน ทำต่อเนื่องกันไป





```
void bubbleSort(int arr[], int n){
    int i, j;
    for (i = 0; i < n - 1; i++)
        // Last i elements are already in place
        for (j = 0; j < n - i - 1; j++)
            if (arr[j] > arr[j + 1])
                swap(&arr[j], &arr[j + 1]);
}
```

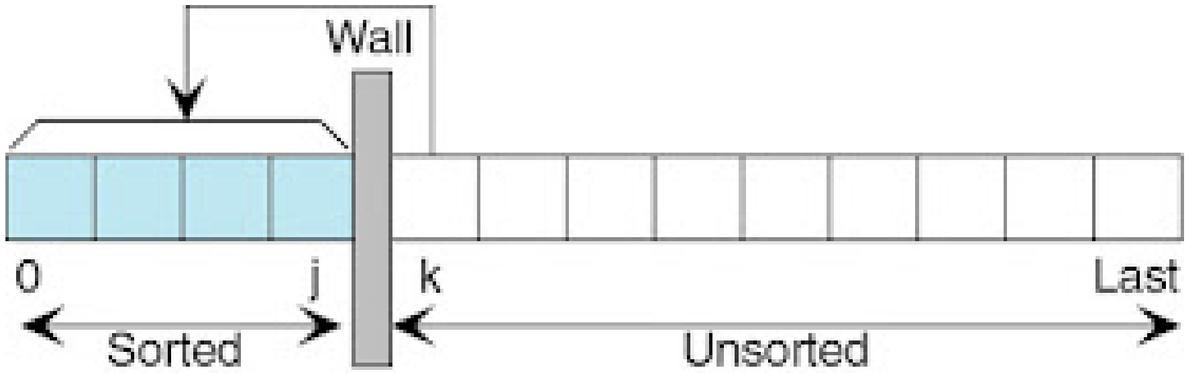
2. Insertion Sort

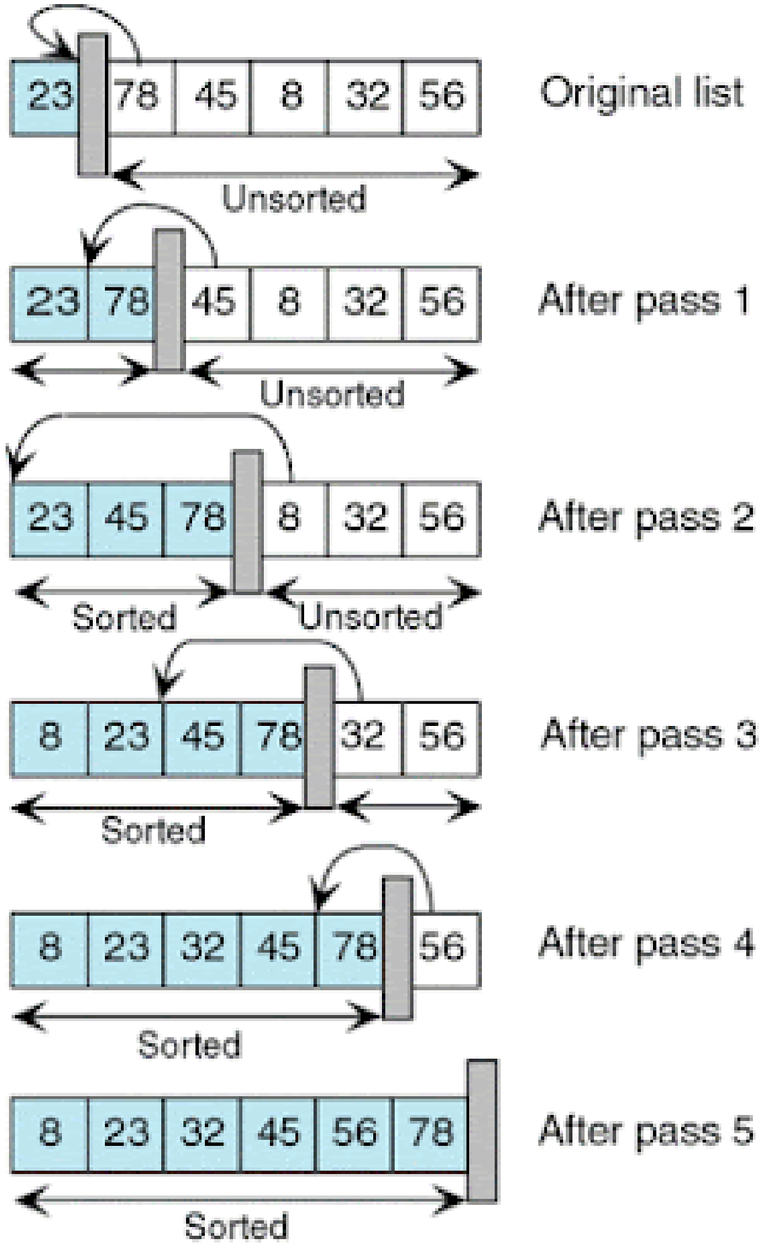
การจัดเรียงแบบแทรก คือ การเรียงข้อมูลโดยนำข้อมูลที่จะทำการจัดเรียงนั้นๆ ไปจัดเรียงทีละตัว โดยการแทรกตัวที่จะเรียงไว้ในตำแหน่งที่เหมาะสมของข้อมูลที่มีการจัดเรียง เรียบร้อยแล้ว ณ ตำแหน่งที่ถูกต้อง

