โครงการโอลิมปิกวิชาการ สาขาคอมพิวเตอร์ ค่าย 2

การจัดเรียงลำดับ (Sorting)

อ.ดร.ปัญญนัท อ้นพงษ์

ภาควิชาคอมพิวเตอร์ คณะวิทยาศาสตร์ มหาวิทยาลัยศิลปากร

aonpong_p@su.ac.th

Outline

- บทนำเกี่ยวกับการจัดเรียงลำดับ
- Sorting Algorithm
- การเปรียบเทียบประสิทธิภาพของ Sorting Algorithm
- การใช้งานและการเลือก Sorting Algorithm
- Standard Template Library (STL)
- การประยุกต์ใช้ Sorting Algorithm

บทนำเกี่ยวกับการจัดเรียงลำดับ

- การจัดเรียงข้อมูล (Sorting) เป็นกระบวนการพื้นฐานที่สำคัญในการประมวลผลข้อมูล
- ช่วยให้การทำงานกับข้อมูลมีประสิทธิภาพมากขึ้น
- ใช้เป็นส่วนหนึ่งของการค้นหา ประมวลผล จัดเก็บ และวิเคราะห์ข้อมูล
- เป็นส่วนสำคัญในการเรียนรู้อัลกอริทึมที่ซับซ้อนในส่วนต่อไป

บทนำเกี่ยวกับการจัดเรียงลำดับ

ตัวอย่าง จงระบุว่าข้อมูลในอาเรย์ต่อไปนี้มีเลข 38 อยู่กี่ตัว

Array[] = {3, 28, 51, 77, 12, 35, 63, 42, 19, 88, 10, 56, 73, 41, 7, 92, 14, 67, 30, 49, 81, 9, 60, 25, 94, 3, 70, 52, 16, 87, 38, 42, 36, 23,79, 38, 24, 1, 95, 72, 35, 98, 74, 32, 38, 42, 75, 76, 39}

บทนำเกี่ยวกับการจัดเรียงลำดับ

ตัวอย่าง จงระบุว่าข้อมูลในอาเรย์ต่อไปนี้มีเลข 38 อยู่กี่ตัว

Array[] = {3, 28, 51, 77, 12, 35, 63, 42, 19, 88, 10, 56, 73, 41, 7, 92, 14, 67, 30, 49, 81, 9, 60, 25, 94, 3, 70, 52, 16, 87, 38, 42, 36, 23,79, 38, 24, 1, 95, 72, 35, 98, 74, 32, 38, 42, 75, 76, 39}

หลังจากทำการจัดเรียงข้อมูลแล้ว จะได้ว่า

1, 3, 3, 7, 9, 10, 12, 14, 16, 19, 23, 24, 25, 28, 30, 32, 35, 35, 36, **38, 38, 38**, 39, 41, 42, 42, 42, 49, 51, 52, 56, 60, 63, 67, 70, 72, 73, 74, 75, 76, 77, 79, 81, 87, 88, 92, 94, 95, 98

บทนำเกี่ยวกับการจัดเรียงลำดับ

การจัดเรียงข้อมูล แบ่งออกเป็น 2 ประเภท

- Internal Sorting คือการจัดเรียงข้อมูลที่ทำได้ในหน่วยความจำ (RAM)
- External Sorting คือการจัดเรียงข้อมูลที่ต้องทำใน Disk (HDD/SSD)

เนื่องจากข้อมูลที่เราจะจัดการไม่ได้ใหญ่ขนาดนั้น และการทำงานของเกรดเดอร์ก็ไม่ยอมให้มีการใช้งาน HDD/SSD ของเซิร์ฟเวอร์อยู่แล้ว ดังนั้นในที่นี้ เราจะไม่กล่าวถึงกระบวนการการทำงานของ External Sorting

Outline

- บทนำเกี่ยวกับการจัดเรียงลำดับ
- Sorting Algorithm
- การเปรียบเทียบประสิทธิภาพของ Sorting Algorithm
- การใช้งานและการเลือก Sorting Algorithm
- Standard Template Library (STL)
- การประยุกต์ใช้ Sorting Algorithm

Sorting Algorithm

วิธีการจัดเรียงที่จะกล่าวถึง จะเข้าถึงข้อมูลในอาเรย์ทุกตำแหน่งที่จะทำการจัดเรียง ซึ่งประกอบด้วยสมาชิก n ตัว

วิธีการจัดเรียงมีหลายวิธี โดยเราจะกล่าวถึงเป็นกลุ่ม ๆ ได้แก่

- Selection Sort, Bubble Sort, Insertion Sort
- Merge Sort, Quick Sort
- Heap Sort

Sorting Algorithm: Selection Sort

Selection sort เป็นอัลกอริทึมการเรียงลำดับที่ง่ายและมีประสิทธิภาพ โดยที่มันทำงานโดยการเลือกสมาชิกที่น้อยที่สุด (หรือมากที่สุด) จากส่วนที่ยังไม่ได้เรียงลำดับของรายการแล้วย้ายมันไปยังส่วนที่เรียงลำดับแล้วของรายการนั้น ๆ อย่างต่อเนื่อง

หากต้องการเรียงข้อมูลต่อไปนี้จากน้อยไปมาก

7	6	4	3	9	1	2	5	8

Sorting Algorithm: Selection Sort

ขั้นตอนการทำงาน

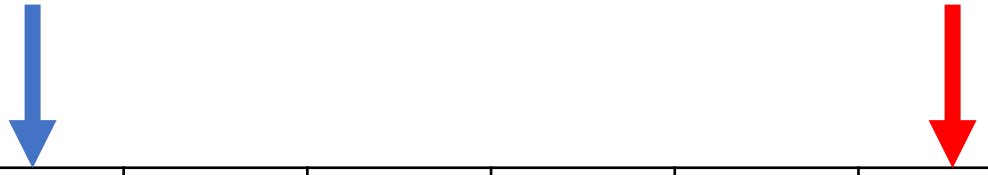
1. กำหนดตำแหน่งแรกของอาร์เรย์ว่าเป็นส่วนที่ยังไม่ได้เรียงลำดับ นั่นคือตำแหน่งที่ 0

index	0	1	2	3	4	5	6	7	8
value	7	6	4	3	9	1	2	5	8

Sorting Algorithm: Selection Sort

ขั้นตอนการทำงาน

1. กำหนดตำแหน่งแรกของอาร์เรย์ว่าเป็นส่วนที่ยังไม่ได้เรียงลำดับ นั่นคือตำแหน่งที่ 0
2. ค้นหาสมาชิกที่น้อยที่สุดในส่วนที่ยังไม่ได้เรียงลำดับ โดยตรวจสอบทีละตัว

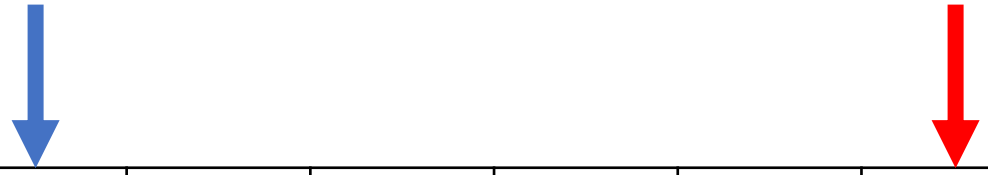


index	0	1	2	3	4	5	6	7	8
value	7	6	4	3	9	1	2	5	8

Sorting Algorithm: Selection Sort

ขั้นตอนการทำงาน

1. กำหนดตำแหน่งแรกของอาร์เรย์ว่าเป็นส่วนที่ยังไม่ได้เรียงลำดับ นั่นคือตำแหน่งที่ 0
2. ค้นหาสมาชิกที่น้อยที่สุดในส่วนที่ยังไม่ได้เรียงลำดับ โดยตรวจสอบทีละตัว
3. สลับตำแหน่งค่าในอาเรย์ตัวแรกในส่วนที่ยังไม่ได้เรียงลำดับกับสมาชิกที่น้อยที่สุด

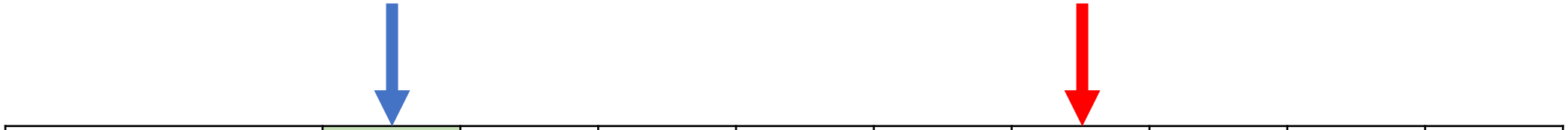


index	0	1	2	3	4	5	6	7	8
value	1	6	4	3	9	7	2	5	8

Sorting Algorithm: Selection Sort

ขั้นตอนการทำงาน

4. ปรับอาเรย์ตัวแรกในส่วนที่ยังไม่ได้เรียงลำดับ (ลูกศรสีฟ้า) ให้กลายเป็นส่วนที่เรียงลำดับแล้ว



index	0	1	2	3	4	5	6	7	8
value	1	6	4	3	9	7	2	5	8

Sorting Algorithm: Selection Sort

ขั้นตอนการทำงาน

4. ปรับอาเรย์ตัวแรกในส่วนที่ยังไม่ได้เรียงลำดับ (ลูกศรสีฟ้า) ให้กลายเป็นส่วนที่เรียงลำดับแล้ว
5. เลื่อนลูกศรสีฟ้าไปที่ตำแหน่งแรกของส่วนที่ยังไม่ได้เรียงลำดับ

index	0	1	2	3	4	5	6	7	8
value	1	6	4	3	9	7	2	5	8

Sorting Algorithm: Selection Sort

ขั้นตอนการทำงาน

6. เฉพาะส่วนที่ยังไม่ได้เรียงลำดับ ให้ทำข้อ 2-5 ซ้ำ จนกว่าทุกข้อมูลจะเปลี่ยนเป็นข้อมูลที่เรียงลำดับแล้ว : หาค่าน้อยสุด

index	0	1	2	3	4	5	6	7	8
value	1	6	4	3	9	7	2	5	8

Sorting Algorithm: Selection Sort

ขั้นตอนการทำงาน

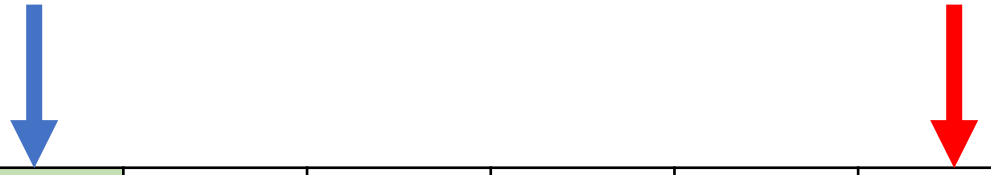
6. เฉพาะส่วนที่ยังไม่ได้เรียงลำดับ ให้ทำข้อ 2-5 ซ้ำ จนกว่าทุกข้อมูลจะเปลี่ยนเป็นข้อมูลที่เรียงลำดับแล้ว : สลับตำแหน่ง

index	0	1	2	3	4	5	6	7	8
value	1	2	4	3	9	7	6	5	8

Sorting Algorithm: Selection Sort

ขั้นตอนการทำงาน

6. เฉพาะส่วนที่ยังไม่ได้เรียงลำดับ ให้ทำข้อ 2-5 ซ้ำ จนกว่าทุกข้อมูลจะเปลี่ยนเป็นข้อมูลที่เรียงลำดับแล้ว : เปลี่ยนลูกศรสีฟ้าให้กลายเป็นข้อมูลที่เรียงลำดับแล้ว

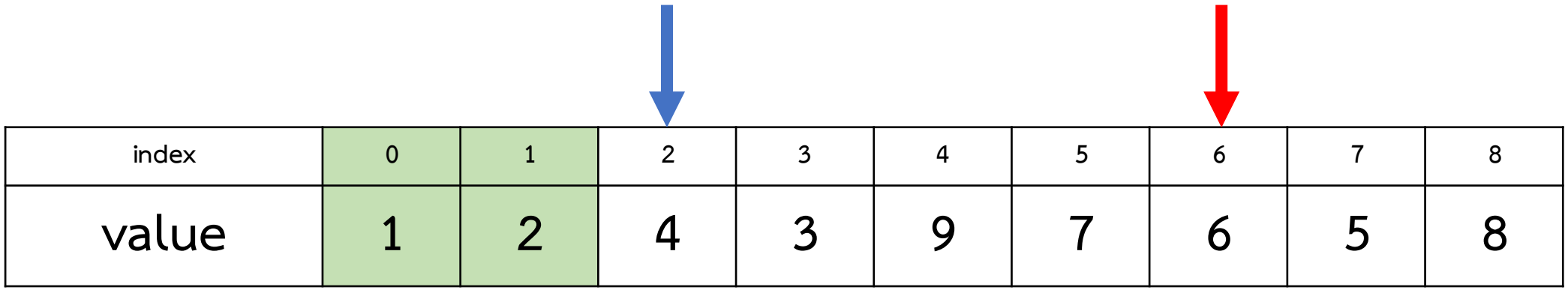


index	0	1	2	3	4	5	6	7	8
value	1	2	4	3	9	7	6	5	8

Sorting Algorithm: Selection Sort

ขั้นตอนการทำงาน

6. เฉพาะส่วนที่ยังไม่ได้เรียงลำดับ ให้ทำข้อ 2-5 ซ้ำ จนกว่าทุกข้อมูลจะเปลี่ยนเป็นข้อมูลที่เรียงลำดับแล้ว : ย้ายลูกศรสีฟ้าไปยังตำแหน่งแรกของข้อมูลที่ยังไม่ได้จัดเรียง

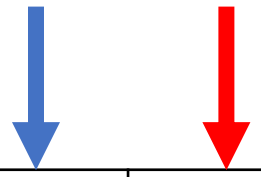


index	0	1	2	3	4	5	6	7	8
value	1	2	4	3	9	7	6	5	8

Sorting Algorithm: Selection Sort

ขั้นตอนการทำงาน

6. เฉพาะส่วนที่ยังไม่ได้เรียงลำดับ ให้ทำข้อ 2-5 ซ้ำ จนกว่าทุกข้อมูลจะเปลี่ยนเป็นข้อมูลที่เรียงลำดับแล้ว

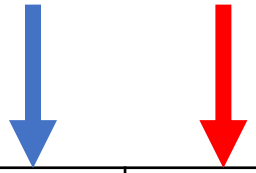


index	0	1	2	3	4	5	6	7	8
value	1	2	4	3	9	7	6	5	8

Sorting Algorithm: Selection Sort

ขั้นตอนการทำงาน

6. เฉพาะส่วนที่ยังไม่ได้เรียงลำดับ ให้ทำข้อ 2-5 ซ้ำ จนกว่าทุกข้อมูลจะเปลี่ยนเป็นข้อมูลที่เรียงลำดับแล้ว

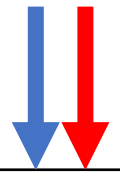


index	0	1	2	3	4	5	6	7	8
value	1	2	3	4	9	7	6	5	8

Sorting Algorithm: Selection Sort

ขั้นตอนการทำงาน

6. เฉพาะส่วนที่ยังไม่ได้เรียงลำดับ ให้ทำข้อ 2-5 ซ้ำ จนกว่าทุกข้อมูลจะเปลี่ยนเป็นข้อมูลที่เรียงลำดับแล้ว



index	0	1	2	3	4	5	6	7	8
value	1	2	3	4	9	7	6	5	8

Sorting Algorithm: Selection Sort

ขั้นตอนการทำงาน

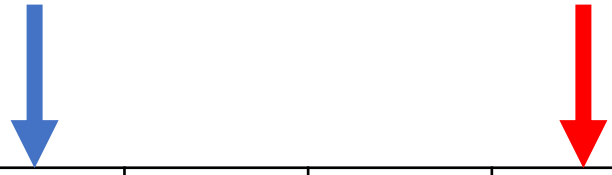
6. เฉพาะส่วนที่ยังไม่ได้เรียงลำดับ ให้ทำข้อ 2-5 ซ้ำ จนกว่าทุกข้อมูลจะเปลี่ยนเป็นข้อมูลที่เรียงลำดับแล้ว

index	0	1	2	3	4	5	6	7	8
value	1	2	3	4	9	7	6	5	8

Sorting Algorithm: Selection Sort

ขั้นตอนการทำงาน

6. เฉพาะส่วนที่ยังไม่ได้เรียงลำดับ ให้ทำข้อ 2-5 ซ้ำ จนกว่าทุกข้อมูลจะเปลี่ยนเป็นข้อมูลที่เรียงลำดับแล้ว

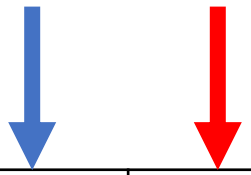


index	0	1	2	3	4	5	6	7	8
value	1	2	3	4	5	7	6	9	8

Sorting Algorithm: Selection Sort

ขั้นตอนการทำงาน

6. เฉพาะส่วนที่ยังไม่ได้เรียงลำดับ ให้ทำข้อ 2-5 ซ้ำ จนกว่าทุกข้อมูลจะเปลี่ยนเป็นข้อมูลที่เรียงลำดับแล้ว

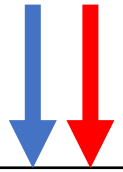


index	0	1	2	3	4	5	6	7	8
value	1	2	3	4	5	7	6	9	8

Sorting Algorithm: Selection Sort

ขั้นตอนการทำงาน

6. เฉพาะส่วนที่ยังไม่ได้เรียงลำดับ ให้ทำข้อ 2-5 ซ้ำ จนกว่าทุกข้อมูลจะเปลี่ยนเป็นข้อมูลที่เรียงลำดับแล้ว

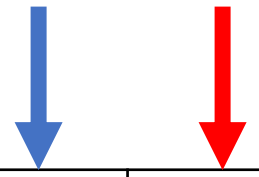


index	0	1	2	3	4	5	6	7	8
value	1	2	3	4	5	6	7	9	8

Sorting Algorithm: Selection Sort

ขั้นตอนการทำงาน

6. เฉพาะส่วนที่ยังไม่ได้เรียงลำดับ ให้ทำข้อ 2-5 ซ้ำ จนกว่าทุกข้อมูลจะเปลี่ยนเป็นข้อมูลที่เรียงลำดับแล้ว

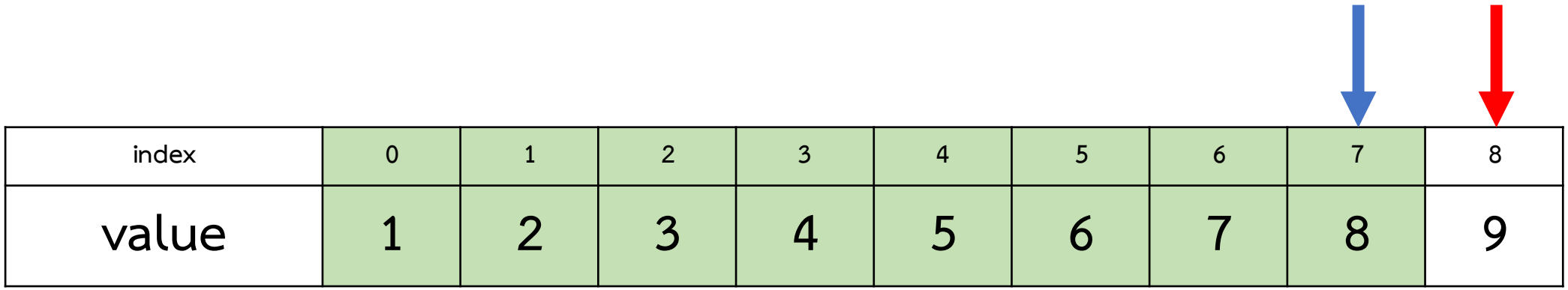


index	0	1	2	3	4	5	6	7	8
value	1	2	3	4	5	6	7	9	8

Sorting Algorithm: Selection Sort

ขั้นตอนการทำงาน

6. เฉพาะส่วนที่ยังไม่ได้เรียงลำดับ ให้ทำข้อ 2-5 ซ้ำ จนกว่าทุกข้อมูลจะเปลี่ยนเป็นข้อมูลที่เรียงลำดับแล้ว



index	0	1	2	3	4	5	6	7	8
value	1	2	3	4	5	6	7	8	9

Sorting Algorithm: Selection Sort

ขั้นตอนการทำงาน

6. เฉพาะส่วนที่ยังไม่ได้เรียงลำดับ ให้ทำข้อ 2-5 ซ้ำ จนกว่าทุกข้อมูลจะเปลี่ยนเป็นข้อมูลที่เรียงลำดับแล้ว

index	0	1	2	3	4	5	6	7	8
value	1	2	3	4	5	6	7	8	9

Sorting Algorithm: Selection Sort

index	0	1	2	3	4	5	6	7	8
Iter=0	7	6	4	3	9	1	2	5	8
Iter=1	1	6	4	3	9	7	2	5	8
Iter=2	1	2	4	3	9	7	6	5	8
Iter=3	1	2	3	4	9	7	6	5	8
Iter=4	1	2	3	4	9	7	6	5	8
Iter=5	1	2	3	4	5	7	6	9	8
Iter=6	1	2	3	4	5	6	7	9	8
Iter=7	1	2	3	4	5	6	7	9	8
Iter=8	1	2	3	4	5	6	7	8	9

Sorting Algorithm: Selection Sort

โจทย์ 1 จากอัลกอริทึม Selection Sort จงเขียนโปรแกรมเพื่อรับค่าจำนวนเต็มเข้าสู่อาเรย์จำนวน N ตัว จากนั้นแสดงผลการจัดเรียงด้วยวิธีการ Selection Sort ในแต่ละ iteration

