Practice Exercise M.2

25 คุลาคม 2568

รู้หมือไร่... (Did You Know That...?)

วันนี้เราจะมาทำความรู้จักกับ**เบื้องหลัง**ของการทำงานภายในฟังก์ชัน **`nextPermutation()`** กัน โดยปกติแล้วการเรียงลำดับใน `nextPermutation()` จะอิงตามรูปแบบ **Lexicographical Order** หรือที่เรียกว่า “ลำดับตามพจนานุกรม” หมายความว่าตัวเลขจะถูกมองเป็นเหมือนตัวอักษรในพจนานุกรม เช่น **1** มาก่อน **2**, **2** มาก่อน **3** และเป็นเช่นนี้ไปเรื่อย ๆ

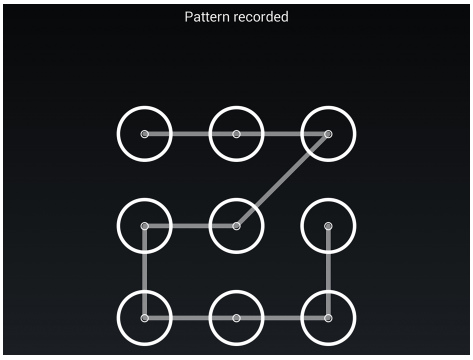
โดยหลักการทำงานของ `nextPermutation()` มีดังนี้ โดยสมมติให้เราต้องการเรียงสับเปลี่ยนตัวเลขตั้งแต่ 1 ถึง n (โดยให้อยู่ในพจน์ของ $\{a_1, a_2, \ldots, a_n\}$)

1. เริ่มจากเราต้องเรียงตัวเลขจาก**น้อยไปมาก** *(ซึ่งก็คือรูปแบบวิธีเรียงสับเปลี่ยนแบบ**แรก**)*
2. หา Index i ที่มากที่สุด ที่ $a_i < a_{i+1}$
3. หา Index j ที่มากที่สุดที่ $a_i < a_j$
4. สลับ a_i และ a_j
5. กลับลำดับ (Reverse) ของค่าตั้งแต่ a_{i+1} ถึง a_n *(ถึงขั้นตอนตรงนี้เราจะได้รูปแบบการเรียงสับเปลี่ยนแบบใหม่แล้ว จะเอาลำดับที่ได้ใหม่ไปทำอะไร เช่น พิมพ์, เทียบกับคำตอบ หรืออะไรก็ตามใจ)*
6. ทำซ้ำข้อที่ 2 - 5 หากว่ารูปแบบที่ได้ใหม่นั้นยังมีเลขที่ติดกันสองตัวที่เรียงแบบน้อยไปมาก *(หมายความว่ายังมี a_p ใด ๆ ที่ $a_p < a_{p+1}$)*

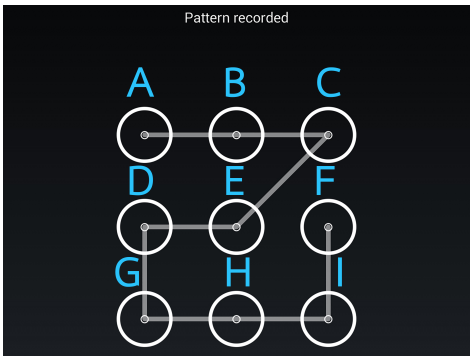
แอนดรอยด์ (Andriod)

โทรศัพท์ Andriod จะมีวิธีการปลดล็อกด้วยรหัสแบบหนึ่ง คือ Pattern Password ซึ่งเป็นเทคโนโลยีที่ล้ำมาก ๆ เพราะแม้แต่ iOS ก็คาดคิดไม่ถึง

นักศึกษาเป็นผู้ใช้โทรศัพท์ Android และต้องการที่จะมีรหัสที่ปลอดภัยที่สุด ดังนั้นนักศึกษาเลยใส่รหัสโดยมีจำนวนจุด Pattern มากที่สุดที่เป็นไปได้ ดังตัวอย่างรูปด้านล่างนี้



หากเราให้แต่ละจุดแทนด้วยเครื่องหมาย



เราจะได้ว่า นักศึกษามีลำดับของรูปแบบนี้คือ **ABCEDGHIF** หรือในอีกกรณีอาจจะเป็น **FIHGDECBA** *(เพราะว่าเราไม่ได้เห็นทิศทางของรูป)* เพื่อให้เข้าใจตรงกัน เราจะอนุมานให้เป็นรูปแบบ **ABCEDGHIF**