ขั้นตอนการทำงาน

. เปรียบเทียบค่ากับตำแหน่งถัดไปแทน

. สลับตำแหน่งให้อยู่ในตำแหน่งที่เหมาะสม



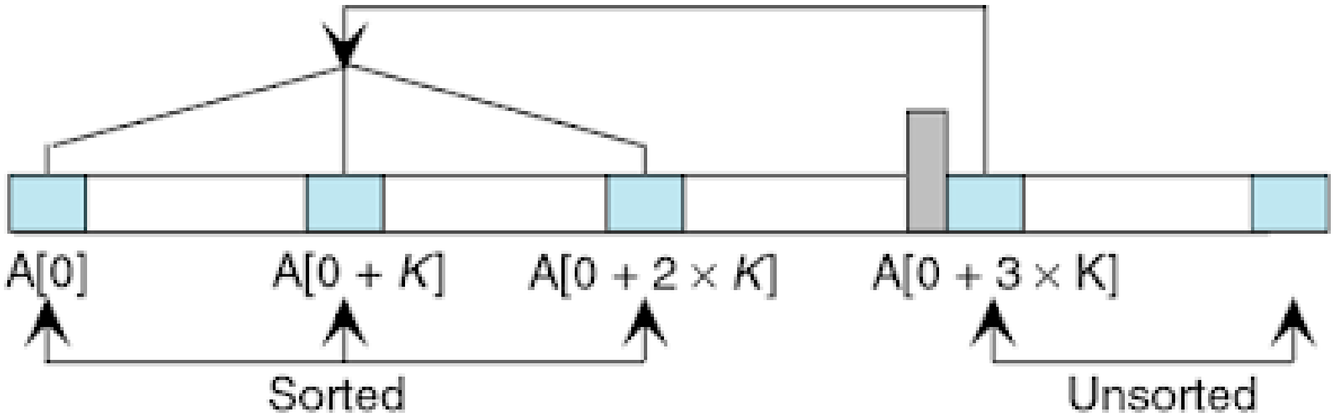


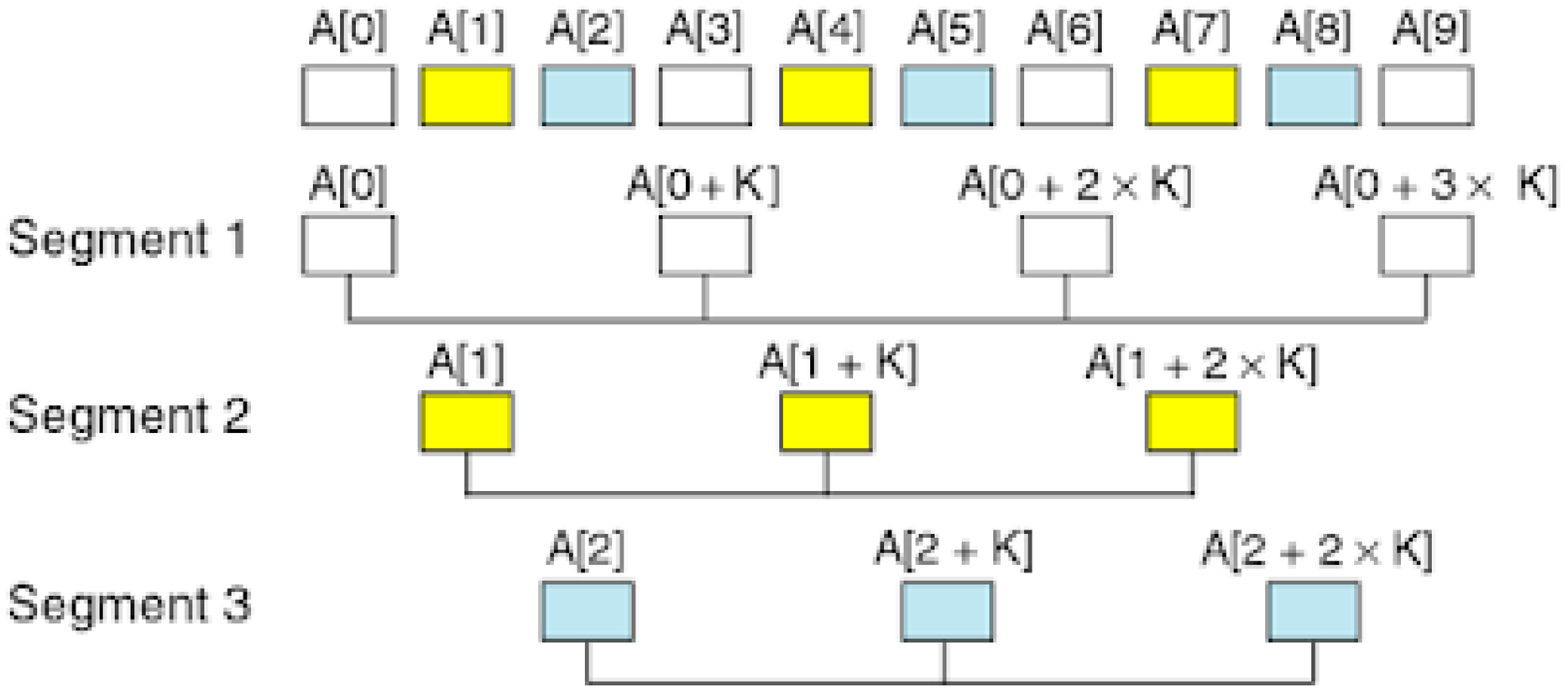
```
void insertionSort(int arr[], int n)
{
    int i, key, j;
    for (i = 1; i < n; i++) {
        key = arr[i];
        j = i - 1;

        /* Move elements of arr[0..i-1], that are
          greater than key, to one position ahead
          of their current position */
        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j = j - 1;
        }
        arr[j + 1] = key;
    }
}
```

3. Shell Sort

การจัดเรียงแบบเชลล์ เป็นการจัดเรียงที่อาศัยเทคนิคการแบ่งข้อมูลออกเป็นกลุ่มย่อยหลายๆ กลุ่ม แล้วจัดเรียงข้อมูลในกลุ่มย่อยๆนั้น หลังจากนั้นก็ให้รวมกลุ่มย่อยๆ ให้ใหญ่ขึ้นเรื่อยๆ ขั้นสุดท้ายให้จัดเรียงข้อมูลทั้งหมดนั้นอีกครั้ง