ข้อมูลนำเข้า บรรทัดแรก จำนวนเต็ม N แทนจำนวนตัวของอาเรย์ที่ต้องการรับค่าเข้า
บรรทัดที่สอง จำนวนเต็ม N ตัว คั่นด้วยเว้นวรรค เป็นค่าที่ต้องการจัดเรียง

ข้อมูลขาออก มี N-1 บรรทัด แต่ละบรรทัดแทนผลลัพธ์ของแต่ละ iteration

Sorting Algorithm: Selection Sort

โจทย์ 1 จากอัลกอริทึม Selection Sort จงเขียนโปรแกรมเพื่อรับค่าจำนวนเต็มเข้าสู่อาเรย์จำนวน N ตัว จากนั้นแสดงผลการจัดเรียงด้วยวิธีการ Selection Sort ในแต่ละ iteration

ตัวอย่าง

ข้อมูลนำเข้า	ผลลัพธ์
5 5 4 3 1 2	1 4 3 5 2 1 2 3 5 4 1 2 3 5 4 1 2 3 4 5

Sorting Algorithm: Selection Sort

Sorting Algorithm: Selection Sort

วิเคราะห์ความซับซ้อนของ Selection Sort

```
19  void selectionSort(int arr[], int n) {
20      for (int i = 0; i < n - 1; ++i) {
21
22          int minIndex = i;
23          for (int j = i + 1; j < n; ++j) {
24              if (arr[j] < arr[minIndex]) {
25                  minIndex = j;
26              }
27          }
28
29          swap(arr[i], arr[minIndex]);
30
31          displayArray(arr, n);
32      }
33  }
34
```

$$O(N^2)$$

Sorting Algorithm: Selection Sort

วิเคราะห์จุดเด่น-จุดด้อยของ Selection Sort

จุดเด่น

1. ทำความเข้าใจได้ง่าย
2. ทำงานได้ดีกับข้อมูลเล็ก ๆ

จุดด้อย

1. ช้า เมื่อข้อมูลมีขนาดใหญ่ เนื่องจาก $O(N^2)$
2. เมื่อเรียนในขั้นสูงขึ้นไปจะมีประเด็นเรื่องความ Stable ของข้อมูลที่มีค่าเท่ากัน

Sorting Algorithm

วิธีการจัดเรียงที่จะกล่าวถึง จะเข้าถึงข้อมูลในอาเรย์ทุกตำแหน่งที่จะทำการจัดเรียง ซึ่งประกอบด้วยสมาชิก n ตัว

วิธีการจัดเรียงมีหลายวิธี โดยเราจะกล่าวถึงเป็นกลุ่ม ๆ ได้แก่

- Selection Sort, Bubble Sort, Insertion Sort
- Merge Sort, Quick Sort
- Heap Sort

Sorting Algorithm: Bubble Sort

Bubble Sort เป็นอัลกอริทึมการเรียงลำดับที่มีแนวคิดเรียบง่าย ทำงานโดยการสลับสมาชิกที่อยู่ติดกันหากพบว่าอยู่ในลำดับที่ไม่ถูกต้อง อัลกอริทึมนี้ไม่เหมาะสำหรับชุดข้อมูลที่มีขนาดใหญ่ เนื่องจากความซับซ้อนของเวลาเฉลี่ยสูงมาก

หากต้องการเรียงข้อมูลต่อไปนี้จากน้อยไปมาก

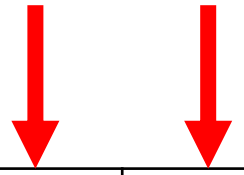
6	3	0	5	9

Sorting Algorithm: Bubble Sort

ขั้นตอนการทำงาน

1. ทำให้ตำแหน่งแรกถูกต้อง โดยการ:

   พิจารณาข้อมูลที่ติดกันทีละคู่ เรียงจากซ้ายไปขวา หากข้อมูลทางขวาน้อยกว่าข้อมูลทางซ้าย ให้สลับข้อมูล



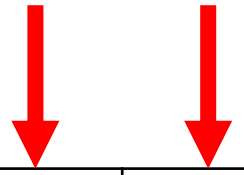
Index	0	1	2	3	4
value	6	3	0	5	9

Sorting Algorithm: Bubble Sort

ขั้นตอนการทำงาน

1. ทำให้ตำแหน่งแรกถูกต้อง โดยการ:

   พิจารณาข้อมูลที่ติดกันทีละคู่ เรียงจากซ้ายไปขวา หากข้อมูลทางขวาน้อยกว่าข้อมูลทางซ้าย ให้สลับข้อมูล



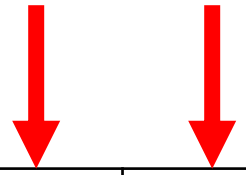
Index	0	1	2	3	4
value	3	6	0	5	9

Sorting Algorithm: Bubble Sort

ขั้นตอนการทำงาน

1. ทำให้ตำแหน่งแรกถูกต้อง โดยการ:

   พิจารณาข้อมูลที่ติดกันทีละคู่ เรียงจากซ้ายไปขวา หากข้อมูลทางขวาน้อยกว่าข้อมูลทางซ้าย ให้สลับข้อมูล



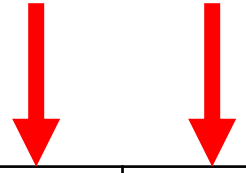
Index	0	1	2	3	4
value	3	6	0	5	9

Sorting Algorithm: Bubble Sort

ขั้นตอนการทำงาน

1. ทำให้ตำแหน่งแรกถูกต้อง โดยการ:

   พิจารณาข้อมูลที่ติดกันทีละคู่ เรียงจากซ้ายไปขวา หากข้อมูลทางขวาน้อยกว่าข้อมูลทางซ้าย ให้สลับข้อมูล



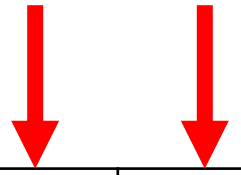
Index	0	1	2	3	4
value	3	0	6	5	9

Sorting Algorithm: Bubble Sort

ขั้นตอนการทำงาน

1. ทำให้ตำแหน่งแรกถูกต้อง โดยการ:

   พิจารณาข้อมูลที่ติดกันทีละคู่ เรียงจากซ้ายไปขวา หากข้อมูลทางขวาน้อยกว่าข้อมูลทางซ้าย ให้สลับข้อมูล



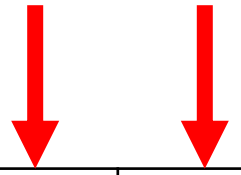
Index	0	1	2	3	4
value	3	0	6	5	9

Sorting Algorithm: Bubble Sort

ขั้นตอนการทำงาน

1. ทำให้ตำแหน่งแรกถูกต้อง โดยการ:

   พิจารณาข้อมูลที่ติดกันทีละคู่ เรียงจากซ้ายไปขวา หากข้อมูลทางขวาน้อยกว่าข้อมูลทางซ้าย ให้สลับข้อมูล



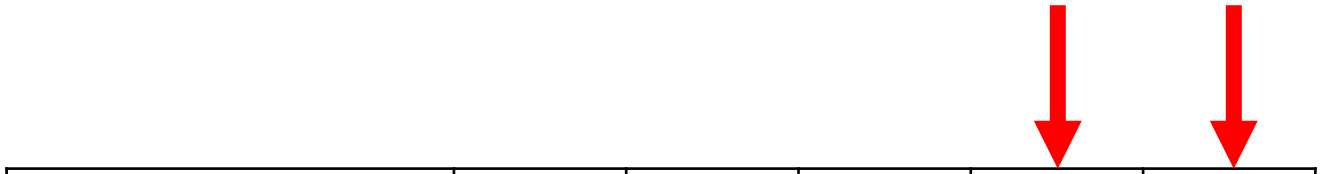
Index	0	1	2	3	4
value	3	0	5	6	9

Sorting Algorithm: Bubble Sort

ขั้นตอนการทำงาน

1. ทำให้ตำแหน่งแรกถูกต้อง โดยการ:

   พิจารณาข้อมูลที่ติดกันทีละคู่ เรียงจากซ้ายไปขวา หากข้อมูลทางขวาน้อยกว่าข้อมูลทางซ้าย ให้สลับข้อมูล



Index	0	1	2	3	4
value	3	0	5	6	9

Sorting Algorithm: Bubble Sort

ขั้นตอนการทำงาน

1. ทำให้ตำแหน่งแรกถูกต้อง โดยการ:

   พิจารณาข้อมูลที่ติดกันทีละคู่ เรียงจากซ้ายไปขวา หากข้อมูลทางขวาน้อยกว่าข้อมูลทางซ้าย ให้สลับข้อมูล

Index	0	1	2	3	4
value	3	0	5	6	9

Sorting Algorithm: Bubble Sort

ขั้นตอนการทำงาน

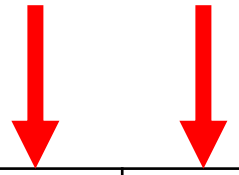
1. ทำให้ตำแหน่งแรกถูกต้อง
2. พิจารณาเฉพาะตำแหน่งที่ยังไม่ได้จัดเรียง ทำ 1 ซ้ำจนกว่าจะจัดเรียงจนครบ

Index	0	1	2	3	4
value	3	0	5	6	9

Sorting Algorithm: Bubble Sort

ขั้นตอนการทำงาน

1. ทำให้ตำแหน่งแรกถูกต้อง
2. พิจารณาเฉพาะตำแหน่งที่ยังไม่ได้จัดเรียง ทำ 1 ซ้ำจนกว่าจะจัดเรียงจนครบ

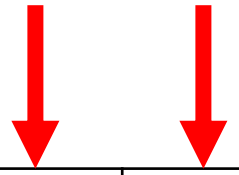


Index	0	1	2	3	4
value	3	0	5	6	9

Sorting Algorithm: Bubble Sort

ขั้นตอนการทำงาน

1. ทำให้ตำแหน่งแรกถูกต้อง
2. พิจารณาเฉพาะตำแหน่งที่ยังไม่ได้จัดเรียง ทำ 1 ซ้ำจนกว่าจะจัดเรียงจนครบ

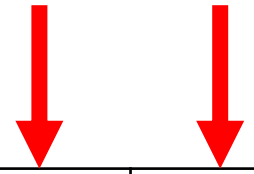


Index	0	1	2	3	4
value	0	3	5	6	9

Sorting Algorithm: Bubble Sort

ขั้นตอนการทำงาน

1. ทำให้ตำแหน่งแรกถูกต้อง
2. พิจารณาเฉพาะตำแหน่งที่ยังไม่ได้จัดเรียง ทำ 1 ซ้ำจนกว่าจะจัดเรียงจนครบ

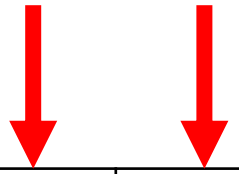


Index	0	1	2	3	4
value	0	3	5	6	9

Sorting Algorithm: Bubble Sort

ขั้นตอนการทำงาน

1. ทำให้ตำแหน่งแรกถูกต้อง
2. พิจารณาเฉพาะตำแหน่งที่ยังไม่ได้จัดเรียง ทำ 1 ซ้ำจนกว่าจะจัดเรียงจนครบ

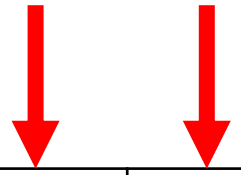


Index	0	1	2	3	4
value	0	3	5	6	9

Sorting Algorithm: Bubble Sort

ขั้นตอนการทำงาน

1. ทำให้ตำแหน่งแรกถูกต้อง
2. พิจารณาเฉพาะตำแหน่งที่ยังไม่ได้จัดเรียง ทำ 1 ซ้ำจนกว่าจะจัดเรียงจนครบ

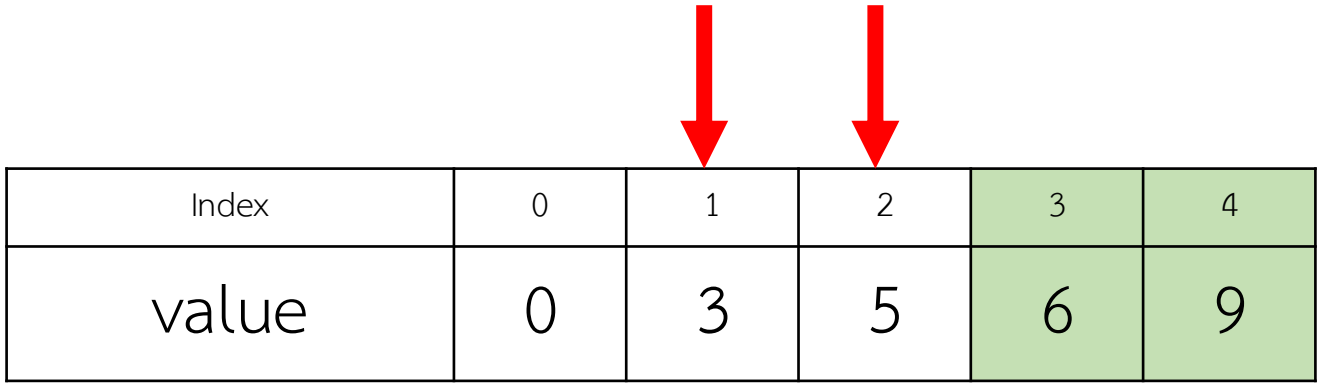


Index	0	1	2	3	4
value	0	3	5	6	9

Sorting Algorithm: Bubble Sort

ขั้นตอนการทำงาน

1. ทำให้ตำแหน่งแรกถูกต้อง
2. พิจารณาเฉพาะตำแหน่งที่ยังไม่ได้จัดเรียง ทำ 1 ซ้ำจนกว่าจะจัดเรียงจนครบ



Index	0	1	2	3	4
value	0	3	5	6	9

Sorting Algorithm: Bubble Sort

ขั้นตอนการทำงาน

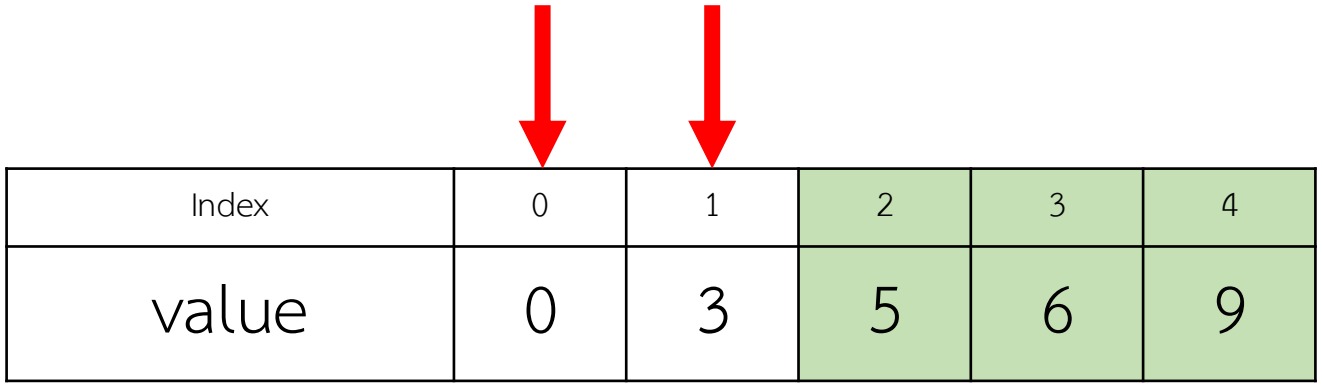
1. ทำให้ตำแหน่งแรกถูกต้อง
2. พิจารณาเฉพาะตำแหน่งที่ยังไม่ได้จัดเรียง ทำ 1 ซ้ำจนกว่าจะจัดเรียงจนครบ

Index	0	1	2	3	4
value	0	3	5	6	9

Sorting Algorithm: Bubble Sort

ขั้นตอนการทำงาน

1. ทำให้ตำแหน่งแรกถูกต้อง
2. พิจารณาเฉพาะตำแหน่งที่ยังไม่ได้จัดเรียง ทำ 1 ซ้ำจนกว่าจะจัดเรียงจนครบ



Index	0	1	2	3	4
value	0	3	5	6	9

Sorting Algorithm: Bubble Sort

ขั้นตอนการทำงาน

1. ทำให้ตำแหน่งแรกถูกต้อง
2. พิจารณาเฉพาะตำแหน่งที่ยังไม่ได้จัดเรียง ทำ 1 ซ้ำจนกว่าจะจัดเรียงจนครบ

Index	0	1	2	3	4
value	0	3	5	6	9

Sorting Algorithm: Bubble Sort

Index	0	1	2	3	4
Iter=0	6	3	0	5	9
Iter=1	3	6	0	5	9
Iter=2	3	0	6	5	9
Iter=3	3	0	5	6	9
Iter=4	3	0	5	6	9
Iter=5	0	3	5	6	9
Iter=6	0	3	5	6	9
Iter=7	0	3	5	6	9
Iter=8	0	3	5	6	9
Iter=9	0	3	5	6	9
Iter=10	0	3	5	6	9

$$no.of\ comparison = (n * (n-1))/2$$

Sorting Algorithm: Bubble Sort

โจทย์ 2 จากอัลกอริทึม Bubble Sort จงเขียนโปรแกรมเพื่อรับค่าจำนวนเต็มเข้าสู่อาเรย์จำนวน N ตัว จากนั้นแสดงผลการจัดเรียงด้วยวิธีการ Bubble Sort ในแต่ละ iteration

ข้อมูลนำเข้า บรรทัดแรก จำนวนเต็ม N แทนจำนวนตัวของอาเรย์ที่ต้องการรับค่าเข้า

บรรทัดที่สอง จำนวนเต็ม N ตัว คั่นด้วยเว้นวรรค เป็นค่าที่ต้องการจัดเรียง

ข้อมูลขาออก มี (N*(N-1))/2 บรรทัด แต่ละบรรทัดแทนผลลัพธ์ของแต่ละ iteration

Sorting Algorithm: Bubble Sort

โจทย์ 2 จากอัลกอริทึม Bubble Sort จงเขียนโปรแกรมเพื่อรับค่าจำนวนเต็มเข้าสู่อาเรย์จำนวน N ตัว จากนั้นแสดงผลการจัดเรียงด้วยวิธีการ Bubble Sort ในแต่ละ iteration

ตัวอย่าง

ข้อมูลนำเข้า	ผลลัพธ์
5 5 4 3 1 2	4 5 3 1 2 4 3 5 1 2 4 3 1 5 2 4 3 1 2 5 3 4 1 2 5 3 1 4 2 5 3 1 2 4 5 1 3 2 4 5 1 2 3 4 5 1 2 3 4 5

Sorting Algorithm: Bubble Sort

Sorting Algorithm: Bubble Sort

วิเคราะห์ความซับซ้อนของ Bubble Sort

```
17  void bubbleSort(int arr[], int n) {
18      for (int i = 0; i < n - 1; ++i) {
19          for (int j = 0; j < n - i - 1; ++j) {
20              if (arr[j] > arr[j + 1]) {
21                  swap(arr[j], arr[j + 1]);
22              }
23              displayArray(arr, n);
24          }
25      }
26  }
```

$$O(N^2)$$

Sorting Algorithm: Bubble Sort

วิเคราะห์จุดเด่น-จุดด้อยของ Bubble Sort

จุดเด่น

1. ทำความเข้าใจได้ง่าย
2. ไม่ต้องการพื้นที่ในการจัดเก็บข้อมูลเพิ่มเติม
3. เมื่อเรียนในขั้นสูงขึ้นไปจะพบว่าวิธีการนี้มีความ Stable กับลำดับของข้อมูลที่มีค่าเท่ากัน

จุดด้อย

1. ช้า เมื่อข้อมูลมีขนาดใหญ่ เนื่องจาก $O(N^2)$
2. เนื่องจากเป็นวิธีการที่มีพื้นฐานมาจากการเปรียบเทียบ ซึ่งเป็นต้นเหตุในการจำกัดประสิทธิภาพในบางกรณี (ทำให้ทำงานช้า)

Sorting Algorithm

วิธีการจัดเรียงที่จะกล่าวถึง จะเข้าถึงข้อมูลในอาเรย์ทุกตำแหน่งที่จะทำการจัดเรียง ซึ่งประกอบด้วยสมาชิก n ตัว

วิธีการจัดเรียงมีหลายวิธี โดยเราจะกล่าวถึงเป็นกลุ่ม ๆ ได้แก่

- Selection Sort, Bubble Sort, Insertion Sort
- Merge Sort, Quick Sort
- Heap Sort

Sorting Algorithm: Insertion Sort

Insertion Sort เป็นวิธีการเรียงลำดับที่ง่ายและทำงานในลักษณะเดียวกับการเรียงลำดับไพ่ในมือ โดยจะมีส่วนของไพ่ที่เรียงลำดับแล้วและส่วนที่ยังไม่ได้เรียงลำดับ คุณจะเลือกการ์ดจากส่วนที่ยังไม่ได้เรียงลำดับและวางไว้ในตำแหน่งที่ถูกต้องในส่วนที่เรียงลำดับแล้ว

หากต้องการเรียงข้อมูลต่อไปนี้จากน้อยไปมาก

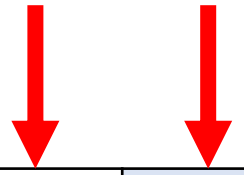
12	11	13	5	6

Sorting Algorithm: Insertion Sort

ขั้นตอนการทำงาน

1. ในขั้นแรก เราจะหาตำแหน่งที่เหมาะสมของข้อมูลในตำแหน่งที่ 1 ก่อน (11)