ในเช้าวันหนึ่ง นักศึกษาตื่นมาแล้วปรากฏว่า **ลืม** รหัสที่ตัวเองได้ตั้งไว้ว่ามันคือรหัสอะไรกันแน่ ! นักศึกษารู้แค่ว่ารูปแบบที่นักศึกษาตั้ง ต้องใช้ทุกจุด (ในกรณีด้านบนคือทั้งหมด 9 จุด) ดังนั้นนักศึกษาเลยลองวาดจุดทุกรูปแบบที่เป็นไปได้ (สมมติว่าไม่มี Feature ในการล็อกโทรศัพท์หลังใส่รหัสผิดไปหลาย ๆ ครั้ง) นักศึกษาเลยสามารถใส่รหัสในทุกรูปแบบที่เป็นไปได้ เพื่อหารหัสที่ถูกต้อง

โชคดีที่ว่านักศึกษาเพิ่งเรียนการ Generate Permutation จากอ.วีเมื่อสักครู่นี้ เพื่อความเป็นระเบียบ เราจะเรียงรูปแบบรหัสที่จะลองตาม Lexicographic คือเรียงตามพจนานุกรม (จะได้ไม่ใส่รหัสแบบมั่ว ๆ สะเปะสะปะไปเรื่อย)

ทีนี้นักศึกษาเลยเริ่มใส่รหัสจาก **ABCDEFGHI** ซึ่งจะเป็นลำดับที่ 1 ต่อจากนั้นตามลำดับพจนานุกรมจะเป็น **ABCDEFGIH** เป็นลำดับที่ 2 ใส่ไปเรื่อย ๆ จนลองใส่ **ABCEDGHIF** เป็นลำดับที่ 130 จึงได้รหัสที่ถูกต้อง ดังนั้นจำนวนครั้งในการลองใส่คือ 130 ครั้ง

ทีนี้นักศึกษาเลยอยากลองให้คุณเขียนโปรแกรมหาว่า หากในอนาคตตั้งรหัสเป็นอย่างอื่นแล้วดันลืมอีก จะต้องลองใส่รหัสกี่ครั้งตามลำดับพจนานุกรม ถึงจะได้รหัสที่ถูกต้อง ตัวอย่างเช่น หากเราตั้งรหัสเป็น **IHGFEDCBA** เราจะต้องลองใส่ทั้งหมด 362880 ครั้งจึงจะได้รหัสที่ถูกต้อง แต่หากเราตั้งรหัสง่าย ๆ เช่น **ABCDEFGHI** ใส่แค่ 1 ครั้งก็ได้รหัสแล้ว

งานของนักศึกษา

จงหาว่า หากนักศึกษามีจำนวนจุด Pattern ทั้งหมด n จุด และมีชุดรหัสทั้งหมด m รหัส แต่ละรหัสต้องผ่านการลองใส่กี่ครั้งตามลำดับพจนานุกรม จึงจะสามารปลดล็อกโทรศัพท์ได้ กำหนดให้รหัสต้องมีทั้งหมด n ตัวเท่านั้น

ข้อมูลนำเข้า (Input)

บรรทัดที่ 1	จำนวนเต็ม n แทนจำนวนจุดใน Pattern และ m แทนจำนวนคำถามที่จะถาม โดยที่ $3 \leq n \leq 11$ และ $1 \leq m \leq 10$
บรรทัดที่ 2 ถึง $m+1$	รูปแบบของรหัสที่นักศึกษาอยากรู้ โดยประกอบไปด้วยตัวอักษรทั้งหมด n ตัว

ข้อมูลส่งออก (Output)