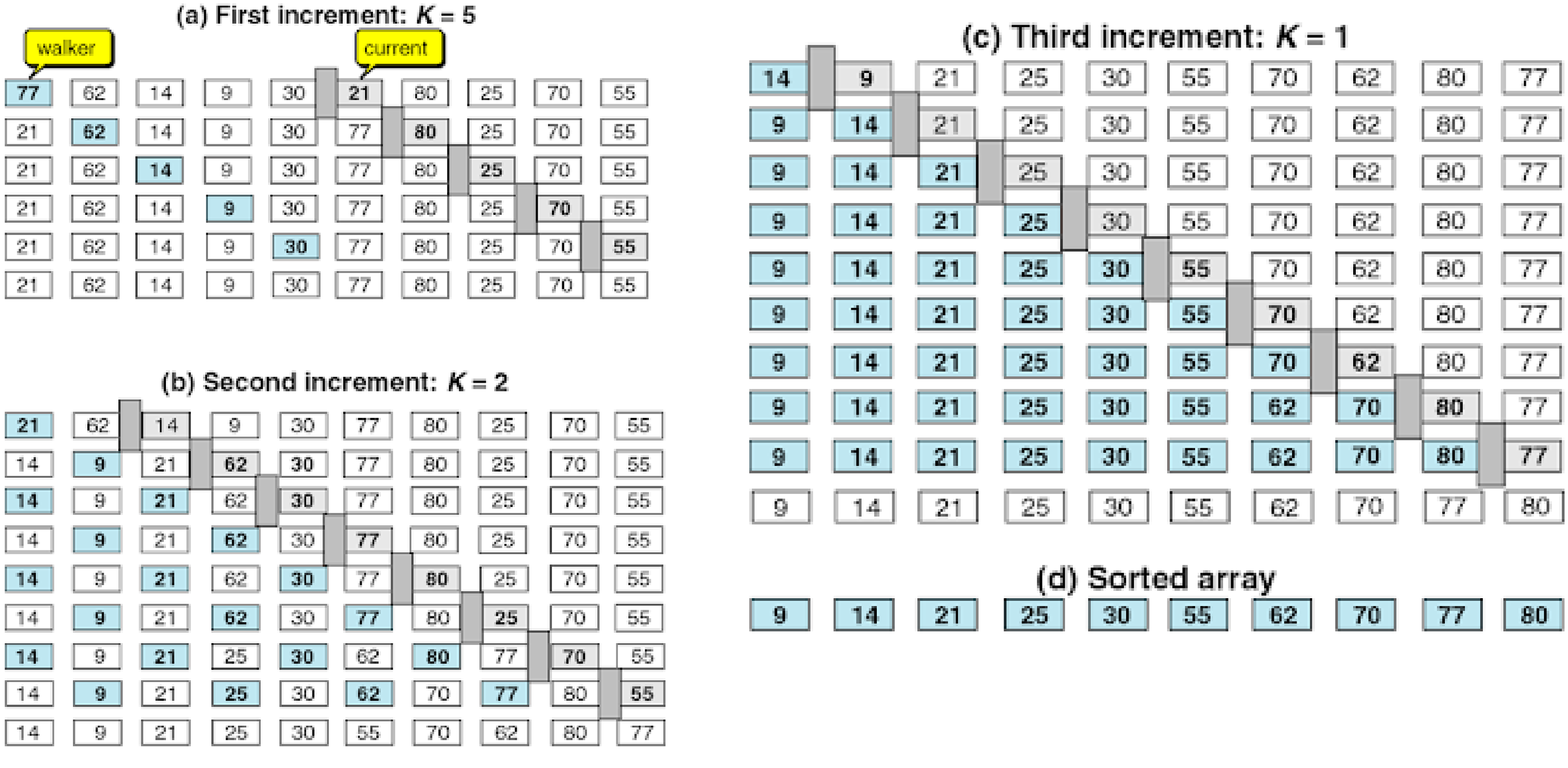

A[0]
A[1]
A[2]
A[3]
A[4]
A[5]
A[6]
A[7]
A[8]
A[9]
Segment 1
A[0]
A[0 + K]
A[0 + 2 × K]
A[0 + 3 × K]
Segment 2
A[1]
A[1 + K]
A[1 + 2 × K]
Segment 3
A[2]
A[2 + K]
A[2 + 2 × K]

ขั้นตอนการทำงาน

. โดยทั่วไปการเลือกค่า K ตัวแรกมักจะเลือกใช้ค่าเท่ากับครึ่งหนึ่งของข้อมูล เช่น ข้อมูลมี 10 ตัว K = n/2 = 10/2 = 5

. เรียงข้อมูลทุกตัวให้เสร็จสิ้น แล้วกำหนดค่า K ใหม่ (โดยทั่วไปจะเป็นครึ่งหนึ่งของค่า K ตัวแรก เช่น K1 = 5; K2 = 5/2 = 2)

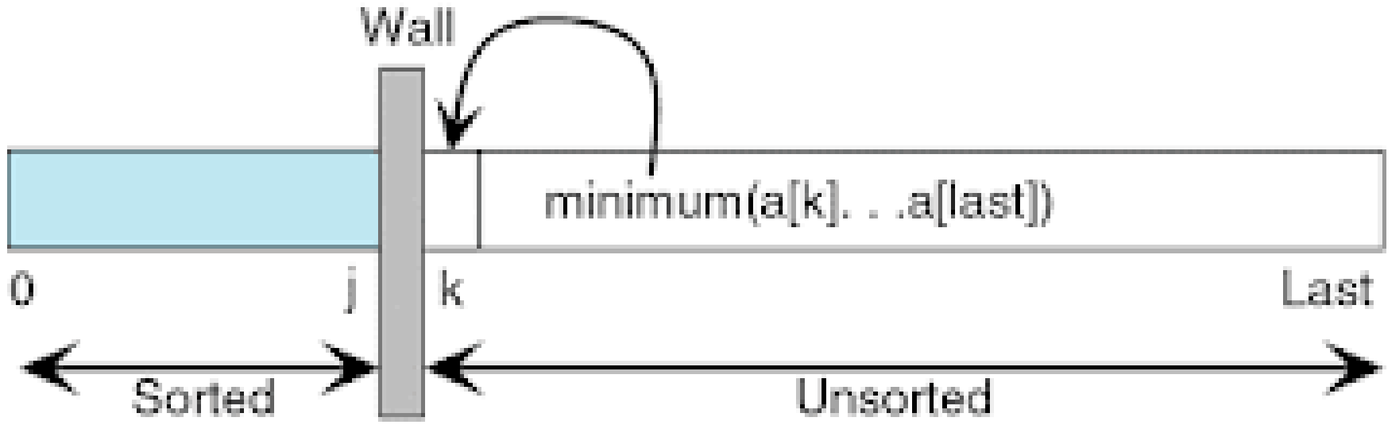
. ถ้า K > 1 ให้ทำซ้ำ จนกระทั่งเหลือข้อมูลกลุ่มเดียว ถ้า K = 1 ให้เรียงลำดับตามปกติ