   พิจารณาข้อมูลในตำแหน่งที่ 0 และ 1 ถ้าพบว่าค่าในตำแหน่งที่ 1 น้อยกว่า ให้สลับข้อมูล



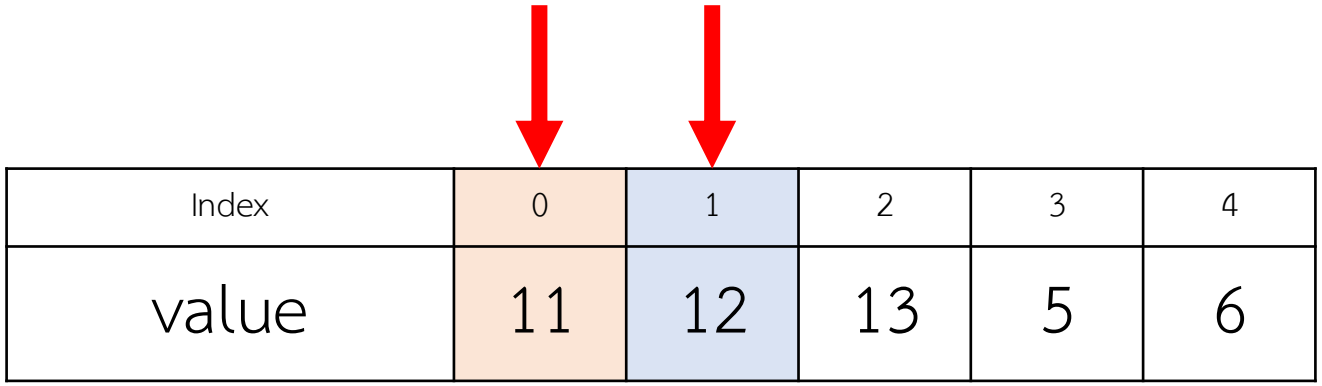
Index	0	1	2	3	4
value	12	11	13	5	6

Sorting Algorithm: Insertion Sort

ขั้นตอนการทำงาน

1. ในขั้นแรก เราจะหาตำแหน่งที่เหมาะสมของข้อมูลในตำแหน่งที่ 1 ก่อน

   โดยเราจะพิจารณาลำดับไปทางซ้ายอย่างเดียว ดังนั้นข้อมูลที่กำลังพิจารณาจะถูกส่งไปยังตำแหน่งที่เหมาะสมของข้อมูล (ถ้าค่าทางซ้ายยังมากกว่าตัวที่พิจารณาก็จะสลับต่อไปเรื่อย ๆ)



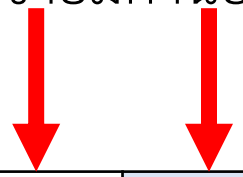
Index	0	1	2	3	4
value	11	12	13	5	6

Sorting Algorithm: Insertion Sort

ขั้นตอนการทำงาน

1. ในขั้นแรก เราจะหาตำแหน่งที่เหมาะสมของข้อมูลในตำแหน่งที่ 1 ก่อน
2. หาตำแหน่งที่เหมาะสมของข้อมูลในตำแหน่งที่ 2 บ้าง (13)

ตำแหน่งนี้เหมาะสมอยู่แล้ว เพราะค่าทางซ้ายมีค่าน้อยกว่า



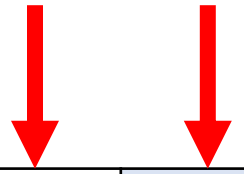
Index	0	1	2	3	4
value	11	12	13	5	6

Sorting Algorithm: Insertion Sort

ขั้นตอนการทำงาน

3. พิจารณาตำแหน่งที่ 3 และหาตำแหน่งที่เหมาะสม (5)

พบว่า 5 น้อยกว่าค่าทางซ้าย ทำการสลับตำแหน่ง

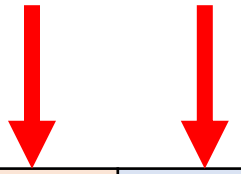


Index	0	1	2	3	4
value	11	12	13	5	6

Sorting Algorithm: Insertion Sort

ขั้นตอนการทำงาน

3. พิจารณาตำแหน่งที่ 3 และหาตำแหน่งที่เหมาะสม (5)



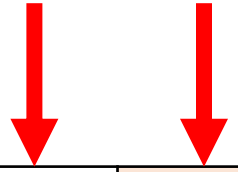
Index	0	1	2	3	4
value	11	12	5	13	6

Sorting Algorithm: Insertion Sort

ขั้นตอนการทำงาน

3. พิจารณาตำแหน่งที่ 3 และหาตำแหน่งที่เหมาะสม (5)

ตอนนี้ก็ยังพบว่า 5 น้อยกว่าค่าทางซ้าย ทำการสลับตำแหน่ง

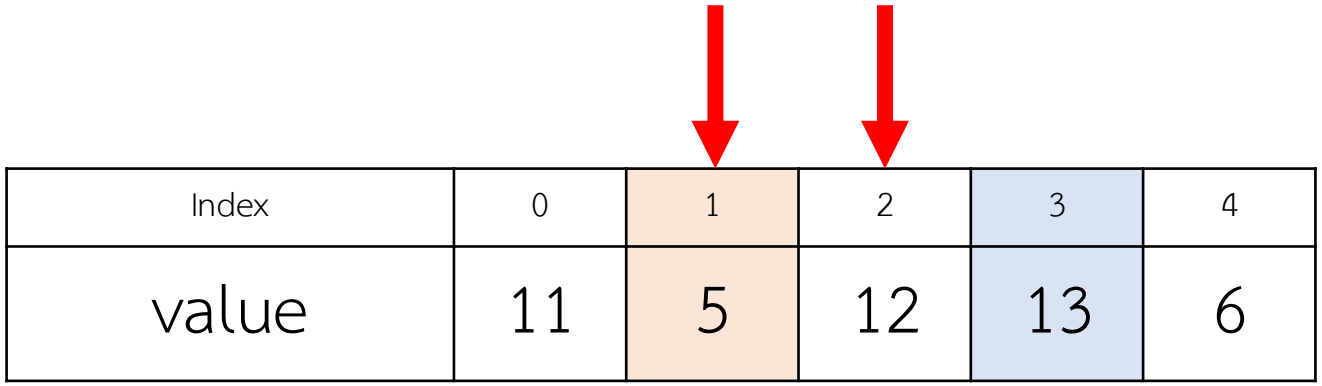


Index	0	1	2	3	4
value	11	12	5	13	6

Sorting Algorithm: Insertion Sort

ขั้นตอนการทำงาน

3. พิจารณาตำแหน่งที่ 3 และหาตำแหน่งที่เหมาะสม (5)



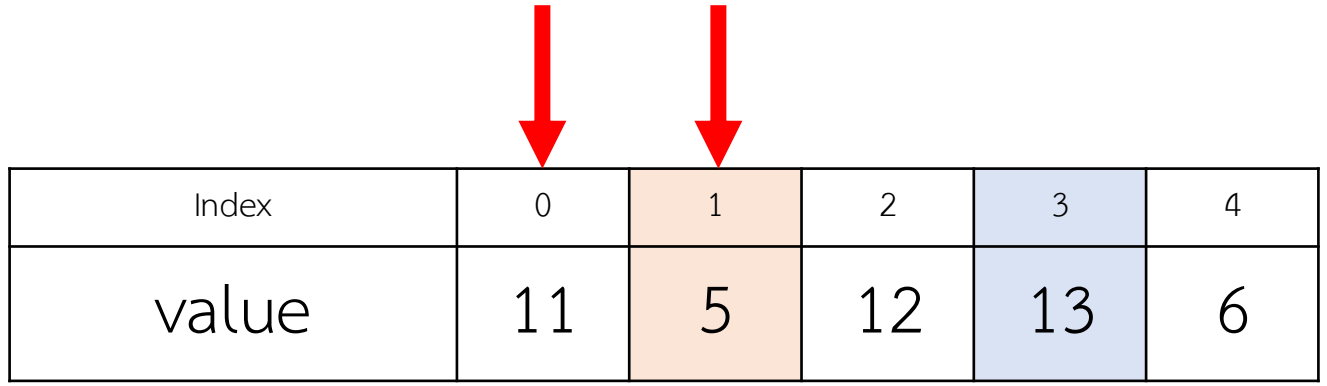
Index	0	1	2	3	4
value	11	5	12	13	6

Sorting Algorithm: Insertion Sort

ขั้นตอนการทำงาน

3. พิจารณาตำแหน่งที่ 3 และหาตำแหน่งที่เหมาะสม (5)

ตอนนี้ก็ยังพบว่า 5 น้อยกว่าค่าทางซ้าย ทำการสลับตำแหน่ง



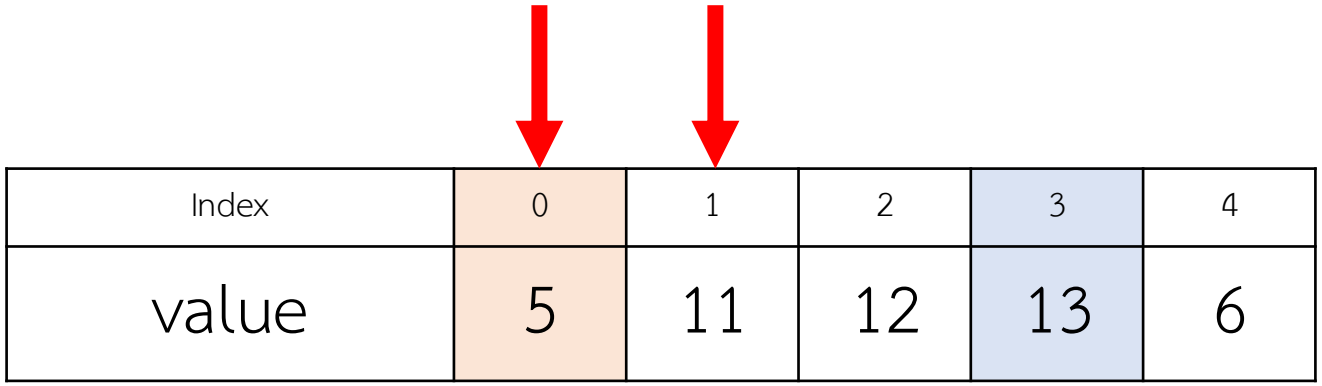
Index	0	1	2	3	4
value	11	5	12	13	6

Sorting Algorithm: Insertion Sort

ขั้นตอนการทำงาน

3. พิจารณาตำแหน่งที่ 3 และหาตำแหน่งที่เหมาะสม (5)

ตอนนี้ 5 ไปยังตำแหน่งแรกแล้ว ไม่มีตัวให้พิจารณาต่อจึงเปลี่ยนเป้าหมายไปยังตัวถัดไป



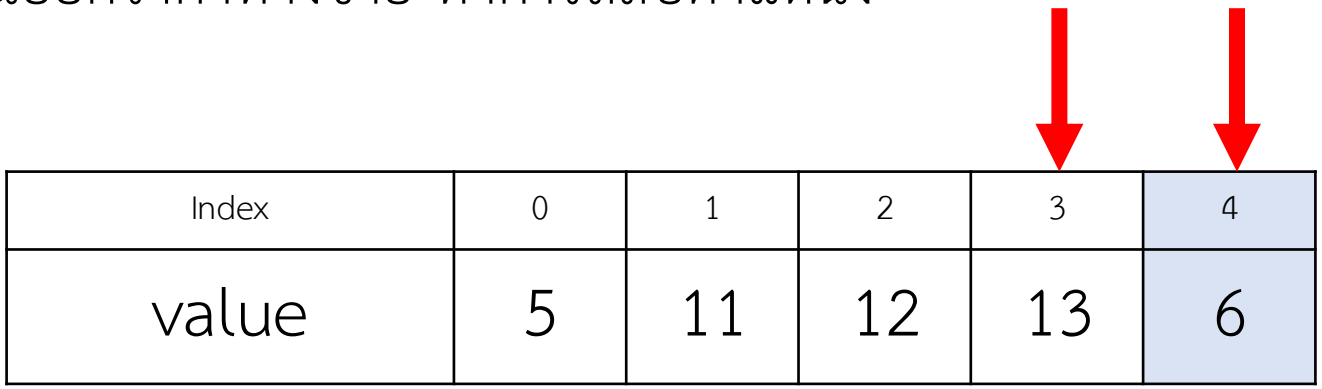
Index	0	1	2	3	4
value	5	11	12	13	6

Sorting Algorithm: Insertion Sort

ขั้นตอนการทำงาน

3. พิจารณาตำแหน่งที่ 3 และหาตำแหน่งที่เหมาะสม (5)
4. พิจารณาตำแหน่งที่ 4 และหาตำแหน่งที่เหมาะสม (6)

พบว่า 6 น้อยกว่าค่าทางซ้าย ทำการสลับตำแหน่ง

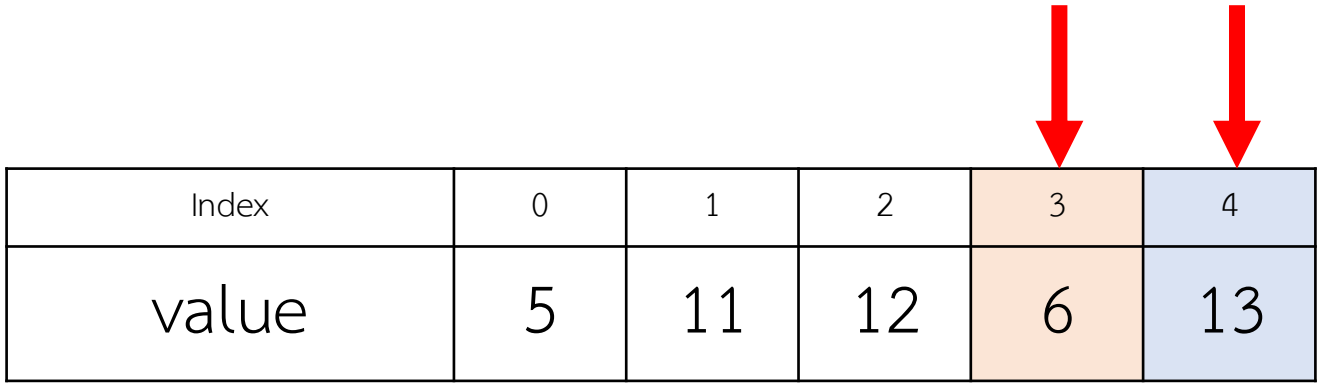


Index	0	1	2	3	4
value	5	11	12	13	6

Sorting Algorithm: Insertion Sort

ขั้นตอนการทำงาน

3. พิจารณาตำแหน่งที่ 3 และหาตำแหน่งที่เหมาะสม (5)
4. พิจารณาตำแหน่งที่ 4 และหาตำแหน่งที่เหมาะสม (6)



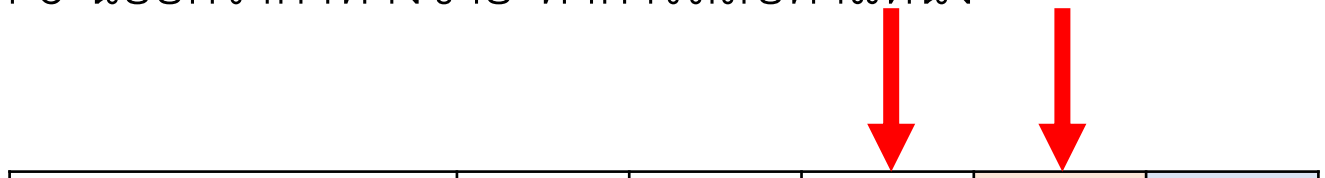
Index	0	1	2	3	4
value	5	11	12	6	13

Sorting Algorithm: Insertion Sort

ขั้นตอนการทำงาน

3. พิจารณาตำแหน่งที่ 3 และหาตำแหน่งที่เหมาะสม (5)
4. พิจารณาตำแหน่งที่ 4 และหาตำแหน่งที่เหมาะสม (6)

ยังคงพบว่า 6 น้อยกว่าค่าทางซ้าย ทำการสลับตำแหน่ง

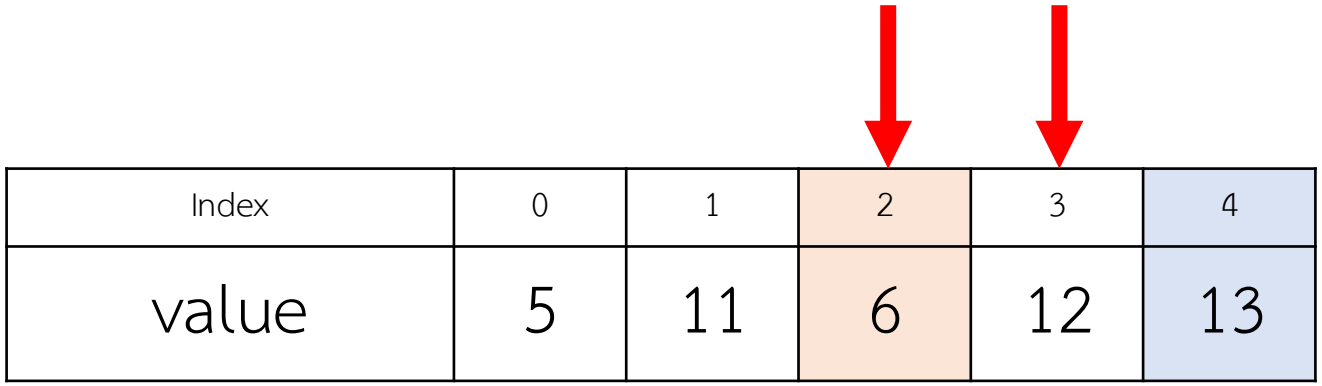


Index	0	1	2	3	4
value	5	11	12	6	13

Sorting Algorithm: Insertion Sort

ขั้นตอนการทำงาน

3. พิจารณาตำแหน่งที่ 3 และหาตำแหน่งที่เหมาะสม (5)
4. พิจารณาตำแหน่งที่ 4 และหาตำแหน่งที่เหมาะสม (6)



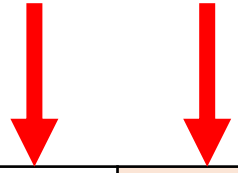
Index	0	1	2	3	4
value	5	11	6	12	13

Sorting Algorithm: Insertion Sort

ขั้นตอนการทำงาน

3. พิจารณาตำแหน่งที่ 3 และหาตำแหน่งที่เหมาะสม (5)
4. พิจารณาตำแหน่งที่ 4 และหาตำแหน่งที่เหมาะสม (6)

ยังคงพบว่า 6 น้อยกว่าค่าทางซ้าย ทำการสลับตำแหน่ง

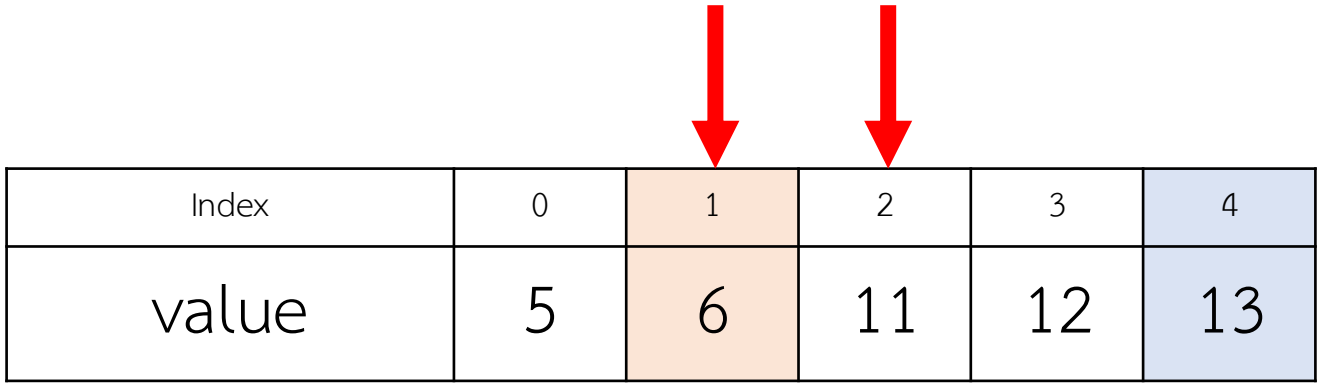


Index	0	1	2	3	4
value	5	11	6	12	13

Sorting Algorithm: Insertion Sort

ขั้นตอนการทำงาน

3. พิจารณาตำแหน่งที่ 3 และหาตำแหน่งที่เหมาะสม (5)
4. พิจารณาตำแหน่งที่ 4 และหาตำแหน่งที่เหมาะสม (6)