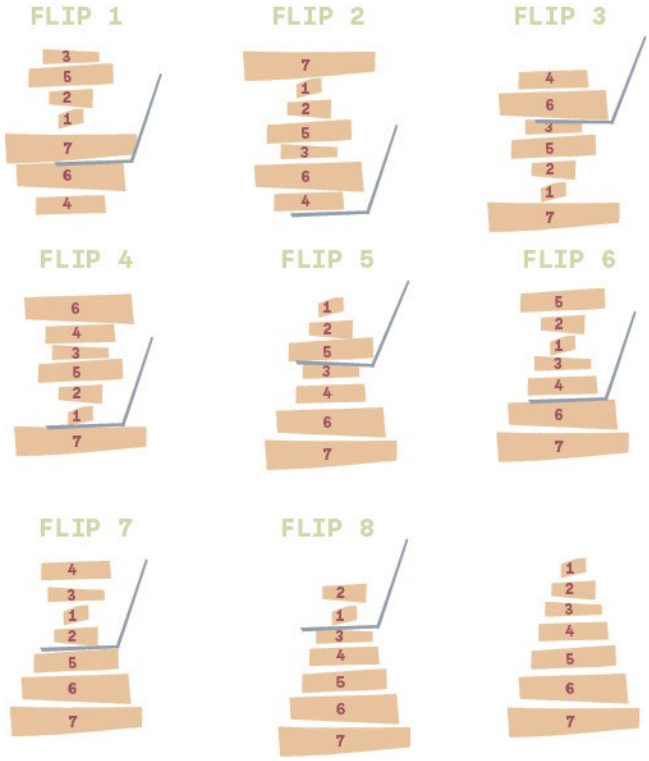
บรรทัดที่ 1 ถึง m	จำนวนครั้งที่ต้องใส่รหัสดังกล่าวตามลำดับพจนานุกรม จึงจะสามารถปลดล็อกโทรศัพท์ได้

ตัวอย่างข้อมูลนำเข้า ส่งออก (Examples of Input & Output)

Input	Output
9 3 ABCEDGHIF IHGFEDCBA ABCDEFGHI	130 362880 1
5 5 ABCED EDCAB BCEAD CDAEB ADECB	2 119 35 62 18

Pancake Sort

Pancake sort เป็น algorithm ที่ใช้สำหรับการจัดเรียงซึ่งมีรูปแบบมาจากพฤติกรรมการพลิก (flip) ของ pancake โดยใช้หลักการของการพลิกข้อมูลจากตำแหน่งแรกไปจนถึงตำแหน่งหนึ่ง เพื่อให้ได้ลำดับที่ต้องการ



โดยที่มาที่ไปของ algorithm นี้เกิดจากการจัดวาง pancake บนจาน โดยการวางนั้นจำเป็นต้องทำให้ชิ้นที่ใหญ่กว่าอยู่ด้านล่าง และชิ้นที่เล็กกว่าอยู่ด้านบน จะทำให้ได้การเรียงลำดับจากน้อยไปมาก

หลักการทำงานของ Pancake sorting

- หาแพนเค้กที่ใหญ่ที่สุดในส่วนที่เหลืออยู่: เริ่มต้นจากแพนเค้กทั้งกอง เลือกแพนเค้กที่ใหญ่ที่สุดจากกองที่ยังไม่ได้จัดเรียง (สมมติว่าเป็นส่วนย่อยของกองแพนเค้กทั้งหมดที่ยังเหลืออยู่)
- พลิกแพนเค้กใหญ่ที่สุดให้อยู่บนสุด: ถ้าแพนเค้กที่ใหญ่ที่สุดไม่ได้อยู่บนสุดของกอง ให้ทำการพลิกแพนเค้กจากตำแหน่งที่มันอยู่ไปจนถึงบนสุด เพื่อให้แพนเค้กใหญ่สุดมาอยู่บนสุด
- พลิกอีกครั้งเพื่อให้แพนเค้กใหญ่อยู่ในตำแหน่งที่ถูกต้อง: จากนั้นให้ทำการพลิกแพนเค้กจากบนสุดจนถึงตำแหน่งที่ต้องการ (ซึ่งก็คือตำแหน่งที่มันควรอยู่ในลำดับที่เรียงถูกต้อง) เพื่อให้แพนเค้กที่ใหญ่สุดลงไปอยู่ในตำแหน่งสุดท้าย
- ทำซ้ำ: จากนั้นทำซ้ำขั้นตอนเดียวกันสำหรับแพนเค้กที่เหลือ จนกระทั่งแพนเค้กทั้งหมดถูกจัดเรียงในลำดับที่ถูกต้อง

งานของนักศึกษา

ให้รับค่าตัวเลขจำนวนเต็ม n จำนวนจากผู้ใช้ จากนั้นทำการเรียงจำนวนที่รับเข้ามาโดยใช้ Pancake sorting algorithm และส่งออกเป็นลำดับของจำนนที่รับเข้ามา ที่ผ่านการเรียงลำดับจากน้อยไปมากเรียบร้อยแล้ว

ข้อมูลนำเข้า (Input)

บรรทัดที่ 1	จำนวนเต็ม n แสดงจำนวนของ Pancake
บรรทัดที่ 2	จำนวนเต็ม n จำนวน แสดงขนาดของ Pancake แต่ละชิ้น

ข้อมูลส่งออก (Output)