(a) First increment: K = 5
walker
current
77 62 14 9 30 21 80 25 70 55
21 62 14 9 30 77 80 25 70 55
21 62 14 9 30 77 80 25 70 55
21 62 14 9 30 77 80 25 70 55
21 62 14 9 30 77 80 25 70 55
21 62 14 9 30 77 80 25 70 55
(b) Second increment: K = 2
21 62 14 9 30 77 80 25 70 55
14 9 21 62 30 77 80 25 70 55
14 9 21 62 30 77 80 25 70 55
14 9 21 62 30 77 80 25 70 55
14 9 21 62 30 77 80 25 70 55
14 9 21 62 30 77 80 25 70 55
14 9 21 25 30 62 80 77 70 55
14 9 21 25 30 62 70 77 80 55
14 9 21 25 30 55 70 62 80 77
(c) Third increment: K = 1
14 9 21 25 30 55 70 62 80 77
9 14 21 25 30 55 70 62 80 77
9 14 21 25 30 55 70 62 80 77
9 14 21 25 30 55 70 62 80 77
9 14 21 25 30 55 70 62 80 77
9 14 21 25 30 55 70 62 80 77
9 14 21 25 30 55 70 62 80 77
9 14 21 25 30 55 62 70 80 77
9 14 21 25 30 55 62 70 80 77
9 14 21 25 30 55 62 70 77 80
(d) Sorted array
9 14 21 25 30 55 62 70 77 80

```
void shellSort(int array[], int n) {
  // Rearrange elements at each n/2, n/4, n/8, ... intervals
  for (int interval = n / 2; interval > 0; interval /= 2) {
    for (int i = interval; i < n; i += 1) {
      int temp = array[i];
      int j;
      for (j = i; j >= interval && array[j - interval] > temp; j -= interval) {
        array[j] = array[j - interval];
      }
      array[j] = temp;
    }
  }
}
```

4. Selection Sort

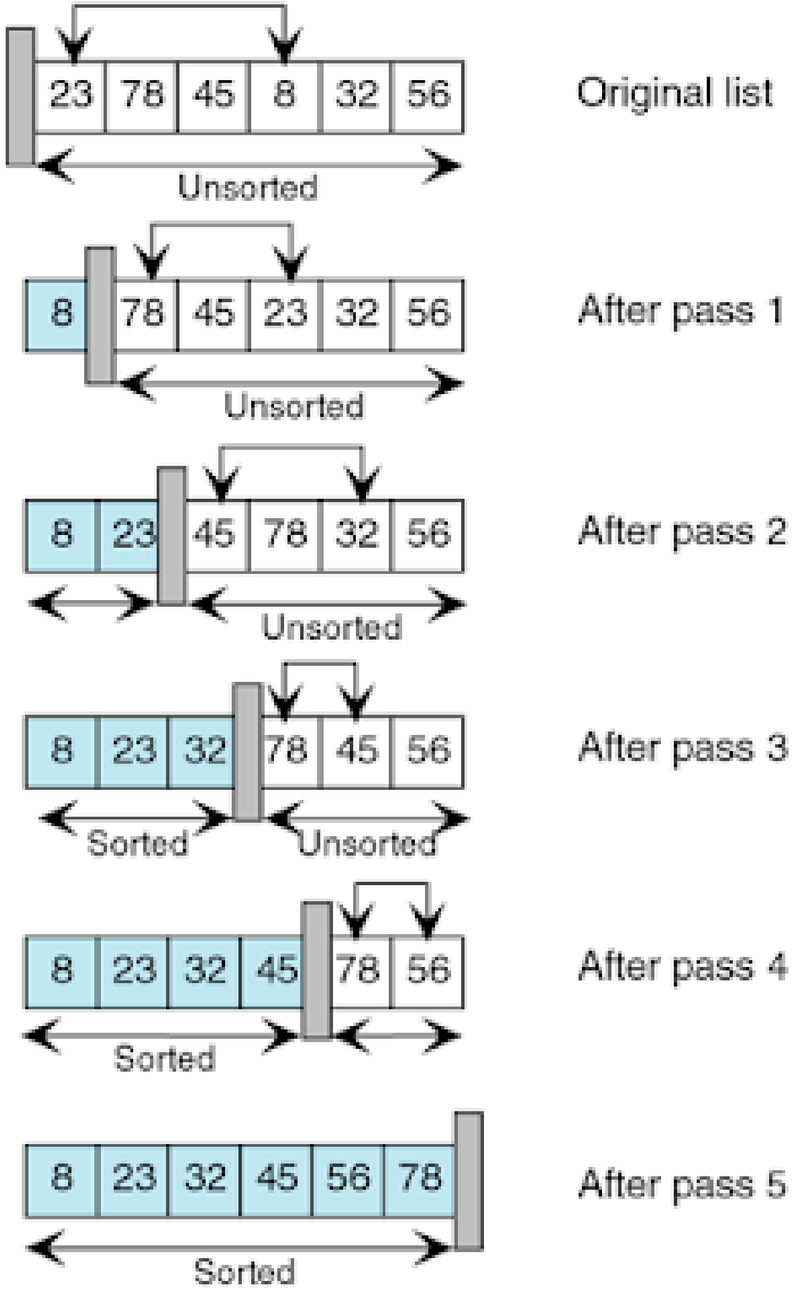
การจัดเรียงแบบเลือก เป็นวิธีการเรียงข้อมูลโดยจะเริ่มต้นค้นหาข้อมูลตัวที่น้อยที่สุดจากข้อมูลที่มีอยู่ทั้งหมด แล้วสลับที่ข้อมูลกับตัวแรก แล้วกลับไปหาข้อมูลตัวที่น้อยที่สุดในกองต่อไปสลับที่กับข้อมูลจนกว่าจะหมด กอง