Index	0	1	2	3	4
value	5	6	11	12	13

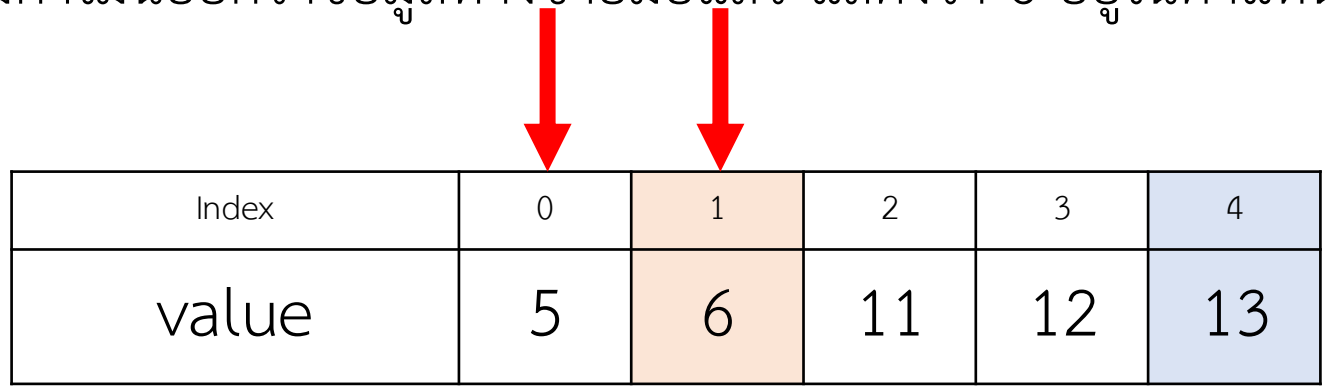
Sorting Algorithm: Insertion Sort

ขั้นตอนการทำงาน

3. พิจารณาตำแหน่งที่ 3 และหาตำแหน่งที่เหมาะสม (5)

4. พิจารณาตำแหน่งที่ 4 และหาตำแหน่งที่เหมาะสม (6)

ตอนนี้ 6 มีค่าไม่น้อยกว่าข้อมูลทางซ้ายมือแล้ว แสดงว่า 6 อยู่ในตำแหน่งที่เหมาะสม



Index	0	1	2	3	4
value	5	6	11	12	13

Sorting Algorithm: Insertion Sort

ขั้นตอนการทำงาน

3. พิจารณาตำแหน่งที่ 3 และหาตำแหน่งที่เหมาะสม (5)
4. พิจารณาตำแหน่งที่ 4 และหาตำแหน่งที่เหมาะสม (6)
5. ถ้ามีข้อมูลในตำแหน่งถัดไป ก็พิจารณาตำแหน่งถัดไปเรื่อย ๆ จนครบ

Index	0	1	2	3	4
value	5	6	11	12	13

Sorting Algorithm: Insertion Sort

Index	0	1	2	3	4
Iter=0	12	11	13	5	6
Iter=1	11	12	13	5	6
Iter=2	11	12	13	5	6
Iter=3	11	12	5	13	6
Iter=4	11	5	12	13	6
Iter=5	5	11	12	13	6
Iter=6	5	11	12	6	13
Iter=7	5	11	6	12	13
Iter=8	5	6	11	12	13

จำนวนครั้งที่ทำการเปรียบเทียบ ขึ้นอยู่กับการจัดเรียงข้อมูลตั้งต้น

Sorting Algorithm: Insertion Sort

โจทย์ 3 จากอัลกอริทึม Insertion Sort จงเขียนโปรแกรมเพื่อรับค่าจำนวนเต็มเข้าสู่อาเรย์จำนวน N ตัว จากนั้นแสดงผลการจัดเรียงด้วยวิธีการ Insertion Sort ในแต่ละ iteration

ข้อมูลนำเข้า	บรรทัดแรก จำนวนเต็ม N แทนจำนวนตัวของอาเรย์ที่ต้องการรับค่าเข้า
	บรรทัดที่สอง จำนวนเต็ม N ตัว คั่นด้วยเว้นวรรค เป็นค่าที่ต้องการจัดเรียง
ข้อมูลขาออก	มีจำนวนบรรทัดไม่แน่นอน ขึ้นอยู่กับการจัดเรียงของข้อมูลตั้งต้น
	แต่ละบรรทัดแทนผลลัพธ์ของแต่ละ iteration

Sorting Algorithm: Insertion Sort

โจทย์ 3 จากอัลกอริทึม Insertion Sort จงเขียนโปรแกรมเพื่อรับค่าจำนวนเต็มเข้าสู่อาเรย์จำนวน N ตัว จากนั้นแสดงผลการจัดเรียงด้วยวิธีการ Insertion Sort ในแต่ละ iteration

ตัวอย่าง

ข้อมูลนำเข้า	ผลลัพธ์
5 4 7 1 3 2	4 1 7 3 2 1 4 7 3 2 1 4 3 7 2 1 3 4 7 2 1 3 4 2 7 1 3 2 4 7 1 2 3 4 7

Sorting Algorithm: Insertion Sort

Sorting Algorithm: Insertion Sort

วิเคราะห์ความซับซ้อนของ Insertion Sort

```
17  void insertionSort(int arr[], int n) {
18      for (int i = 1; i < n; ++i) {
19          int key = arr[i];
20          int j = i - 1;
21          while (j >= 0 && arr[j] > key) {
22              swap(arr[j], arr[j+1]);
23              j = j - 1;
24              displayArray(arr, n);
25          }
26      }
27  }
```

Worst case: $O(N^2)$

Avg case: $O(N^2)$

Best case: $O(N)$

Sorting Algorithm: Insertion Sort

วิเคราะห์จุดเด่น-จุดด้อยของ Insertion Sort

จุดเด่น

1. ทำความเข้าใจได้ง่าย
2. ถ้าข้อมูลมีลักษณะเรียงกันอยู่แล้วก่อนนำมาเข้ากระบวนการ จะทำให้เวลาที่ใช้ดำเนินการสั้นลง
3. การออกแบบอัลกอริทึมมีความยืดหยุ่นและเข้ากับข้อมูลที่พบได้ทั่วไป ที่มักมีการจัดเรียงข้อมูลมาบางส่วนอยู่แล้ว

จุดด้อย

1. ช้า เมื่อข้อมูลมีขนาดใหญ่ เนื่องจาก $O(N^2)$
2. ถ้าข้อมูลจัดเรียงแบบตรงข้ามกับสิ่งที่ต้องการ ก็จะให้ผลแย่ไม่ต่างจากวิธี Selection และ Bubble

Sorting Algorithm

```
19  void selectionSort(int arr[], int n) {
20      for (int i = 0; i < n - 1; ++i) {
21
22          int minIndex = i;
23          for (int j = i + 1; j < n; ++j) {
24              if (arr[j] < arr[minIndex]) {
25                  minIndex = j;
26              }
27          }
28
29          swap(arr[i], arr[minIndex]);
30
31          displayArray(arr, n);
32      }
33  }
34
```

```
17  void bubbleSort(int arr[], int n) {
18      for (int i = 0; i < n - 1; ++i) {
19          for (int j = 0; j < n - i - 1; ++j) {
20              if (arr[j] > arr[j + 1]) {
21                  swap(arr[j], arr[j + 1]);
22              }
23              displayArray(arr, n);
24          }
25      }
26  }
```

```
17  void insertionSort(int arr[], int n) {
18      for (int i = 1; i < n; ++i) {
19          int key = arr[i];
20          int j = i - 1;
21          while (j >= 0 && arr[j] > key) {
22              swap(arr[j], arr[j+1]);
23              j = j - 1;
24              displayArray(arr, n);
25          }
26      }
27  }
```

Sorting Algorithm: โจทย์ปัญหา

โจทย์ 4 จงเขียนโปรแกรมที่รับค่าจำนวนเต็ม N และอาเรย์ที่เก็บจำนวนเต็มขนาด N ตัว เข้ามา จากนั้นรับค่าจำนวนเต็ม k ที่ k<N เพื่อระบุว่าตัวเลขที่มากที่สุดเป็นอันดับที่ k มีค่าเท่าใด

ข้อมูลนำเข้า : บรรทัดแรก จำนวนเต็ม N

บรรทัดที่ 2 อาเรย์ของจำนวนเต็ม จำนวน N ตัว คั่นด้วยเว้นวรรค

บรรทัดที่ 3 จำนวนเต็ม k

ข้อมูลส่งออก : เป็นตัวเลขจำนวนเต็มตัวเดียว ระบุตัวเลขที่มากที่สุดอันดับที่ k ในอาเรย์

Sorting Algorithm: โจทย์ปัญหา

โจทย์ 4 จงเขียนโปรแกรมที่รับค่าจำนวนเต็ม N และอาเรย์ที่เก็บจำนวนเต็มขนาด N ตัว เข้ามา จากนั้นรับค่าจำนวนเต็ม k ที่ k<N เพื่อระบุว่าตัวเลขที่มากที่สุดเป็นอันดับที่ k มีค่าเท่าใด

ข้อมูลนำเข้า : บรรทัดแรก จำนวนเต็ม N

บรรทัดที่ 2 อาเรย์ของจำนวนเต็ม จำนวน N ตัว คั่นด้วยเว้นวรรค

บรรทัดที่ 3 จำนวนเต็ม k

ข้อมูลส่งออก : เป็นตัวเลขจำนวนเต็มตัวเดียว ระบุตัวเลขที่มากที่สุดอันดับที่ k ในอาเรย์

Sorting Algorithm: Insertion Sort

โจทย์ 4 จงเขียนโปรแกรมที่รับค่าจำนวนเต็ม N และอาเรย์ที่เก็บจำนวนเต็มขนาด N ตัวเข้ามา จากนั้นรับค่าจำนวนเต็ม k ที่ k<N เพื่อระบุว่าตัวเลขที่มากที่สุดเป็นอันดับที่ k มีค่าเท่าใด

ตัวอย่าง

ข้อมูลนำเข้า	ผลลัพธ์
10 9 5 12 65 42 13 28 1 99 4 4	28
5 1 4 3 3 2 3	3

Sorting Algorithm

Sorting Algorithm

โจทย์ 5 จากอัลกอริทึม Sort ทั้งสามตัวที่เรียนมา จงเขียนโปรแกรมเพื่อรับค่าจำนวนเต็มเข้าสู่อาเรย์จำนวน N ตัว จากนั้นแสดงผลการจัดเรียงด้วยวิธีการ Sort ในแต่ละ iteration แบบจัดเรียง**จากมากไปน้อย**

ข้อมูลนำเข้า บรรทัดแรก จำนวนเต็ม N แทนจำนวนตัวของอาเรย์ที่ต้องการรับค่าเข้า
บรรทัดที่สอง จำนวนเต็ม N ตัว คั่นด้วยเว้นวรรค เป็นค่าที่ต้องการจัดเรียง

ข้อมูลขาออก มีจำนวนบรรทัดไม่แน่นอน ขึ้นอยู่กับวิธีการที่เลือกใช้ และอาจเกี่ยวข้องกับการจัดเรียงของข้อมูลตั้งต้นด้วย แต่ละบรรทัดแทนผลลัพธ์ของแต่ละ iteration

Sorting Algorithm

วิธีการจัดเรียงที่จะกล่าวถึง จะเข้าถึงข้อมูลในอาเรย์ทุกตำแหน่งที่จะทำการจัดเรียง ซึ่งประกอบด้วยสมาชิก n ตัว

วิธีการจัดเรียงมีหลายวิธี โดยเราจะกล่าวถึงเป็นกลุ่ม ๆ ได้แก่

- Selection Sort, Bubble Sort, Insertion Sort
- Merge Sort, Quick Sort
- Heap Sort

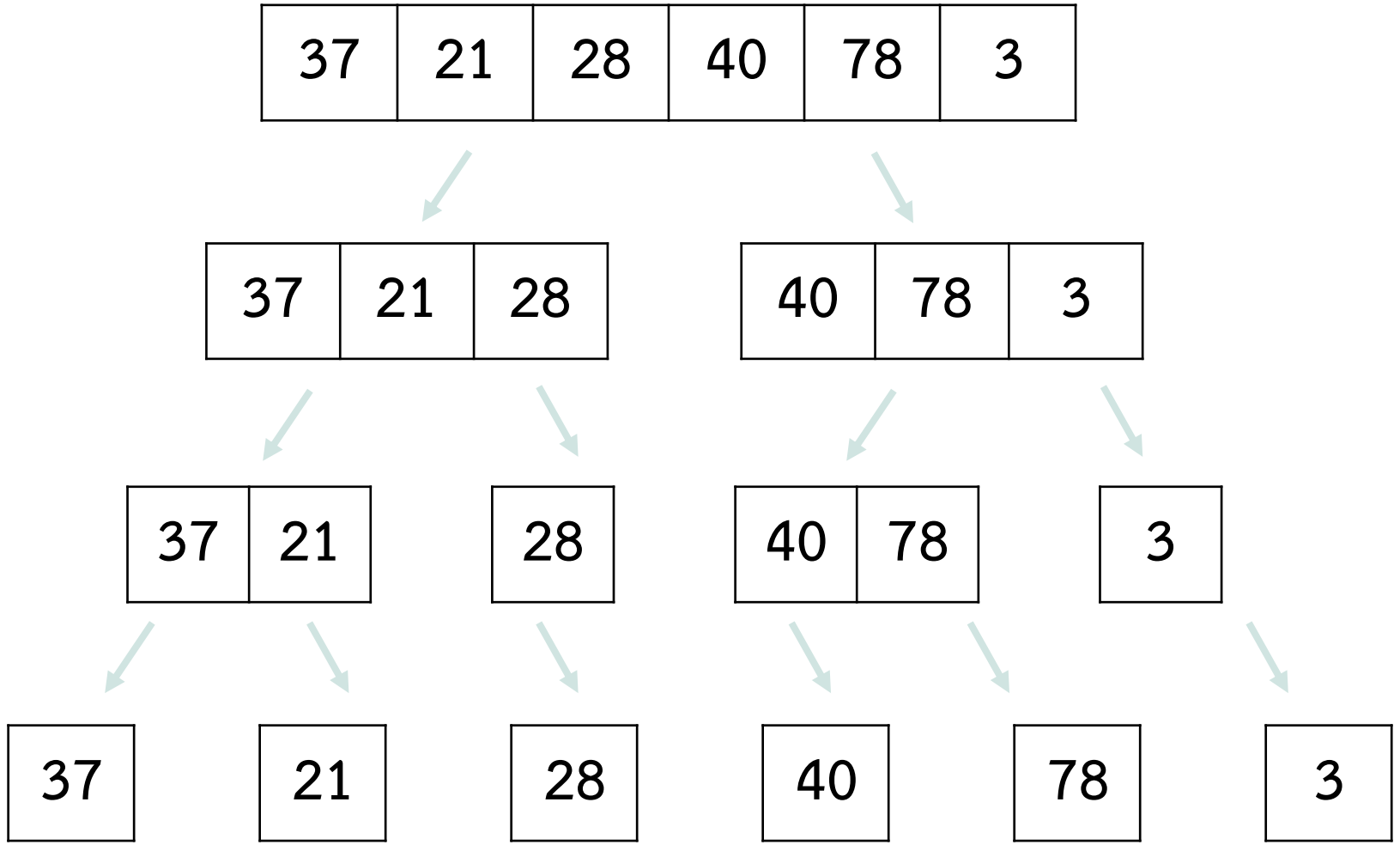
Sorting Algorithm: Merge Sort

Merge sort คือการแบ่งอาเรย์เป็นครึ่งหนึ่ง จัดเรียงแต่ละครึ่ง และจากนั้นรวมครึ่งที่เรียงลำดับแล้วกลับมารวมกัน กระบวนการนี้จะทำซ้ำไปเรื่อย ๆ จนกว่าอาเรย์ทั้งหมดจะถูกเรียงลำดับ มีวิธีการซับซ้อนกว่าวิธีการในกลุ่มแรก (ใช้ Recursive, หลักการ Divide and Conquer) แต่เร็วกว่ามาก

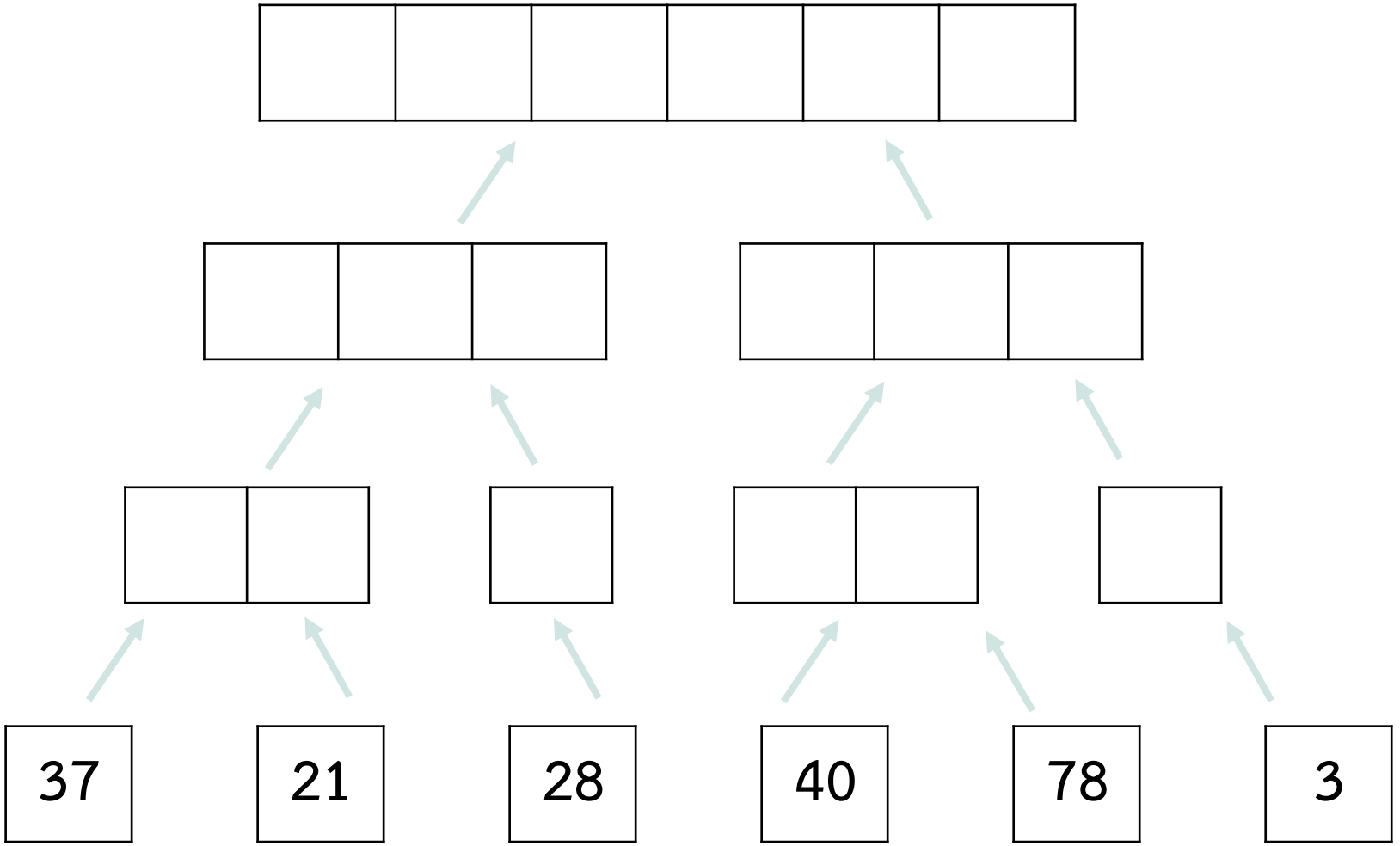
หากต้องการเรียงข้อมูลต่อไปนี้จากน้อยไปมาก