ขั้นตอนการทำงาน

. ค้นหาตัวเลขที่มีค่าน้อย/มากที่สุดตั้งแต่ตัวแรกไปจนถึงตัวสุดท้าย

. สลับตำแหน่งตัวเลขที่มีค่าน้อย/มากที่สุด



```
void selectionSort(int arr[], int n)
{
    int i, j, min_idx;

    // One by one move boundary of unsorted subarray
    for (i = 0; i < n-1; i++)
    {
    // Find the minimum element in unsorted array
        min_idx = i;
        for (j = i+1; j < n; j++)
        if (arr[j] < arr[min_idx])
            min_idx = j;

    // Swap the found minimum element with the first element
        if(min_idx != i)
            swap(&arr[min_idx], &arr[i]);
    }
}
```

5. Heap Sort

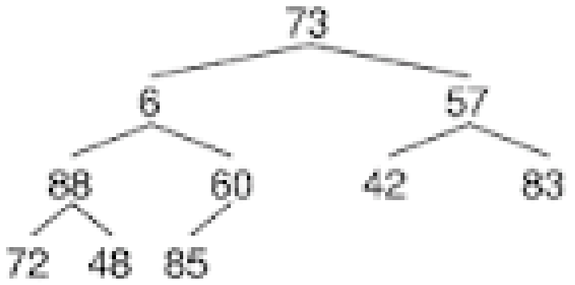
ฮีปเป็นโครงสร้างข้อมูลที่มีลักษระการจัดเก็บข้อมูลแบบไบนารีทรี คือ แต่ละโหนดจะมีโหนดลูกได้ไม่เกิน 2 โหนด และการจัดเก็บข้อมูลจะต้องจัดเก็บให้เต็มทีละชั้นเรียงจากโหนดด้านซ้ายมือไป ด้านขวามือเสมอ

ฮีปแบ่งเป็น 2 ประเภท คือ

- Min Heap: ที่โหนดใดๆ ก็ตามภายในฮีป ข้อมูลที่ซับทรีจะต้องมีค่ามากกว่าหรือเท่ากับข้อมูลที่ตัวมันเสมอ(รูทโหนดมีค่าต่ำที่สุด)
- Max Heap : ที่โหนดใดๆ ก็ตามภายในฮีป ข้อมูลที่ซับทรีจะต้องมีค่าน้อยกว่าหรือเท่ากับข้อมูลที่ตัวมันเสมอ (รูทโหนดมีค่าสูงที่สุด)

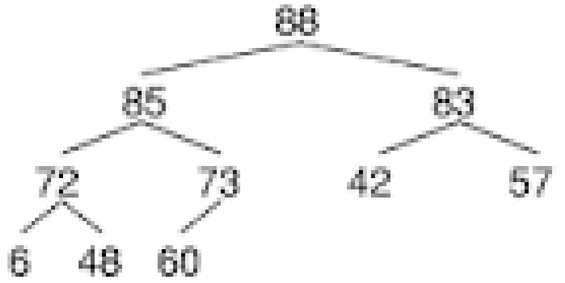
Original Numbers

73	6	57	88	60	42	83	72	48	85



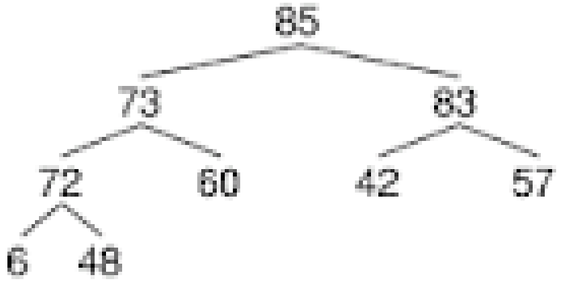
Build Heap

88	85	83	72	73	42	57	6	48	60



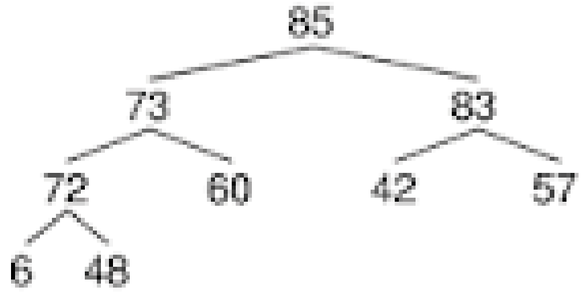
Remove 88

85	73	83	72	60	42	57	6	48	88



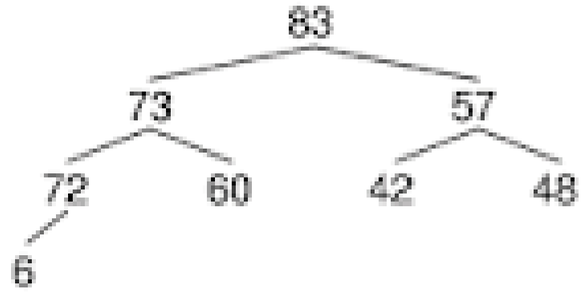
Remove 88

85	73	83	72	60	42	57	6	48	88



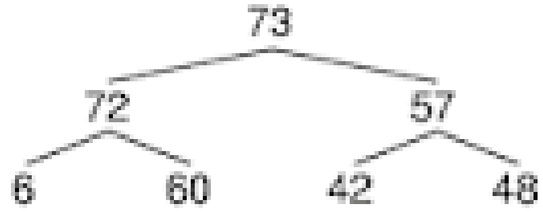
Remove 85

83	73	57	72	60	42	48	6	85	88



Remove 83

73	72	57	6	60	42	48	83	85	88



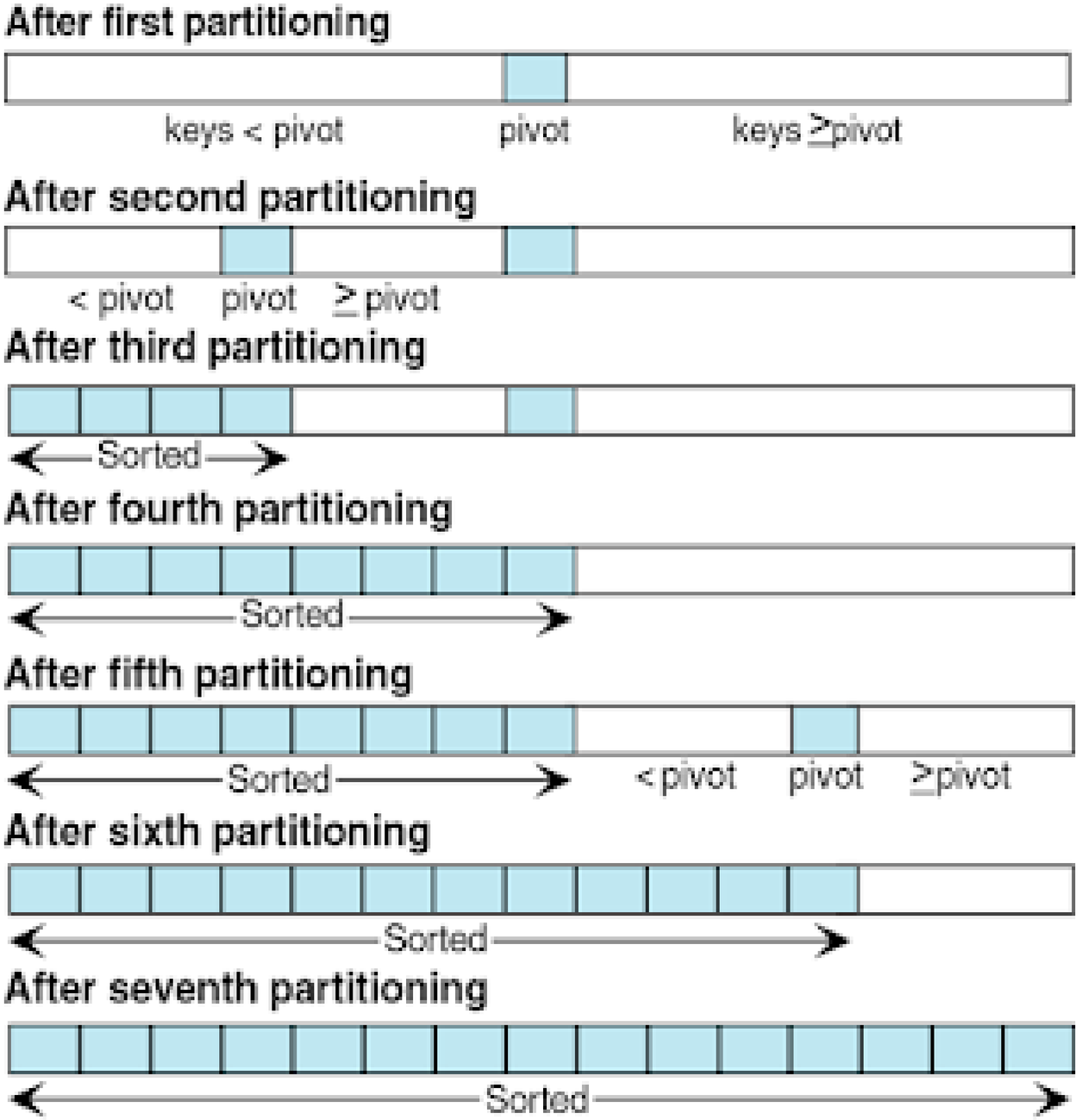
```
void heapify(int arr[], int N, int i)
{
      // Find largest among root, left child and right child // Initialize largest as root
      int largest = i;
      // left = 2*i + 1
      int left = 2 * i + 1;
      // right = 2*i + 2
      int right = 2 * i + 2;
      // If left child is larger than root
      if (left < N && arr[left] > arr[largest])
            largest = left;
      // If right child is larger than largest so far
      if (right < N && arr[right] > arr[largest])
            largest = right;
      // Swap and continue heapifying if root is not largest
      // If largest is not root
      if (largest != i) {
            swap(&arr[i], &arr[largest]);
            // Recursively heapify the affected
            // sub-tree
            heapify(arr, N, largest);
      }
}
```