37	21	28	40	78	3

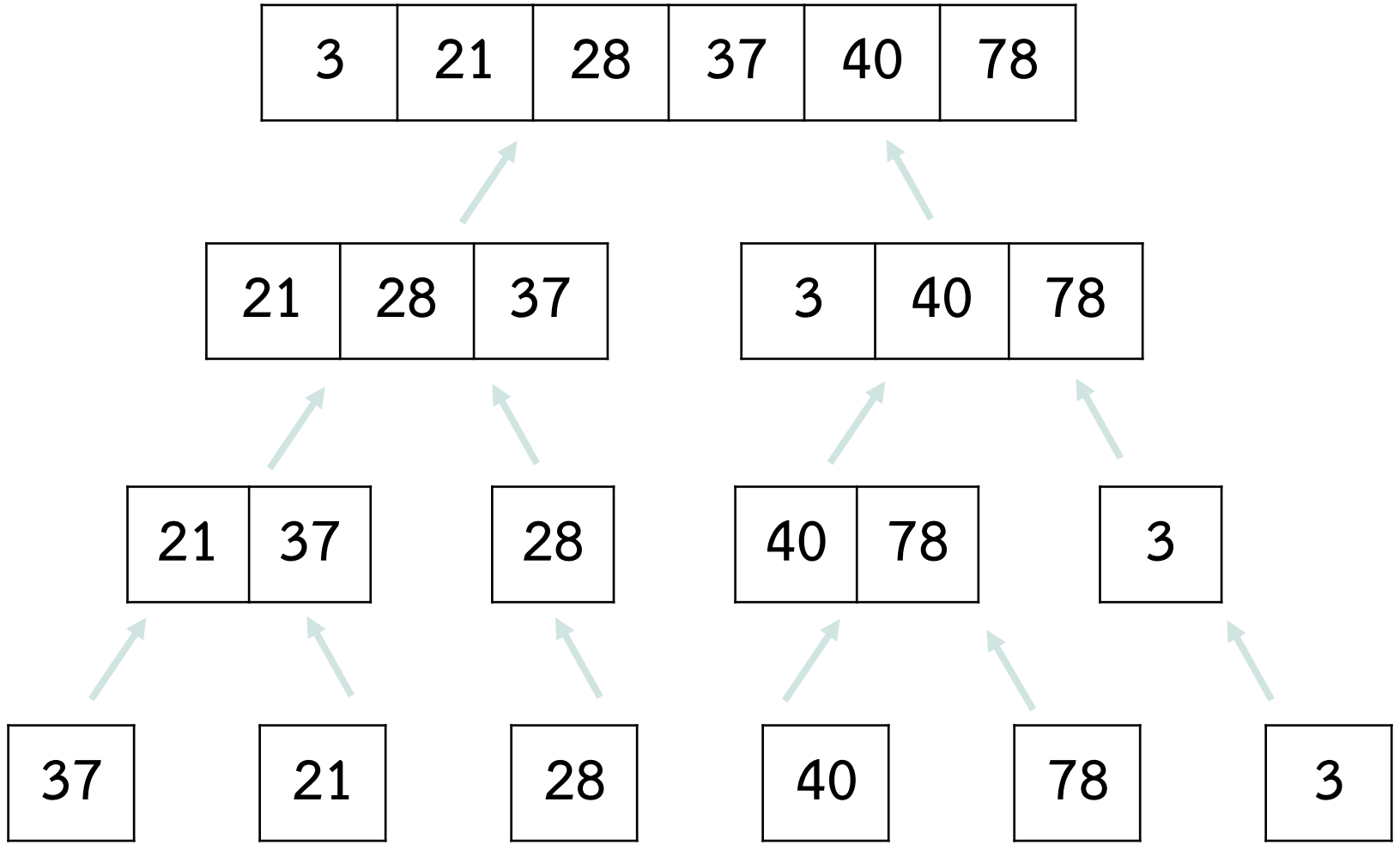
Sorting Algorithm: Merge Sort



Sorting Algorithm: Merge Sort



Sorting Algorithm: Merge Sort



Sorting Algorithm: Merge Sort

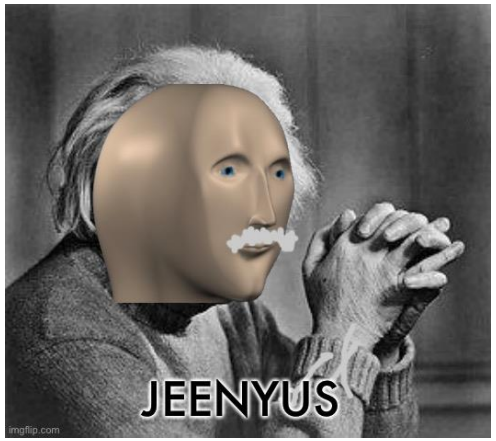
```cpp
#include<iostream>
using namespace std;

void merge(int array[], int const left, int const mid,
           int const right)
{
    int const subArrayOne = mid - left + 1;
    int const subArrayTwo = right - mid;

    auto *leftArray = new int[subArrayOne],
         *rightArray = new int[subArrayTwo];

    for (auto i = 0; i < subArrayOne; i++)
        leftArray[i] = array[left + i];
    for (auto j = 0; j < subArrayTwo; j++)
        rightArray[j] = array[mid + 1 + j];

    auto indexOfSubArrayOne = 0, indexOfSubArrayTwo = 0;
    int indexOfMergedArray = left;
```




```cpp
    while (indexOfSubArrayOne < subArrayOne
           && indexOfSubArrayTwo < subArrayTwo) {
        if (leftArray[indexOfSubArrayOne]
            <= rightArray[indexOfSubArrayTwo]) {
            array[indexOfMergedArray]
                = leftArray[indexOfSubArrayOne];
            indexOfSubArrayOne++;
        }
        else {
            array[indexOfMergedArray]
                = rightArray[indexOfSubArrayTwo];
            indexOfSubArrayTwo++;
        }
        indexOfMergedArray++;
    }

    while (indexOfSubArrayOne < subArrayOne) {
        array[indexOfMergedArray]
            = leftArray[indexOfSubArrayOne];
        indexOfSubArrayOne++;
        indexOfMergedArray++;
    }

    while (indexOfSubArrayTwo < subArrayTwo) {
        array[indexOfMergedArray]
            = rightArray[indexOfSubArrayTwo];
        indexOfSubArrayTwo++;
        indexOfMergedArray++;
    }
    delete[] leftArray;
    delete[] rightArray;
}
```

```cpp
void mergeSort(int array[], int const begin, int const end)
{
    if (begin >= end)
        return;

    int mid = begin + (end - begin) / 2;
    mergeSort(array, begin, mid);
    mergeSort(array, mid + 1, end);
    merge(array, begin, mid, end);
}

void printArray(int A[], int size)
{
    for (int i = 0; i < size; i++)
        cout << A[i] << " ";
    cout << endl;
}

int main()
{
    int arr[] = { 37, 21, 28, 40, 78, 3 };
    int arr_size = sizeof(arr) / sizeof(arr[0]);

    cout << "Given array is \n";
    printArray(arr, arr_size);

    mergeSort(arr, 0, arr_size - 1);

    cout << "\nSorted array is \n";
    printArray(arr, arr_size);
    return 0;
}
```

Sorting Algorithm: Merge Sort

วิเคราะห์ความซับซ้อนของ Merge Sort

- หั่นทีละครึ่ง จำนวน n รอบ ดังนั้นความซับซ้อนคือ $O(n \log n)$
- เร็วกว่ากลุ่ม 3 วิธีแรก (Selection, Bubble, Insertion) อย่างมากเมื่อข้อมูลมีขนาดใหญ่

Sorting Algorithm: Merge Sort

วิเคราะห์จุดเด่น-จุดด้อยของ Merge Sort

จุดเด่น

1. การันตี Worst case ที่ให้ความซับซ้อนระดับ $O(n \log n)$ เท่านั้น
2. สามารถรันแบบคู่ขนานได้ (Parallel) โดยใช้ประโยชน์จากโปรเซสเซอร์หรือ Threads

จุดด้อย

1. ช้ากว่าวิธีการที่เรียบง่ายเมื่อข้อมูลมีขนาดเล็กหรือจัดเรียงมาบ้างแล้ว เพราะในบางเคส ให้ความซับซ้อนสูงกว่า Selection Sort
2. ต้องใช้หน่วยความจำมากกว่า ทำให้เสียเปรียบเมื่อหน่วยความจำมีจำกัดมาก

Sorting Algorithm

วิธีการจัดเรียงที่จะกล่าวถึง จะเข้าถึงข้อมูลในอาเรย์ทุกตำแหน่งที่จะทำการจัดเรียง ซึ่งประกอบด้วยสมาชิก n ตัว

วิธีการจัดเรียงมีหลายวิธี โดยเราจะกล่าวถึงเป็นกลุ่ม ๆ ได้แก่

- Selection Sort, Bubble Sort, Insertion Sort
- Merge Sort, Quick Sort
- Heap Sort

Sorting Algorithm: Quick Sort

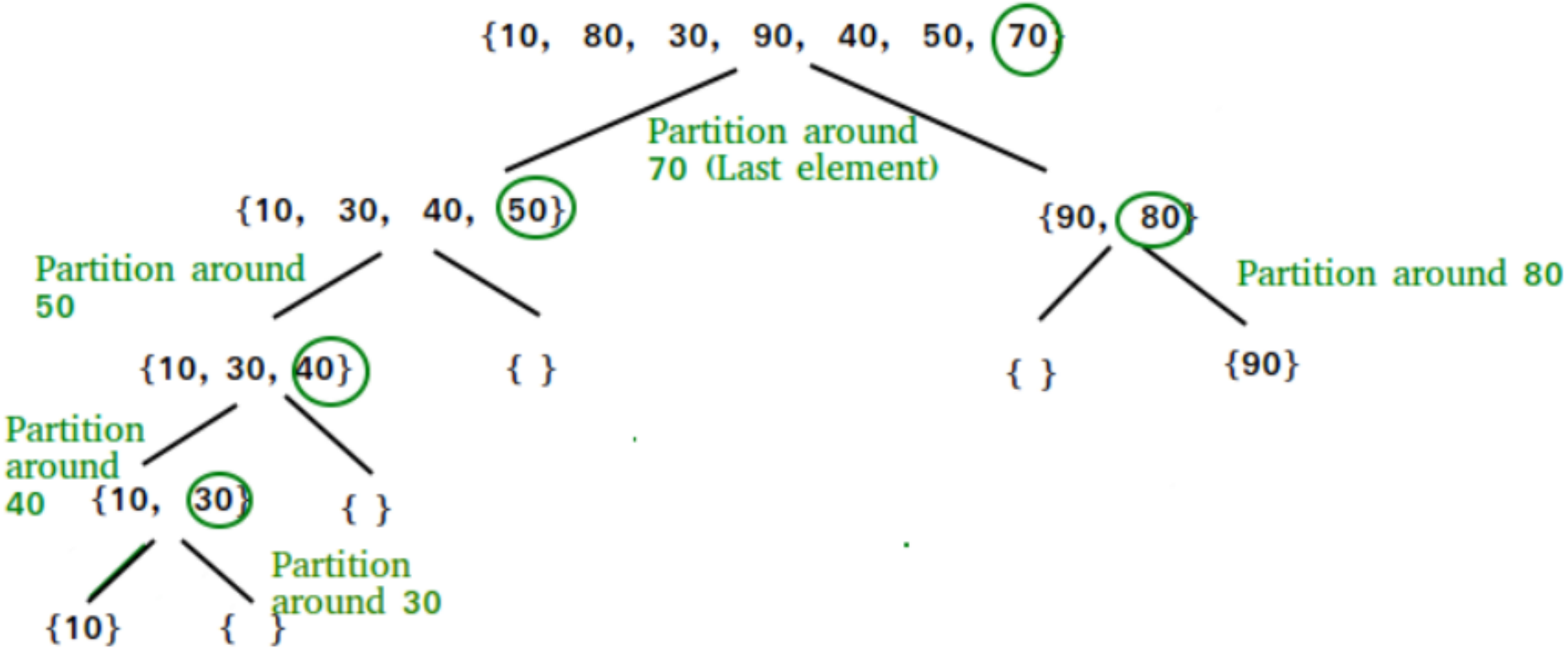
Quick Sort เป็นอัลกอริทึมในการเรียงลำดับที่ใช้หลักการของ Divide and Conquer โดยการเลือกสมาชิกหนึ่งตัวเป็น pivot และแบ่งแยกอาเรย์ที่กำหนดให้อยู่รอบๆ pivot โดยการวาง pivot ในตำแหน่งที่ถูกต้องของอาเรย์ที่เรียงลำดับไว้แล้วให้ชัดเจน

หัวใจสำคัญของ Quick Sort คือการแบ่ง Partition โดยเราจะหาตำแหน่งของ Partition ที่เหมาะสมและวางข้อมูลตัวที่เป็น Pivot ลงไป

หากต้องการเรียงข้อมูลต่อไปนี้จากน้อยไปมาก

10	80	30	90	40	50	70

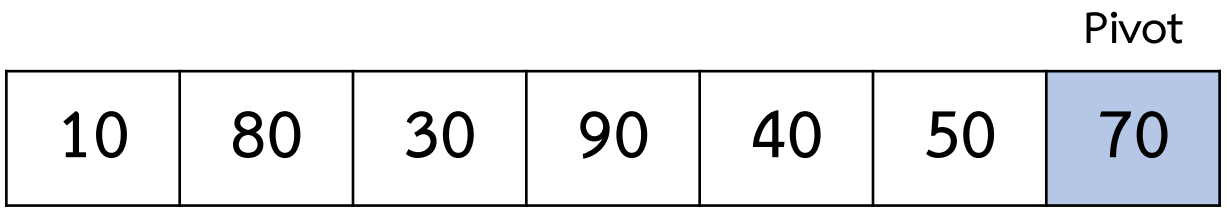
Sorting Algorithm: Quick Sort



Sorting Algorithm: Quick Sort

ขั้นตอนวิธี

1. เลือก Pivot (มีวิธีการเลือกหลายวิธี อาจเลือกจากตัวซ้ายสุด ตัวกลาง หรือตัวขวา ในที่นี้จะเลือกตัวขวาสุดเสมอ)

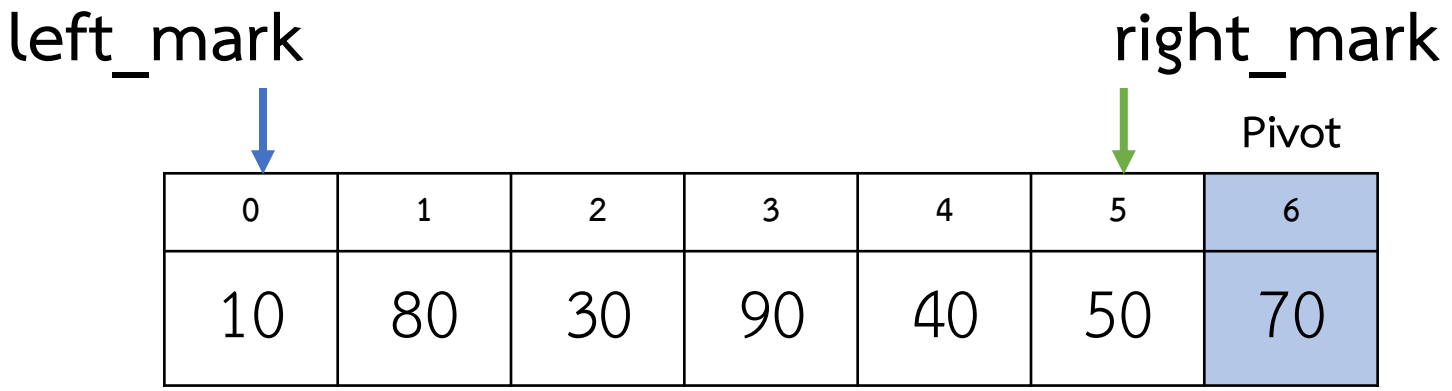


https://runestone.academy/ns/books/published/pythonds/SortSearch/TheQuickSort.html

Sorting Algorithm: Quick Sort

ขั้นตอนวิธี

1. เลือก Pivot
2. กำหนด left_mark และ right_mark เพื่อชี้ตำแหน่งสำหรับเตรียมการสลับที่



https://runestone.academy/ns/books/published/pythonds/SortSearch/TheQuickSort.html

Sorting Algorithm: Quick Sort

ขั้นตอนวิธี

3. การปรับตำแหน่ง และการดำเนินการกับ left_mark และ right_mark มีกฎเกณฑ์ดังนี้
   i. ถ้าค่าในตำแหน่ง left_mark < pivot แล้ว left_mark จะเลื่อนไปทางขวาจนกว่า ค่าในตำแหน่ง left_mark > pivot
   ii. ถ้าค่าในตำแหน่ง right_mark > pivot แล้ว right_mark ก็จะเลื่อนไปทางซ้ายจนกว่า ค่าในตำแหน่ง right_mark < pivot
   iii. สลับค่าในตำแหน่ง left_mark และ right_mark
   iv. ทำซ้ำ ข้อ i, ii, iii จนทั้ง left_mark และ right_mark สลับด้านกัน

https://runestone.academy/ns/books/published/pythonds/SortSearch/TheQuickSort.html

Sorting Algorithm: Quick Sort

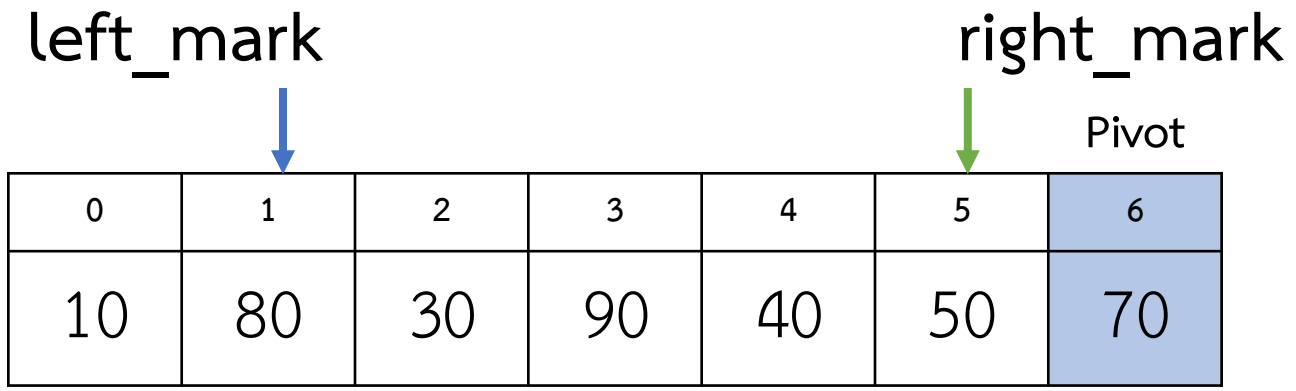
ขั้นตอนวิธี

i. ถ้าค่าในตำแหน่ง left_mark < pivot แล้ว left_mark จะเลื่อนไปทางขวาจนกว่าค่าในตำแหน่ง left_mark > pivot

ii. ถ้าค่าในตำแหน่ง right_mark > pivot แล้ว right_mark ก็จะเลื่อนไปทางซ้ายจนกว่า ค่าในตำแหน่ง right_mark < pivot

iii. สลับค่าในตำแหน่ง left_mark และ right_mark

iv. ทำซ้ำ ข้อ i, ii, iii จนทั้ง left_mark และ right_mark สลับด้านกัน



https://runestone.academy/ns/books/published/pythonds/SortSearch/TheQuickSort.html

Sorting Algorithm: Quick Sort

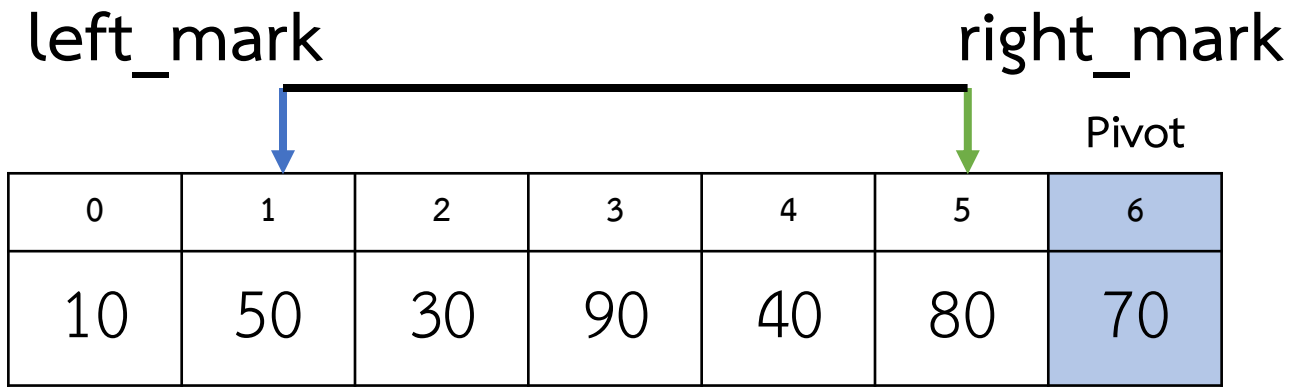
ขั้นตอนวิธี

i. ถ้าค่าในตำแหน่ง left_mark < pivot แล้ว left_mark จะเลื่อนไปทางขวาจนกว่าค่าในตำแหน่ง left_mark > pivot

ii. ถ้าค่าในตำแหน่ง right_mark > pivot แล้ว right_mark ก็จะเลื่อนไปทางซ้ายจนกว่า ค่าในตำแหน่ง right_mark < pivot

iii. สลับค่าในตำแหน่ง left_mark และ right_mark

iv. ทำซ้ำ ข้อ i, ii, iii จนทั้ง left_mark และ right_mark สลับด้านกัน



https://runestone.academy/ns/books/published/pythonds/SortSearch/TheQuickSort.html

Sorting Algorithm: Quick Sort

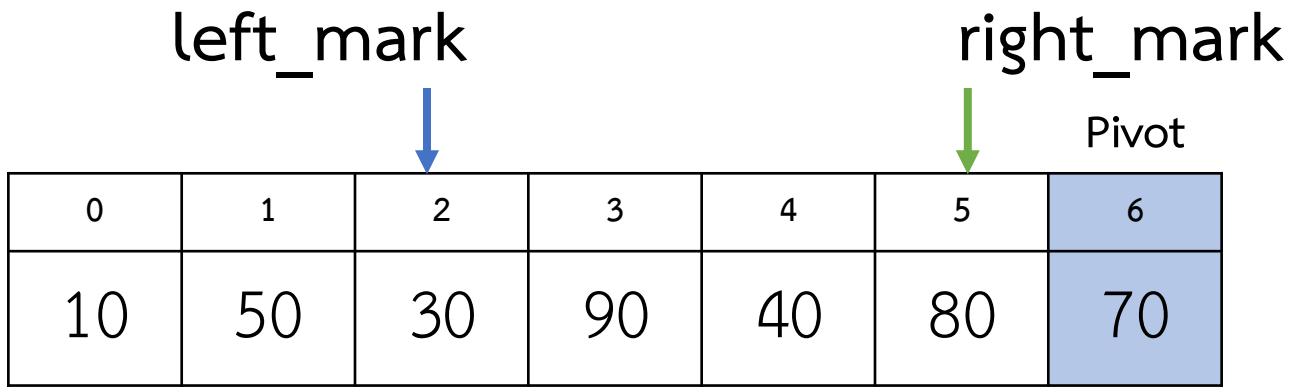
ขั้นตอนวิธี

i. ถ้าค่าในตำแหน่ง left_mark < pivot แล้ว left_mark จะเลื่อนไปทางขวาจนกว่าค่าในตำแหน่ง left_mark > pivot

ii. ถ้าค่าในตำแหน่ง right_mark > pivot แล้ว right_mark ก็จะเลื่อนไปทางซ้ายจนกว่า ค่าในตำแหน่ง right_mark < pivot

iii. สลับค่าในตำแหน่ง left_mark และ right_mark

iv. ทำซ้ำ ข้อ i, ii, iii จนทั้ง left_mark และ right_mark สลับด้านกัน



https://runestone.academy/ns/books/published/pythonds/SortSearch/TheQuickSort.html