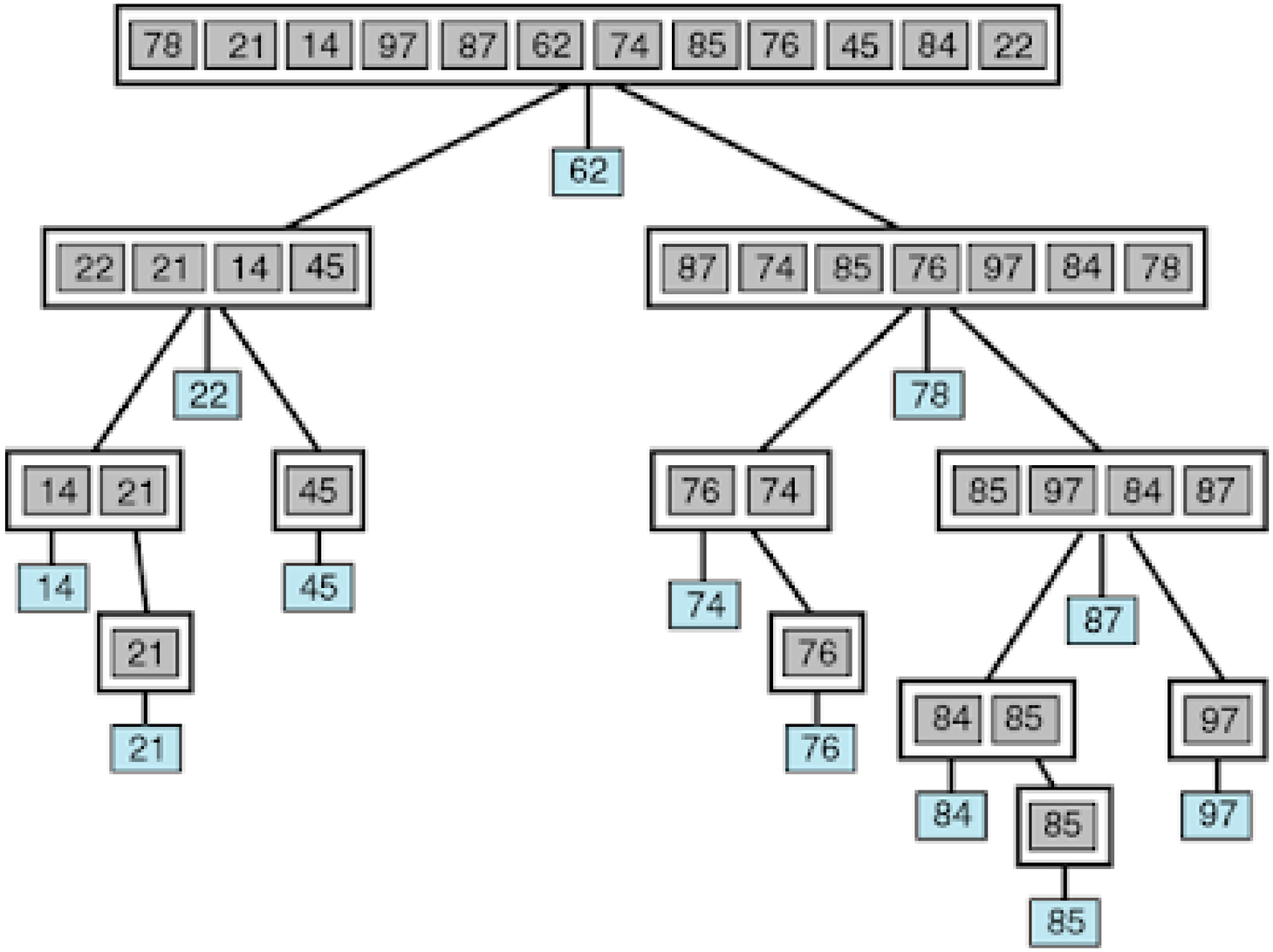
6. Quick Sort

การจัดเรียงแบบเร็ว ใช้หลักการ divide-and-conquer อาศัยการจัดแบ่งข้อมูลทั้งหมดออกเป็น 2 กลุ่ม โดยกลุ่มแรกจะเป็นกลุ่มของข้อมูลที่มีค่าน้อยกว่าค่ากลางที่กำหนด และส่วนที่สองเป็นกลุ่มของข้อมูลที่มีค่ามากกว่าค่ากลางที่กำหนด หลังจากนั้นแบ่งข้อมูลแต่ละส่วนออกเป็น 2 ส่วนเช่นเดิม แบ่งไปเรื่อยๆจนไม่สามารถแบ่งได้ก็จะได้ข้อมูลที่เรียงกัน

ขั้นตอนการทำงาน

1. มีการเลือกข้อมูลตัวหนึ่งเรียกว่า Pivot ที่ใช้เป็นตัวแบ่งแยกชุดข้อมูลที่เรามีออกเป็นส่วน คือ ข้อมูลที่มีค่าน้อยกว่า Pivot และข้อมูลที่มีค่ามากกว่า Pivot
2. แบ่งข้อมูลไปเรื่อยๆ
3. เรียงข้อมูลแต่ละส่วนย่อยๆ