Sorting Algorithm: Quick Sort

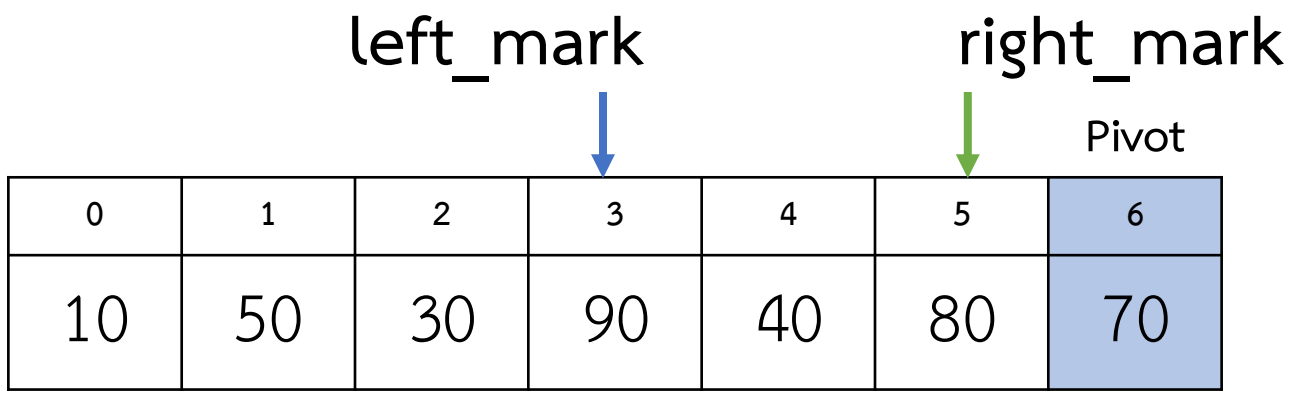
ขั้นตอนวิธี

i. ถ้าค่าในตำแหน่ง left_mark < pivot แล้ว left_mark จะเลื่อนไปทางขวาจนกว่าค่าในตำแหน่ง left_mark > pivot

ii. ถ้าค่าในตำแหน่ง right_mark > pivot แล้ว right_mark ก็จะเลื่อนไปทางซ้ายจนกว่า ค่าในตำแหน่ง right_mark < pivot

iii. สลับค่าในตำแหน่ง left_mark และ right_mark

iv. ทำซ้ำ ข้อ i, ii, iii จนทั้ง left_mark และ right_mark สลับด้านกัน



https://runestone.academy/ns/books/published/pythonds/SortSearch/TheQuickSort.html

Sorting Algorithm: Quick Sort

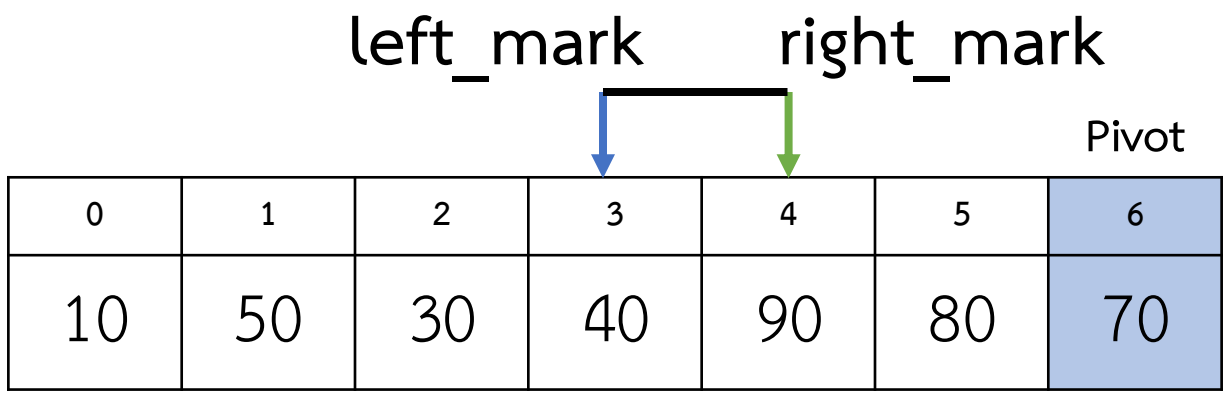
ขั้นตอนวิธี

i. ถ้าค่าในตำแหน่ง left_mark < pivot แล้ว left_mark จะเลื่อนไปทางขวาจนกว่าค่าในตำแหน่ง left_mark > pivot

ii. ถ้าค่าในตำแหน่ง right_mark > pivot แล้ว right_mark ก็จะเลื่อนไปทางซ้ายจนกว่า ค่าในตำแหน่ง right_mark < pivot

iii. สลับค่าในตำแหน่ง left_mark และ right_mark

iv. ทำซ้ำ ข้อ i, ii, iii จนทั้ง left_mark และ right_mark สลับด้านกัน



https://runestone.academy/ns/books/published/pythonds/SortSearch/TheQuickSort.html

Sorting Algorithm: Quick Sort

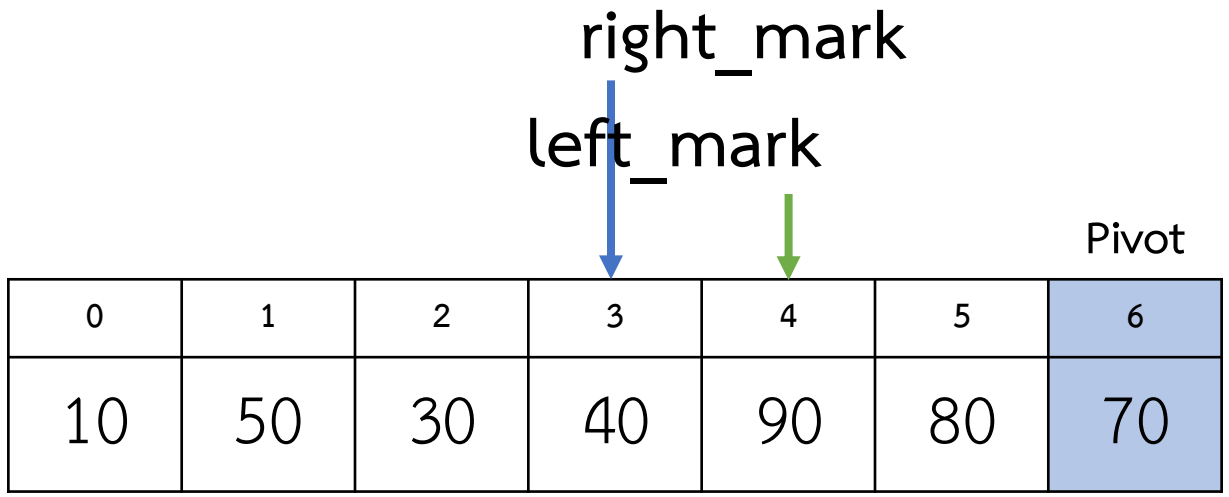
ขั้นตอนวิธี

i. ถ้าค่าในตำแหน่ง left_mark < pivot แล้ว left_mark จะเลื่อนไปทางขวาจนกว่าค่าในตำแหน่ง left_mark > pivot

ii. ถ้าค่าในตำแหน่ง right_mark > pivot แล้ว right_mark ก็จะเลื่อนไปทางซ้ายจนกว่า ค่าในตำแหน่ง right_mark < pivot

iii. สลับค่าในตำแหน่ง left_mark และ right_mark

iv. ทำซ้ำ ข้อ i, ii, iii จนทั้ง left_mark และ right_mark สลับด้านกัน

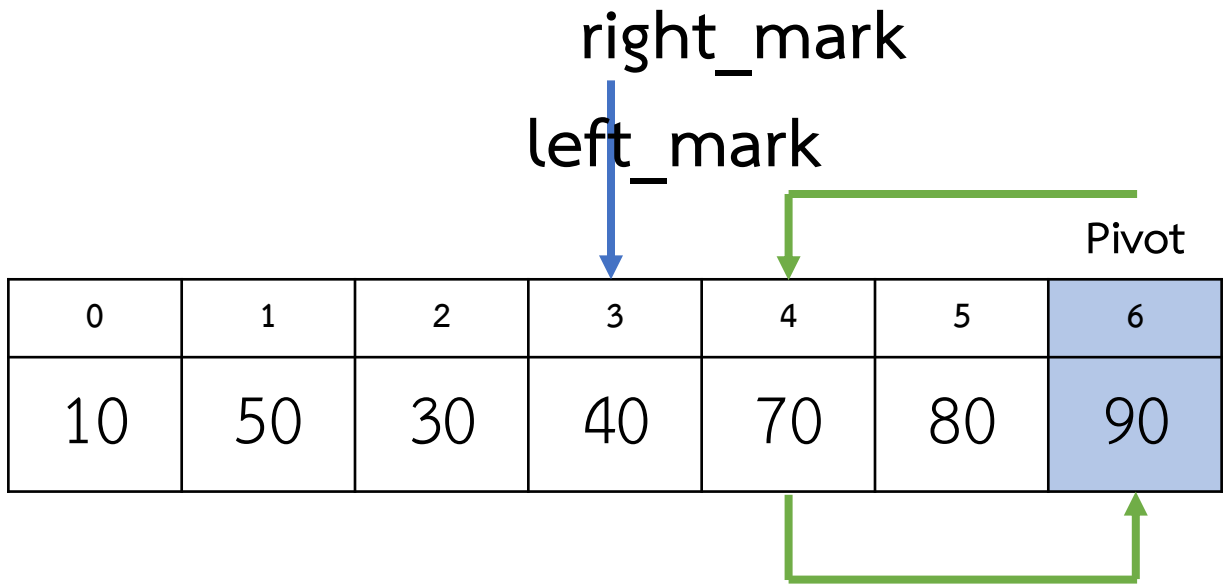


https://runestone.academy/ns/books/published/pythonds/SortSearch/TheQuickSort.html

Sorting Algorithm: Quick Sort

ขั้นตอนวิธี

4. สลับที่ pivot กับ left_mark

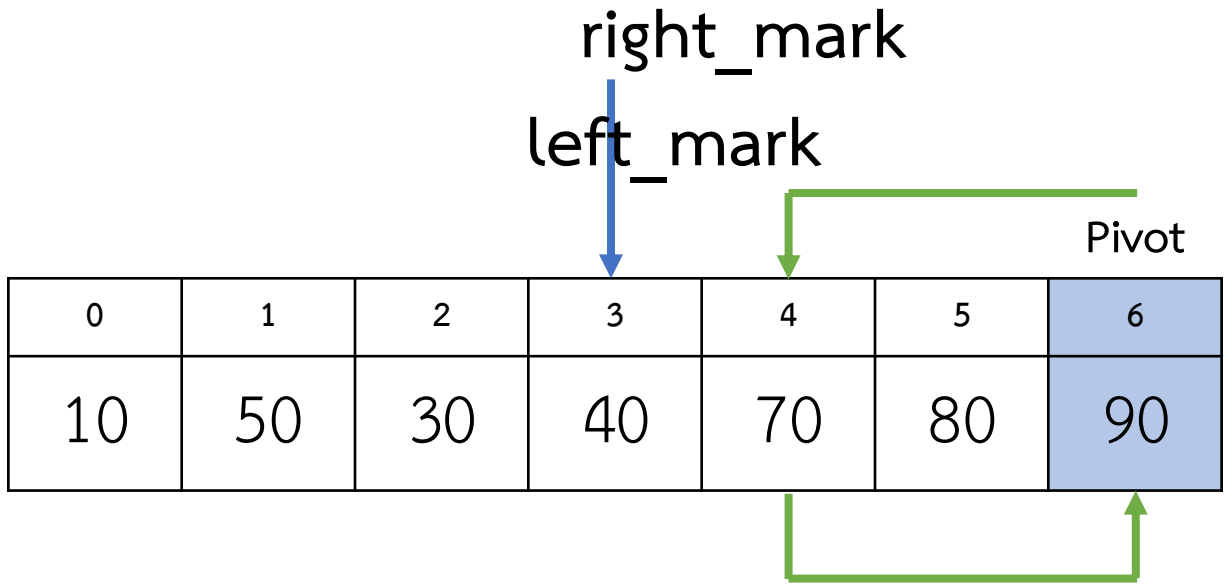


https://runestone.academy/ns/books/published/pythonds/SortSearch/TheQuickSort.html

Sorting Algorithm: Quick Sort

ขั้นตอนวิธี

4. สลับที่ pivot กับ left_mark



https://runestone.academy/ns/books/published/pythonds/SortSearch/TheQuickSort.html

Sorting Algorithm: Quick Sort

- สังเกตว่าฝั่งซ้ายของจุดที่ pivot ย้ายที่มา จะมีค่าน้อยกว่า pivot ทั้งหมด
- และฝั่งขวาของจุดที่ pivot ย้ายที่มา จะมีค่ามากกว่า pivot ทั้งหมด

0	1	2	3	4	5	6
10	50	30	40	70	80	90

https://runestone.academy/ns/books/published/pythonds/SortSearch/TheQuickSort.html

Sorting Algorithm: Quick Sort

ขั้นตอนวิธี

4. สลับที่ pivot กับ left_mark
5. แยกฝั่งซ้าย-ขวา ทำ 1-4 ซ้ำกับฝั่งซ้าย-ขวา จนเหลือ element ที่เล็กที่สุด

0	1	2	3	4	5	6
10	50	30	40	70	80	90

0	1	2	3
10	50	30	40

4
70

5	6
80	90

Sorting Algorithm: Quick Sort

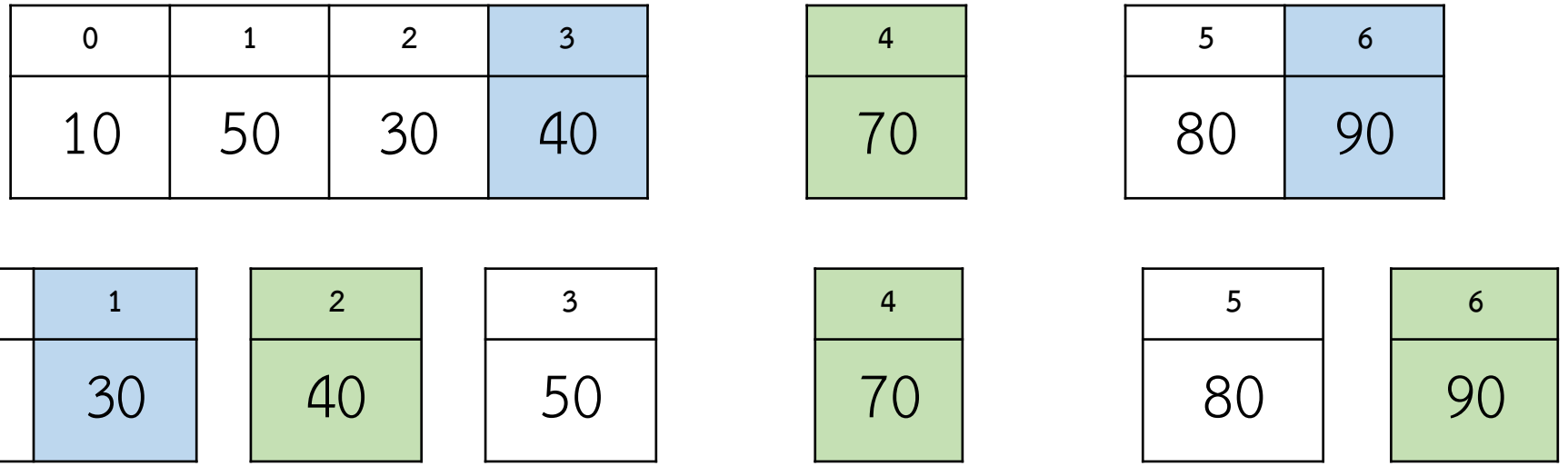
0	1	2	3	4	5	6
10	50	30	40	70	80	90

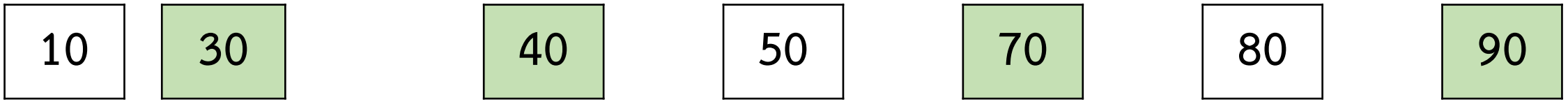
0	1	2	3
10	50	30	40

4
70

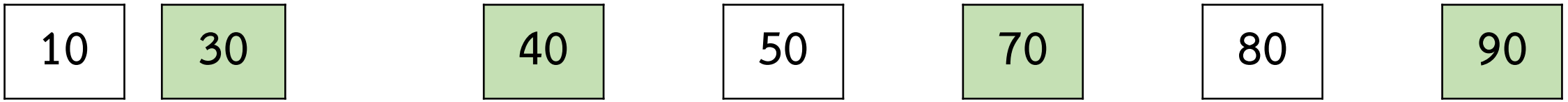
5	6
80	90

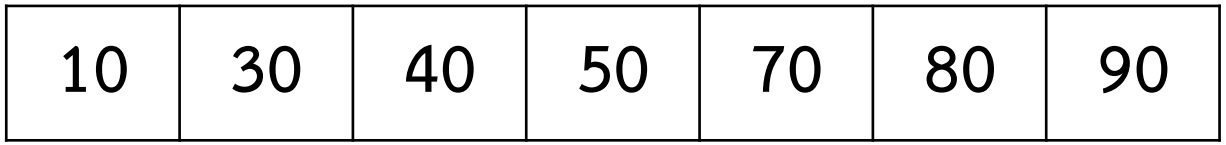
Sorting Algorithm: Quick Sort





Sorting Algorithm: Quick Sort





Sorting Algorithm: Quick Sort

```cpp
#include<iostream>
using namespace std;

void swap(int& a, int& b) {
    int temp = a;
    a = b;
    b = temp;
}

 int partition(int arr[],int low,int high)
 {
   int pivot=arr[high];
   int i=(low-1);
   for(int j=low;j<=high;j++)
   {
     if(arr[j]<pivot)
     {
       i++;
       swap(arr[i],arr[j]);
     }
   }
   swap(arr[i+1],arr[high]);
   return (i+1);
 }
```

```cpp
void quickSort(int arr[],int low,int high)
{
  if(low<high)
  {
    int pi=partition(arr,low,high);

    quickSort(arr,low,pi-1);
    quickSort(arr,pi+1,high);
  }
}
int main() {
  int arr[]={10, 80, 30, 90, 40, 50, 70};
  int n=sizeof(arr)/sizeof(arr[0]);
  quickSort(arr,0,n-1);
  cout<<"Sorted Array\n";
  for(int i=0;i<n;i++)
  {
    cout<<arr[i]<<" ";
  }
  return 0;
}
```

Sorting Algorithm: Quick Sort

วิเคราะห์ความซับซ้อนของ Quick Sort

- หั่นทีละครึ่ง จำนวน n รอบ ดังนั้นความซับซ้อนโดยเฉลี่ยคือ $O(n \log n)$
- เคสที่แย่ที่สุด (เนื่องจากต้องรันจนครบ loop 2 ชั้น) ทำให้ $O(n^2)$

สถานการณ์ที่เลวร้ายที่สุดสำหรับ Quicksort เกิดขึ้นเมื่อ pivot ที่แต่ละขั้นตอนทำให้มีการแบ่งแยก partition ที่ไม่สมดุลมาก ๆ โดยเฉพาะเมื่ออาเรย์มีการเรียงลำดับแล้วและ pivot ถูกเลือกเป็นสมาชิกที่เล็กที่สุดหรือสมาชิกที่มากที่สุดเสมอ

Sorting Algorithm: Quick Sort

วิเคราะห์จุดเด่น-จุดด้อยของ Quick Sort

จุดเด่น

1. มีประสิทธิภาพสูงในข้อมูลขนาดใหญ่
2. ใช้หน่วยความจำไม่มาก
3. สามารถรันแบบคู่ขนานได้ (Parallel) โดยใช้ประโยชน์จากโปรเซสเซอร์หรือ Threads

จุดด้อย

1. เมื่อเกิด worst case จะได้ $O(N^2)$
2. ไม่ใช่ทางเลือกที่ดีสำหรับข้อมูลขนาดเล็ก
3. เมื่อเรียนในขั้นสูงขึ้นไปจะพบว่า Quick Sort เป็นวิธีการที่ไม่เสถียร (Unstable)

Sorting Algorithm: โจทย์ปัญหา

โจทย์ 6 Wave-like array จงเขียนโปรแกรมที่รับค่าจำนวนเต็ม N และอาเรย์ที่เก็บจำนวนเต็มขนาด N ตัวเข้ามา ให้เรียงลำดับอาเรย์ให้อยู่ในรูปแบบ wave-like array (ในที่นี้หมายถึงเรียงลำดับสมาชิกในลำดับที่ arr[1] >= arr[2] <= arr[3] >= arr[4] <= arr[5].....) โดยที่จุดยอดและท้องคลื่นจะสูงขึ้นจากซ้ายไปขวาเสมอ

ข้อมูลนำเข้า : บรรทัดแรก จำนวนเต็ม N

บรรทัดที่ 2 อาเรย์ของจำนวนเต็ม จำนวน N ตัว คั่นด้วยเว้นวรรค การันตีว่าจะไม่มีเลขซ้ำกันในอาเรย์