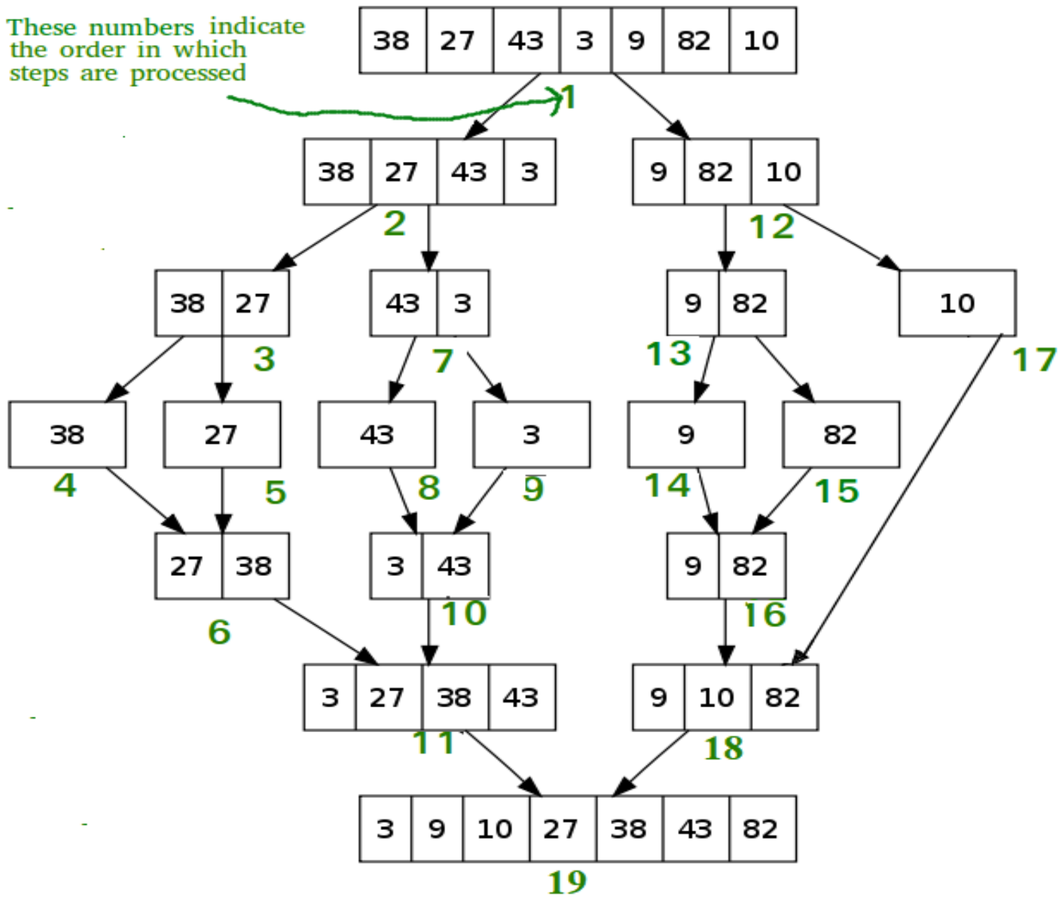

After first partitioning
keys < pivot
pivot
keys ≥ pivot
After second partitioning
< pivot
pivot
≥ pivot
After third partitioning
Sorted
After fourth partitioning
Sorted
After fifth partitioning
Sorted
< pivot
pivot
≥ pivot
After sixth partitioning
Sorted
After seventh partitioning
Sorted


78 21 14 97 87 62 74 85 76 45 84 22
62
22 21 14 45
87 74 85 76 97 84 78
22
78
14 21
45
76 74
85 97 84 87
14
45
74
76
87
21
84 85
97
21
76
84
85
97
85

7. Merge Sort

การเรียงแบบผสาน (Merge Sort) ใช้หลักการ divide-and-conquer เหมือนกับ Quick Sort มีลักษณะของการแบ่งข้อมูลออกเป็นส่วนๆ แต่กระบวนการเรียงข้อมูลนั้นจะแตกต่างไปจาก Quick sort Quick sort กระทำการสลับข้อมูลไปพร้อมกับการแบ่งข้อมูลออกเป็นส่วนๆ แต่ merge sort นี้ กระทำการแบ่งข้อมูลออกเป็นส่วนๆก่อน แล้วค่อยเรียงข้อมูลในส่วนย่อย จากนั้นนำเอาข้อมูลส่วนย่อยที่เรียงไว้แล้ว มารวมกันและเรียงไปในเวลาเดียวกัน อัลกอริทึมจะเรียงพร้อมกับผสานข้อมูล เข้าด้วยกันจนกระทั่งข้อมูลทุกตัวรวมกันกลายเป็นข้อมูลเดียวอีกครั้ง

36	20	17	13	28	14	23	15	แบ่งข้อมูลออกเป็นข้อมูลย่อยๆ
20	36	13	17	14	28	15	23	จัดเรียงข้อมูลย่อย
13	17	20	36	14	15	23	28	นำข้อมูลย่อยๆ นั้นมารวมกันให้เป็นข้อมูลเดียว
13	14	15	17	20	23	28	36	



```
void merge(int arr[], int l, int m, int r)
{
	int i, j, k;
	int n1 = m - l + 1;
	int n2 = r - m;
	/* create temp arrays */
	int L[n1], R[n2];
	/* Copy data to temp arrays L[] and R[] */
	for (i = 0; i < n1; i++)
		L[i] = arr[l + i];
	for (j = 0; j < n2; j++)
		R[j] = arr[m + 1 + j];
	/* Merge the temp arrays back into arr[l..r]*/
	i = 0; // Initial index of first subarray
	j = 0; // Initial index of second subarray
	k = l; // Initial index of merged subarray
	while (i < n1 && j < n2) {
		if (L[i] <= R[j]) {
			arr[k] = L[i];
			i++;
		}
		else {
			arr[k] = R[j];
			j++;
		}
		k++;
	}
```

```
    /* Copy the remaining elements of L[], if there
    are any */
    while (i < n1) {
        arr[k] = L[i];
        i++;
        k++;
    }

    /* Copy the remaining elements of R[], if there
    are any */
    while (j < n2) {
        arr[k] = R[j];
        j++;
        k++;
    }
}
```

Sorting Algorithms	Time Complexity			Space Complexity
	Best Case	Average Case	Worst Case	Worst Case
Bubble Sort	O(n)	O(n^2)	O(n^2)	O(1)
Selection Sort	O(n^2)	O(n^2)	O(n^2)	O(1)
Insertion Sort	O(n)	O(n^2)	O(n^2)	O(1)
Merge Sort	O(nlogn)	O(nlogn)	O(nlogn)	O(n)
Quick Sort	O(nlogn)	O(nlogn)	O(n^2)	O(n)
Heap Sort	O(nlogn)	O(nlogn)	O(nlogn)	O(1)
Counting Sort	O(n + k)	O(n + k)	O(n + k)	O(k)
Radix Sort	O(nk)	O(nk)	O(nk)	O(n + k)
Bucket Sort	O(n + k)	O(n + k)	O(n^2)	O(n)

ฟังก์ชัน sort ใน c

qsort()

การใช้งาน

void qsort(void *base, size_t nitems, size_t size, int (*compar)(const void *, const void*))

Parameters

base – This is the pointer to the first element of the array to be sorted.

nitems – This is the number of elements in the array pointed by base.

size – This is the size in bytes of each element in the array.

compar – This is the function that compares two elements.

```
#include <stdio.h>
#include <stdlib.h>
int comp (const void * elem1, const void * elem2)
{
    int f = *((int*)elem1);
    int s = *((int*)elem2);
    if (f > s) return  1;
    if (f < s) return -1;
    return 0;
}
int main(int argc, char* argv[])
{
    int x[] = {4,5,2,3,1,0,9,8,6,7};

    qsort (x, sizeof(x)/sizeof(*x), sizeof(*x), comp);

    for (int i = 0 ; i < 10 ; i++)
        printf ("%d ", x[i]);
```

Sort in c++

```
#include <bits/stdc++.h>

 using namespace std;

bool compare(int a,int b){
    return a>b;
}
main(){
    int a[]={7,2,1,9,3,7,8,5,1,2};
    sort(a,a+10,compare);
    int i;
    for(i=0;i<=10;i++){
        printf("%d",a[i]);
    }
}
```