ข้อมูลส่งออก : อาเรย์ที่จัดเรียงตัวเลขแบบ Wave-like

โจทย์ 6 Wave-like array

ตัวอย่าง

ข้อมูลนำเข้า	ผลลัพธ์
11 9 5 12 65 42 13 28 1 99 4 100	4 1 9 5 13 12 42 28 99 65 100
5 1 4 3 5 2	2 1 4 3 5
10 9 8 1 2 5 6 7 4 3 0	1 0 3 2 5 4 7 6 9 8

Sorting Algorithm

วิธีการจัดเรียงที่จะกล่าวถึง จะเข้าถึงข้อมูลในอาเรย์ทุกตำแหน่งที่จะทำการจัดเรียง ซึ่งประกอบด้วยสมาชิก n ตัว

วิธีการจัดเรียงมีหลายวิธี โดยเราจะกล่าวถึงเป็นกลุ่ม ๆ ได้แก่

- Selection Sort, Bubble Sort, Insertion Sort
- Merge Sort, Quick Sort
- Heap Sort

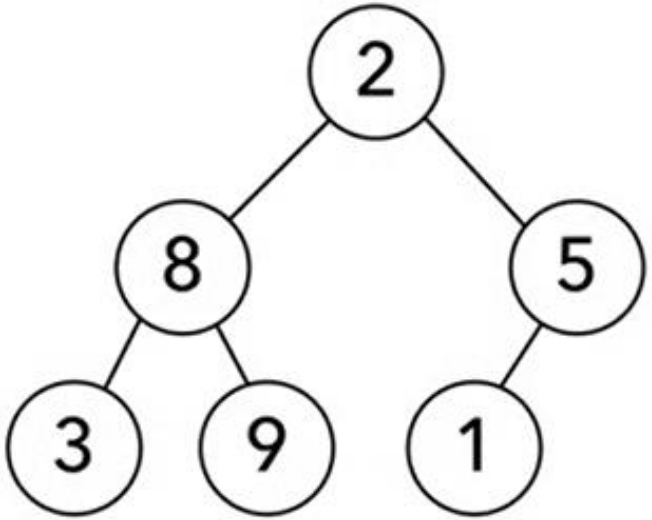
Sorting Algorithm: Heap Sort

Heap Sort อาศัยหลักการของ Tree ในการจัดเรียง (เราจะได้เรียนเรื่อง Tree ในภายหลัง) โดย Tree ที่จะถูกนำมาใช้งานจะเป็นประเภท Ordered Binary Tree

ในหัวข้อนี้จะไม่พูดถึงเรื่องเทคนิคละเอียดมากนัก เพราะยังไม่รู้จัก tree อย่างละเอียด แต่เราจะมาดูแนวคิดของ Heap Sort กัน

Sorting Algorithm: Heap Sort

สิ่งเดียวที่ขอให้รู้ตอนนี้คือ สิ่งนี้คือ tree ที่เรากำลังพูดถึง



Sorting Algorithm: Heap Sort

หากต้องการเรียงข้อมูลต่อไปนี้จากน้อยไปมาก

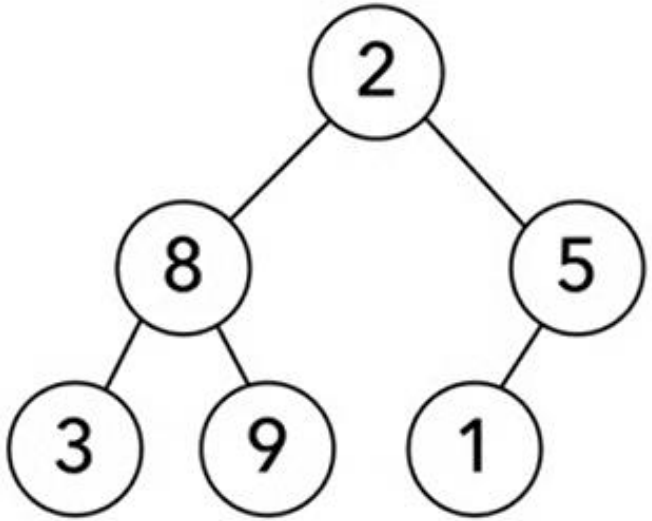
2	8	5	3	9	1

Sorting Algorithm: Heap Sort

ขั้นตอนวิธี

1. นำอาเรย์นี้ไปสร้างเป็นโครงสร้าง tree

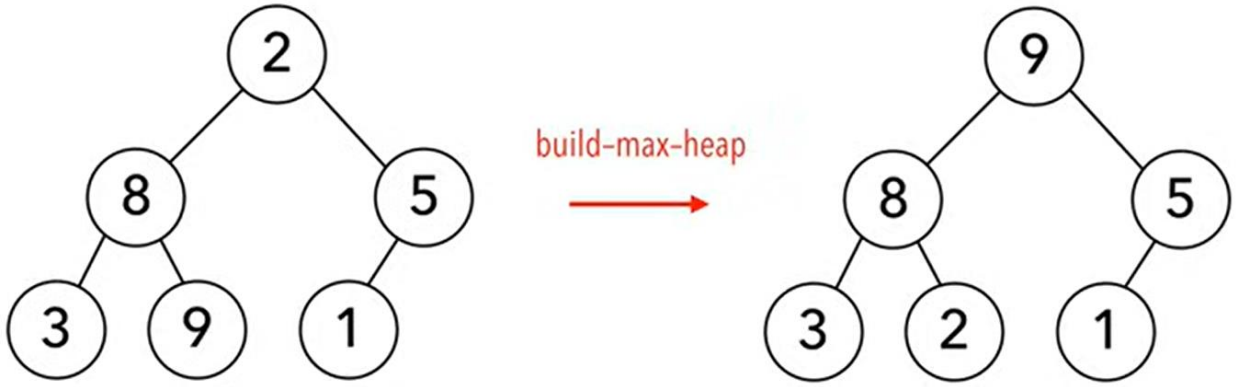
2	8	5	3	9	1



Sorting Algorithm: Heap Sort

ขั้นตอนวิธี

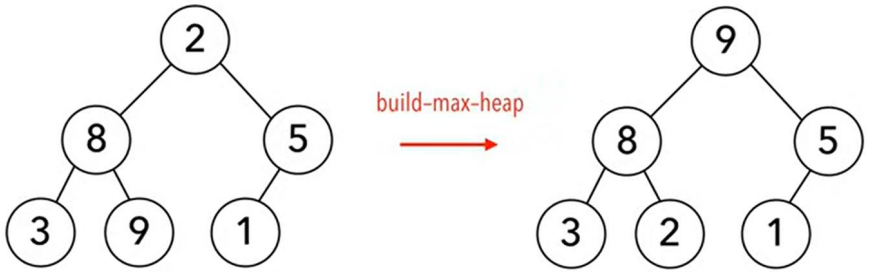
1. นำอาเรย์นี้ไปสร้างเป็นโครงสร้าง tree
2. สร้าง max-heap (รอบต่อ ๆ ไปจะใช้การ heapify)



9	8	5	2	3	1

Sorting Algorithm: Heap Sort

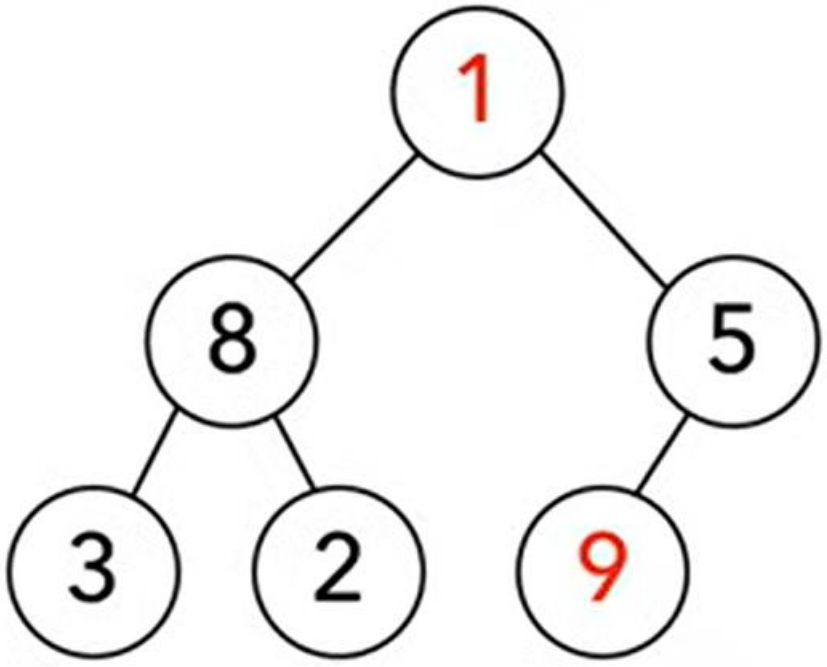
แล้วเราจะสร้าง max-heap ได้อย่างไร



Sorting Algorithm: Heap Sort

ขั้นตอนวิธี

1. นำอาเรย์นี้ไปสร้างเป็นโครงสร้าง tree
2. สร้าง max-heap
3. สลับข้อมูลบนสุด กับข้อมูลล่างขวาสุด

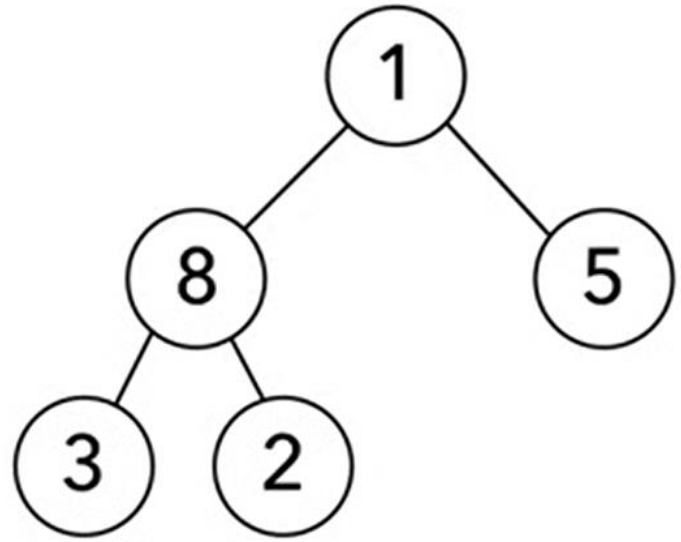


1	8	5	2	3	9

Sorting Algorithm: Heap Sort

ขั้นตอนวิธี

1. นำอาเรย์นี้ไปสร้างเป็นโครงสร้าง tree
2. สร้าง max-heap
3. สลับข้อมูลบนสุด กับข้อมูลล่างขวาสุด
4. ลบข้อมูลล่างขวาสุดออกจากทรี (ถือว่าเรียงไปแล้ว)

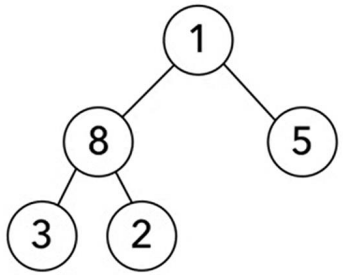


1	8	5	2	3	9

Sorting Algorithm: Heap Sort

แล้ว heapify คืออะไร?

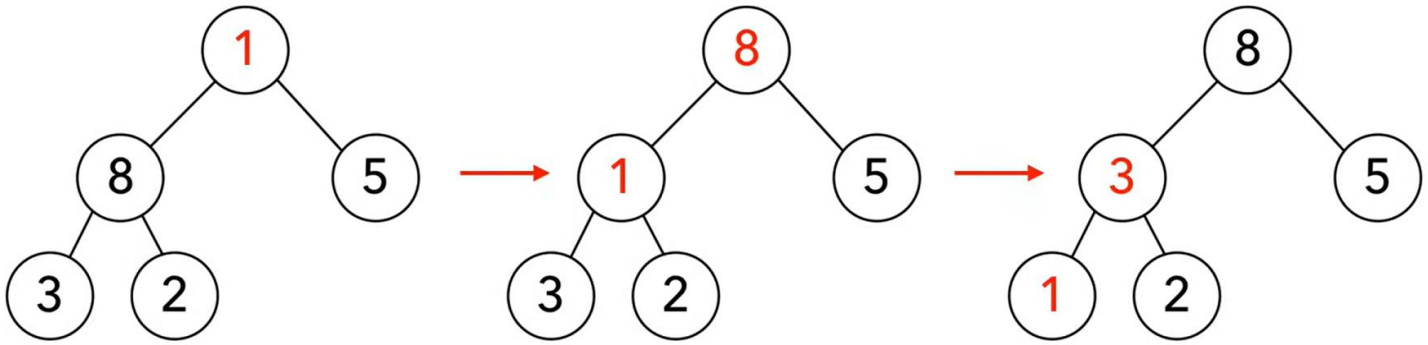
- หลัก ๆ แล้วมีไอเดียเหมือนการสร้าง max-heap แต่เราสมมติว่ามีโครงสร้างส่วนใหญ่ของ tree ที่มีการจัดเรียงไว้ก่อนหน้าแล้ว
- ตามหลักการเราจัดเรียงทั้ง tree ไว้ตั้งแต่ตอนแรก เพียงแต่มีการสลับเอาตัวท้ายสุดมาไว้เป็นตัวบนสุด ดังนั้น เราต้องแก้ไขลำดับที่ผิดที่ผิดทางเพียงตัวเดียวนี้



Sorting Algorithm: Heap Sort

แทนที่การ heapify จะไล่ดูตั้งแต่ tree ย่อยล่างสุดไล่ขึ้นมา เราก็จะดูจากบนลงล่างไปเลย ว่าตัวเลขบนสุดควรจะสลับกับตัวข้างล่างทางซ้าย หรือขวา หรือควรวางไว้ตำแหน่งเดิม โดยมีหลักการง่าย ๆ ว่า

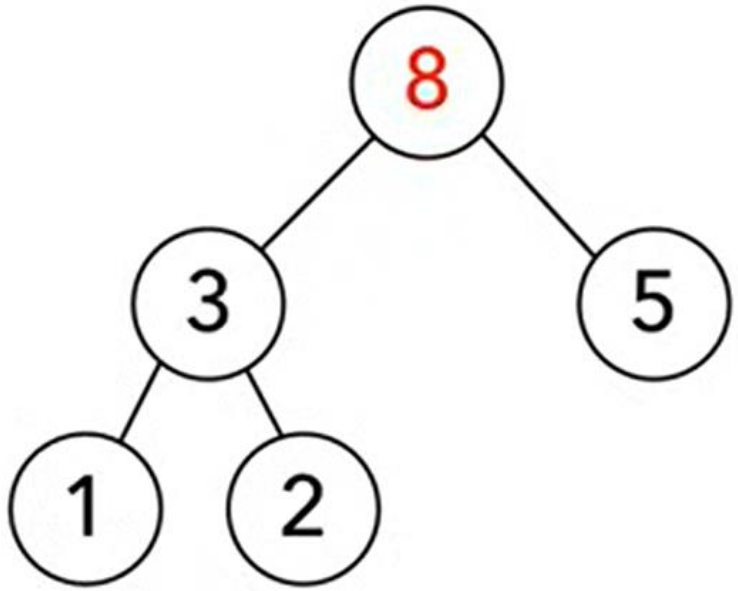
- ถ้าตัวบนสุดน้อยกว่าตัวข้างล่างตัวใดตัวหนึ่ง จะสลับกับตัวข้างล่างฝั่งที่มีค่ามากกว่า
- ถ้าตัวบนสุดมีค่ามากกว่าตัวข้างล่างทั้งสองตัวอยู่แล้ว จะไม่สลับตำแหน่ง



Sorting Algorithm: Heap Sort

ขั้นตอนวิธี

1. นำอาเรย์นี้ไปสร้างเป็นโครงสร้าง tree
2. สร้าง max-heap (รอบถัดไปทำ heapify)
3. สลับข้อมูลบนสุด กับข้อมูลล่างขวาสุด
4. ลบข้อมูลล่างขวาสุดออกจากทรี (ถือว่าเรียงไปแล้ว)
5. ทำ 2-4 ซ้ำจน tree ไม่มีสมาชิกเหลืออยู่

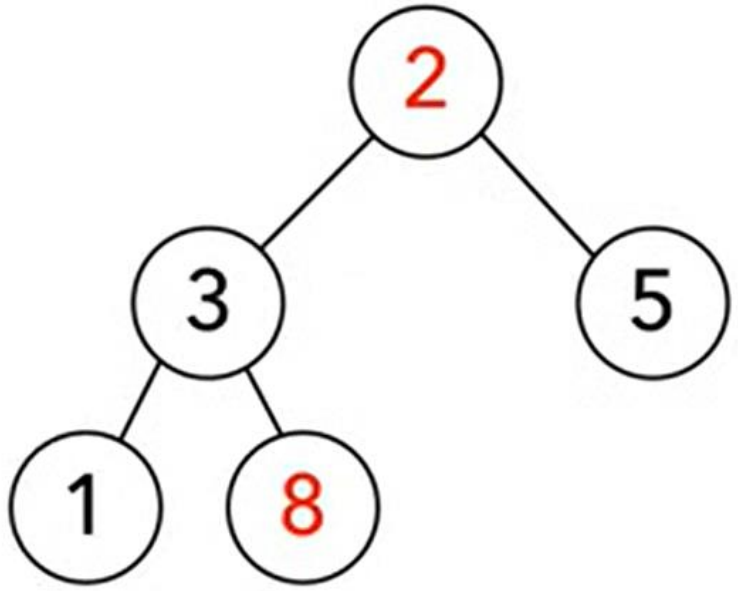


8	3	5	1	2	9

Sorting Algorithm: Heap Sort

ขั้นตอนวิธี

1. นำอาเรย์นี้ไปสร้างเป็นโครงสร้าง tree
2. สร้าง max-heap
3. สลับข้อมูลบนสุด กับข้อมูลล่างขวาสุด
4. ลบข้อมูลล่างขวาสุดออกจากทรี (ถือว่าเรียงไปแล้ว)
5. ทำ 2-4 ซ้ำจน tree ไม่มีสมาชิกเหลืออยู่ (ข้อ 2 จะใช้การ heapify แทน)

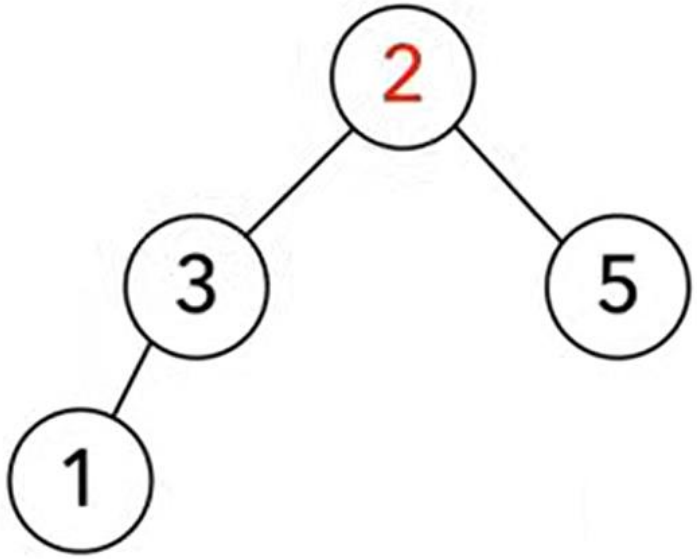


2	3	5	1	8	9

Sorting Algorithm: Heap Sort

ขั้นตอนวิธี

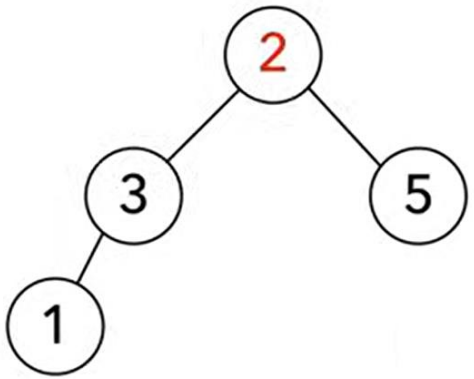
1. นำอาเรย์นี้ไปสร้างเป็นโครงสร้าง tree
2. สร้าง max-heap
3. สลับข้อมูลบนสุด กับข้อมูลล่างขวาสุด
4. ลบข้อมูลล่างขวาสุดออกจากทรี (ถือว่าเรียงไปแล้ว)
5. ทำ 2-4 ซ้ำจน tree ไม่มีสมาชิกเหลืออยู่ (ข้อ 2 จะใช้การ heapify แทน)



2	3	5	1	8	9

Sorting Algorithm: Heap Sort

หลังจากนี้ เป็นยังไงต่อ?



Sorting Algorithm: Heap Sort

```cpp
#include <iostream>
using namespace std;

void swap(int& a, int& b) {
    int temp = a;
    a = b;
    b = temp;
}

void heapify(int arr[], int N, int i)
{
    int largest = i;

    int l = 2 * i + 1;

    int r = 2 * i + 2;

    if (l < N && arr[l] > arr[largest])
        largest = l;
    if (r < N && arr[r] > arr[largest])
        largest = r;
    if (largest != i) {
        swap(arr[i], arr[largest]);
        heapify(arr, N, largest);
    }
}

void heapSort(int arr[], int N)
{
    for (int i = N / 2 - 1; i >= 0; i--)
        heapify(arr, N, i);

    for (int i = N - 1; i > 0; i--) {

        swap(arr[0], arr[i]);

        heapify(arr, i, 0);
    }
}

void printArray(int arr[], int N)
{
    for (int i = 0; i < N; ++i)
        cout << arr[i] << " ";
    cout << "\n";
}

int main()
{
    int arr[] = { 2, 8, 5, 3, 9, 1 };
    int N = sizeof(arr) / sizeof(arr[0]);

    heapSort(arr, N);

    cout << "Sorted array is \n";
    printArray(arr, N);
}
```

Sorting Algorithm: Heap Sort

วิเคราะห์ความซับซ้อนของ Heap Sort

- ความซับซ้อนโดยเฉลี่ยคือ $O(n \log n)$

Sorting Algorithm: Heap Sort

วิเคราะห์จุดเด่น-จุดด้อยของ Heap Sort

จุดเด่น

1. มีความซับซ้อนเป็น $O(N \log N)$ เสมอ ปัจจัย log n มาจากความสูงของ binary heap และทำให้อัลกอริทึมรักษาประสิทธิภาพอย่างดีแม้ในกรณีของข้อมูลที่มีจำนวนมากมาย
2. ความจำที่ใช้ใน Heap Sort สามารถลดลงได้เนื่องจากมีความจำที่จำเป็นสำหรับการเก็บรายการเริ่มต้นของรายการที่ต้องการจัดเรียงลำดับเท่านั้น ไม่ต้องการพื้นที่เพิ่มเติมในการทำงาน

จุดด้อย

1. มี cost ในการรันสูง
2. เป็นการจัดเรียงประเภทที่ไม่เสถียร
3. ไม่ได้มีประสิทธิภาพสูงมากนักเมื่อทำงานกับข้อมูลที่มีความซับซ้อนมาก

Outline

- บทนำเกี่ยวกับการจัดเรียงลำดับ
- Sorting Algorithm
- การเปรียบเทียบประสิทธิภาพของ Sorting Algorithm
- การใช้งานและการเลือก Sorting Algorithm
- Standard Template Library (STL)
- การประยุกต์ใช้ Sorting Algorithm

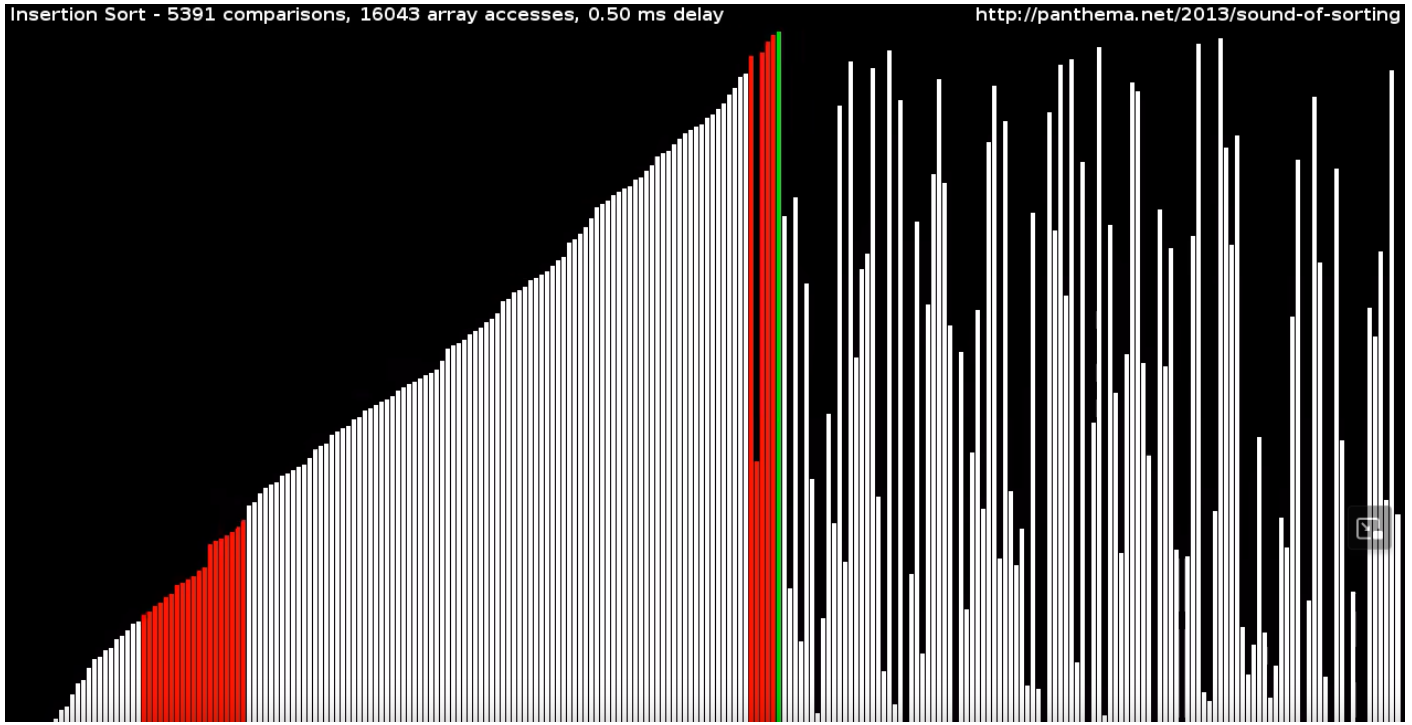
การเปรียบเทียบประสิทธิภาพ

Sorting Algorithms	Time Complexity			Space Complexity
	Best Case	Average Case	Worst Case	Worst Case
Bubble Sort	Ω(N)	Θ(N^2)	O(N^2)	O(1)
Selection Sort	Ω(N^2)	Θ(N^2)	O(N^2)	O(1)
Insertion Sort	Ω(N)	Θ(N^2)	O(N^2)	O(1)
Quick Sort	Ω(N log N)	Θ(N log N)	O(N^2)	O(N)
Merge Sort	Ω(N log N)	Θ(N log N)	O(N log N)	O(N)
Heap Sort	Ω(N log N)	Θ(N log N)	O(N log N)	O(1)

การเปรียบเทียบประสิทธิภาพ

อยากดูความแตกต่างของความเร็วในการ Sorting ให้เป็นรูปธรรม + Sorting ประเภทอื่น ๆ คลิก

https://www.youtube.com/watch?v=kPRA0W1kECg



Outline

- บทนำเกี่ยวกับการจัดเรียงลำดับ
- Sorting Algorithm
- การเปรียบเทียบประสิทธิภาพของ Sorting Algorithm
- การใช้งานและการเลือก Sorting Algorithm
- Standard Template Library (STL)
- การประยุกต์ใช้ Sorting Algorithm

การใช้งานและการเลือก Sorting Algorithm

การเลือกอัลกอริทึมการเรียงลำดับ (Sorting Algorithm) นั้นขึ้นอยู่กับสถานการณ์และความต้องการของแต่ละปัญหา โดยมีปัจจัยหลายอย่างที่ควรพิจารณาเมื่อต้องการเลือกใช้ Sorting Algorithm

ก่อนหน้านี้ เราได้พูดถึงข้อดี-ข้อเสีย รวมถึงความซับซ้อนของอัลกอริทึมไปแล้ว ในหัวข้อนี้ แทนที่เราจะโฟกัสไปที่แต่ละวิธีการ แต่เราจะลองมองย้อนกลับ โดยการพิจารณาวิธีการที่จะใช้จากปัญหาที่เราได้รับมา

การใช้งานและการเลือก Sorting Algorithm

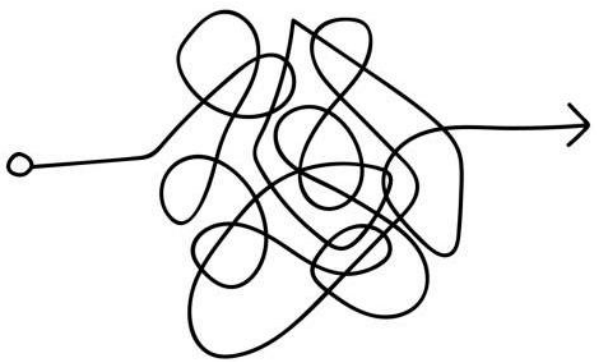
1. ขนาดของข้อมูล

หากข้อมูลมีขนาดใหญ่มาก โดยเฉพาะในกรณีของข้อมูลขนาดเยอะมาก เราอาจต้องพิจารณาใช้อัลกอริทึมที่มีประสิทธิภาพสูงบนข้อมูลขนาดใหญ่ เช่น Merge Sort หรือ Quick Sort ซึ่งมีเวลาทำงานในลักษณะเฉลี่ยน้อยกว่า Bubble Sort หรือ Insertion Sort

การใช้งานและการเลือก Sorting Algorithm

2. ความซับซ้อนของอัลกอริทึม

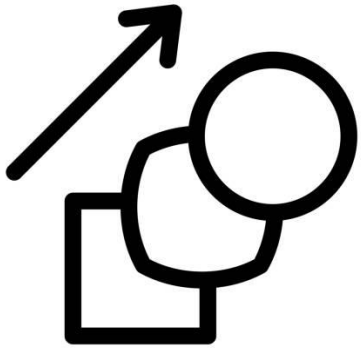
บางครั้งอาจต้องพิจารณาความซับซ้อนของอัลกอริทึมในการเลือกใช้ เช่น ถ้าเราต้องการอัลกอริทึมที่เรียงลำดับข้อมูลได้อย่างรวดเร็วและมีประสิทธิภาพ แต่ไม่ต้องการความซับซ้อนมาก เราสามารถใช้ Insertion Sort หรือ Selection Sort ที่มีความเรียบง่ายในการทำงานได้

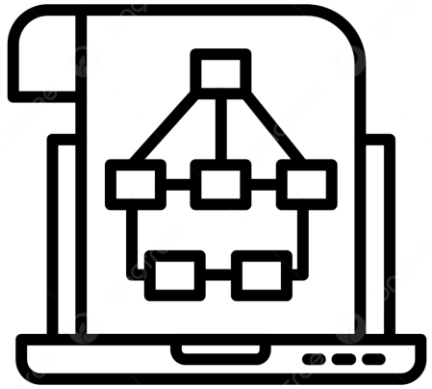


การใช้งานและการเลือก Sorting Algorithm

3. ลักษณะของข้อมูล

อัลกอริทึมที่มีประสิทธิภาพสูงบนข้อมูลที่ไม่มีลำดับหรือข้อมูลที่มีการกำหนดค่าแบบสุ่มอาจไม่ได้ผลดีเมื่อใช้กับข้อมูลที่มีลำดับมาก่อนหน้า ในกรณีนี้เราอาจต้องพิจารณาใช้ Insertion Sort หรือ Bubble Sort ซึ่งมีประสิทธิภาพดีกว่าในกรณีนี้





การใช้งานและการเลือก Sorting Algorithm

4. ความจำเป็นในการจัดเรียงข้อมูล

โจทย์บางข้อโกหกเราโดยทำเหมือนว่าต้องจัดเรียงข้อมูลก่อน ทั้งที่จริง ๆ แล้วเป็นเพียงโจทย์คำนวณ ถ้ารู้สึกว่าทำการจัดเรียงแล้วไปไม่ถึงฝั่งฝันเสียที ให้ลองพิจารณาโจทย์อีกครั้งว่าจำเป็นต้องใช้การจัดเรียงหรือไม่

การใช้งานและการเลือก Sorting Algorithm

5. ความต้องการในการใช้หน่วยความจำ

บางอัลกอริทึมอาจมีความต้องการในการใช้หน่วยความจำมากกว่าอัลกอริทึมอื่น ๆ เมื่อจัดเรียงข้อมูลขนาดใหญ่ ซึ่งอาจต้องพิจารณาหากเรามีข้อจำกัดในการใช้หน่วยความจำ (ในระดับเราอาจไม่ต้องกังวลกับจุดนี้มากนัก)

Outline

- บทนำเกี่ยวกับการจัดเรียงลำดับ
- Sorting Algorithm
- การเปรียบเทียบประสิทธิภาพของ Sorting Algorithm
- การใช้งานและการเลือก Sorting Algorithm
- Standard Template Library (STL)
- การประยุกต์ใช้ Sorting Algorithm

Standard Template Library (STL)

- **Sorting** เป็นหนึ่งในฟังก์ชันพื้นฐานที่ใช้กับข้อมูล หมายถึงการจัดเรียงข้อมูลให้เรียงลำดับในลักษณะที่เฉพาะเจาะจง เช่น เรียงลำดับให้อยู่ในลำดับเพิ่มขึ้นหรือลำดับลดลง
- ใน C++ STL มีฟังก์ชันชื่อว่า sort() ซึ่งเป็นฟังก์ชันที่มีอยู่แล้วภายใน STL
- ฟังก์ชันนี้ใช้ IntroSort ภายใน โดยการนำ QuickSort, HeapSort และ InsertionSort มาผสมกันเข้าไป

Standard Template Library (STL)

- โดยปกติแล้ว จะเลือกใช้ Quick Sort
- แต่หาก Quick Sort ทำการแบ่ง partition อย่างไม่เป็นธรรมและใช้เวลามากกว่า $O(NlogN)$ คอมไพเลอร์จะเปลี่ยนไปใช้ Heap Sort และเมื่อขนาดของอาร์เรย์เล็กมากพอ คอมไพเลอร์จะเปลี่ยนไปใช้ Insertion Sort แทน

Standard Template Library (STL)


```
sort(startaddress, endaddress)

startaddress: the address of the first
              element of the array
endaddress: the address of the next
            contiguous location of the
            last element of the array.
So actually sort() sorts in the
range of [startaddress,endaddress)
```

Standard Template Library (STL)

```cpp
#include <algorithm>
#include <iostream>
using namespace std;

int main()
{
    int arr[] = {3, 5, 1, 2, 4};
    sort(begin(arr), end(arr));

    for (int i : arr)
    {
        cout << i << " ";
    }
    return 0;
}
```

Outline

- บทนำเกี่ยวกับการจัดเรียงลำดับ
- Sorting Algorithm
- การเปรียบเทียบประสิทธิภาพของ Sorting Algorithm
- การใช้งานและการเลือก Sorting Algorithm
- Standard Template Library (STL)
- การประยุกต์ใช้ Sorting Algorithm

การประยุกต์ใช้ Sorting Algorithm

- ตอนนี้เราได้เรียนรู้วิธีการ sort ทั้งแบบทำเอง และแบบ STL แล้ว ทำให้โจทย์ปัญหาการจัดเรียงแบบพื้นฐานสามารถแก้ได้ง่ายขึ้นอย่างมาก
- ต่อจากนี้เราจะลองทดสอบประกอบความรู้ที่ได้เรียนมาทั้งหมดเข้าด้วยกัน

การประยุกต์ใช้ Sorting Algorithm

โจทย์ 7 เรียงวันในปฏิทิน

จงเขียนโปรแกรมรับค่า วัน เดือน ปี มาเป็นจำนวน n ชุด จากนั้นจัดเรียงวันจากน้อยไปมาก และแสดงผลการเรียงลำดับออกทางจอภาพ

ข้อมูลนำเข้า บรรทัดแรก ค่า n แทนจำนวนชุดข้อมูล วัน เดือน ปี ที่จะจัดเรียง

บรรทัดที่ 2 ถึง n+1 แทนข้อมูล วัน เดือน ปี ที่คั่นระหว่างข้อมูลด้วยเว้นวรรค

ข้อมูลส่งออก บรรทัดที่ 1 ถึง n แต่ละบรรทัดแทนข้อมูลวัน เดือน ปี ที่จัดเรียงแล้วแต่ละวัน คั่นระหว่างข้อมูลวัน เดือน ปี ด้วยเว้นวรรค

โจทย์ 7 เรียงวันในปฏิทิน

ตัวอย่าง

ข้อมูลนำเข้า	ผลลัพธ์
5 5 1 2545 6 2 1579 12 1 2410 1 12 2555 30 3 2565	6 2 1579 12 1 2410 5 1 2545 1 12 2555 30 3 2565

การประยุกต์ใช้ Sorting Algorithm

โจทย์ 8 เรียงสตริง

จงเขียนโปรแกรมรับค่าสตริงมาเป็นจำนวน n ชุด จากนั้นจัดเรียงสตริงอ้างอิงตามตาราง ASCII และแสดงผลการเรียงลำดับออกทางจอภาพ

ข้อมูลนำเข้า บรรทัดแรก ค่า n แทนจำนวนข้อมูลสายอักขระที่จะจัดเรียง (ไม่ใช่จำนวนอักษร)

บรรทัดที่ 2 ถึง n+1 แทนข้อมูลสายอักขระ การันตีว่าไม่มีเว้นวรรค

ข้อมูลส่งออก บรรทัดที่ 1 ถึง n แต่ละบรรทัดแทนข้อมูลสายอักขระ ที่จัดเรียงแล้วตามหลักพจนานุกรม 1 บรรทัดต่อ 1 สตริง

โจทย์ 8 เรียงสตริง

ตัวอย่าง

ข้อมูลนำเข้า	ผลลัพธ์
3 ILOVEYOU SILPAKORN NICEONE	ILOVEYOU NICEONE SILPAKORN

การประยุกต์ใช้ Sorting Algorithm

โจทย์ 9 เรียงเลขใหญ่โต

ให้แสดงตัวเลขในอาร์เรย์ขนาด n ของจำนวนเต็มบวกที่มีทุกจำนวนเต็มสามารถมีจำนวนหลักได้สูงสุดถึง 10^6 ในลำดับเรียงจากน้อยไปหามาก

ข้อมูลนำเข้า บรรทัดแรก ค่า n แทนจำนวนข้อมูลที่จะจัดเรียง

บรรทัดที่ 2 แทนข้อมูลตัวเลขขนาดใหญ่ที่จะจัดเรียง คั่นด้วยเว้นวรรค

ข้อมูลส่งออก บรรทัดเดียว แสดงตัวเลขที่จัดเรียงแล้ว

โจทย์ 9 เรียงเลขใหญ่โต

ตัวอย่าง

ข้อมูลนำเข้า	ผลลัพธ์
5 3 4 5 9 1	1 3 4 5 9

การประยุกต์ใช้ Sorting Algorithm

โจทย์ 10 เรียง B แต่ปริ๊น A

ให้เขียนโปรแกรมภาษา C++ เพื่อรับข้อมูลอาเรย์ A และ B ที่เก็บจำนวนเต็ม ขนาด N ตัวและจัดเรียงอาเรย์ A ให้เรียงตามลำดับน้อยไปมากของอาเรย์ B และแสดงผลลัพธ์อาเรย์ A หลังจากการจัดเรียง (เรียง B จากน้อยไปมาก แล้วเอาลำดับที่ได้มาเรียง A ดังนั้น A อาจไม่ได้เรียงจากน้อยไปมากก็ได้ เพราะการจัดเรียงขึ้นอยู่กับ B) โดยถ้าใน B มีตัวเลขที่เท่ากัน ให้เรียง A ตามการมาก่อนหลัง

ข้อมูลนำเข้า	บรรทัดแรก ค่า n แทนจำนวนข้อมูลของอาเรย์ A และ B
	บรรทัดที่ 2 แทนข้อมูลจำนวนเต็มของ A ที่จะให้แสดงผล
	บรรทัดที่ 3 แทนข้อมูลจำนวนเต็มของ B ที่ใช้ในการเรียงลำดับ
ข้อมูลส่งออก	บรรทัดเดียว แสดงตัวเลขในอาเรย์ A ที่จัดเรียงตามลำดับของ B แล้ว

Sorting Algorithm

โจทย์ 10 เรียง B แต่ปริ๊น A

ตัวอย่าง

ข้อมูลนำเข้า	ผลลัพธ์
3 4 7 2 3 1 2	7 2 4
5 99 12 65 41 88 3 2 1 4 5	65 12 99 41 88
7 12 65 43 47 86 21 81 23 47 94 10 32 94 65	47 12 86 65 81 43 21

การประยุกต์ใช้ Sorting Algorithm

โจทย์ 11 เรียงคะแนน

ให้สร้างโปรแกรมที่ใช้งานอัลกอริทึมการเรียงลำดับเพื่อจัดเรียงข้อมูลของนักเรียนในชั้นเรียน โดยมีข้อมูลของนักเรียนที่ประกอบด้วยชื่อ, นามสกุล, และคะแนนสอบ โปรแกรมจะต้องสามารถเรียงลำดับนักเรียนตามคะแนนสอบจากสูงไปต่ำ และในกรณีที่นักเรียนมีคะแนนเท่ากัน ให้เรียงตามอักษร (lexicographically) ของชื่อและนามสกุล โดยโปรแกรมจะแสดงผลลัพธ์เป็นลำดับการเรียงของนักเรียนทั้งหมด

ข้อมูลนำเข้า	บรรทัดแรก ค่า n แทนจำนวนนักเรียน
	บรรทัดที่ 2 ถึง n+1 แทนข้อมูลนักเรียนแต่ละคน ได้แก่ ชื่อ นามสกุล และคะแนน
ข้อมูลส่งออก	มี n บรรทัด แสดงผล ชื่อ นามสกุล และคะแนน 1 ชื่อต่อ 1 บรรทัด เรียงตามลำดับคะแนนจาก**มากไปน้อย** หากคะแนนซ้ำกันจะเรียงลำดับตามชื่อ-สกุล

Sorting Algorithm

โจทย์ 11 เรียงคะแนน

ตัวอย่าง

ข้อมูลนำเข้า	ผลลัพธ์
5 John Doe 85 Alice Smith 90 Bob Johnson 80 Mary Brown 90 David Lee 85	Alice Smith 90 Mary Brown 90 John Doe 85 David Lee 85 Bob Johnson 80

Outline

- บทนำเกี่ยวกับการจัดเรียงลำดับ
- Sorting Algorithm
- การเปรียบเทียบประสิทธิภาพของ Sorting Algorithm
- การใช้งานและการเลือก Sorting Algorithm
- Standard Template Library (STL)
- การประยุกต์ใช้ Sorting Algorithm