บรรทัดที่ 1	จำนวนเต็ม n จำนวนที่ผ่านการเรียงลำดับจากน้อยไปมาก

ตัวอย่างข้อมูลนำเข้า ส่งออก (Examples of Input & Output)

Input	Output
5 9 9 4 5 6	4 5 6 9 9
7 8 7 6 5 4 3 2	2 3 4 5 6 7 8

Quick Select

Quickselect ที่ใช้ Lomuto partition ทำงานโดยแบ่ง list เป็นสองส่วนตามแนวทางของ Lomuto partition scheme ซึ่งเป็นการเลือก pivot และแบ่ง list ให้ pivot อยู่ในตำแหน่งที่ถูกต้องหลังจากการ partition แต่ละครั้ง

ขั้นตอนการทำงานของ Quickselect โดยใช้ Lomuto Partition:

1. เลือก pivot: ปกติแล้วเลือก pivot เป็นตัวสุดท้ายของ list
2. ทำ Lomuto Partition: เลื่อนค่าที่น้อยกว่าหรือเท่ากับ pivot ไปทางซ้าย และค่าที่มากกว่า pivot ไปทางขวา:
   - กำหนดตัวชี้ i ไว้ที่ตำแหน่งแรก
   - เปรียบเทียบทุกค่าสำหรับตำแหน่ง j กับ pivot ถ้าค่าที่ตำแหน่ง j น้อยกว่าหรือเท่ากับ pivot ให้สลับค่าที่ตำแหน่ง i กับ j แล้วเลื่อนตัวชี้ i ไปข้างหน้า
   - สุดท้าย สลับ pivot ไปที่ตำแหน่งที่เหมาะสม (ตำแหน่งของ i) ทำให้ pivot อยู่ในตำแหน่งที่ถูกต้อง
3. ตรวจสอบตำแหน่งของ pivot:
   - ถ้าตำแหน่งของ pivot ตรงกับค่า k ที่เราต้องการ ให้คืนค่า pivot นั้นเป็นคำตอบ
   - ถ้า pivot อยู่ในตำแหน่งมากกว่า k ให้ทำ Quickselect กับลิสต์ฝั่งซ้ายของ pivot
   - ถ้า pivot อยู่ในตำแหน่งน้อยกว่า k ให้ทำ Quickselect กับลิสต์ฝั่งขวาของ pivot
4. ทำซ้ำจนกว่าจะหาคำตอบได้: ทำขั้นตอนการเลือก pivot และแบ่ง list ต่อไป จนกว่าจะพบค่าอันดับที่ต้องการ

งานของนักศึกษา

ให้รับค่าตัวเลขจำนวนเต็ม n จำนวนจากผู้ใช้ จากนั้นทำการหาจำนวนที่น้อยที่สุดลำดับที่ k จากชุดจำนวนเต็มที่รับเข้ามา และแสดงผล โดยที่ $k \leq n$

ข้อมูลนำเข้า (Input)

บรรทัดที่ 1	จำนวนเต็ม n แสดงถึงจำนวนของตัวเลขใน list
บรรทัดที่ 2	จำนวนเต็ม n แสดงสมาชิกในแต่ละตัว list
บรรทัดที่ 3	จำนวนเต็ม k แสดงตำแหน่งตัวเลขที่น้อยที่สุดลำดับที่ k

ข้อมูลส่งออก (Output)

บรรทัดที่ 1	จำนวนเต็มที่เป็นสมาชิกของ list ที่เข้ามาและเป็นจำนวนที่น้อยที่สุดลำดับที่ k

ตัวอย่างข้อมูลนำเข้า ส่งออก (Examples of Input & Output)

Input	Output
8 9 5 7 3 2 7 0 1 4	3
10 12 5 76 90 3 5 15 3 21 7 6	12
5 1 2 3 4 5 4	4

Merge Sort หรอ ? แปลก ๆ นะ

ในคาบเรียนเราได้เรียนรู้การเขียน merge sort จาก pseudocode แล้ว ซึ่งเป็นวิธีการเขียนแบบ recursive ทำให้ได้การทำงานแบบ top-down แต่ในการเขียนโปรแกรมนั้นสามารถทำได้อีกวิธีคือการเขียน iterative โดยการเราลองประยุกต์ใช้เทคนิคนี้กับ merge sort จะได้รูปแบบเป็น bottom-up ซึ่ง 2 วิธีการนี้ใช้หลักการ merge เหมือนกัน

ดูตัวอย่างได้ดังรูป

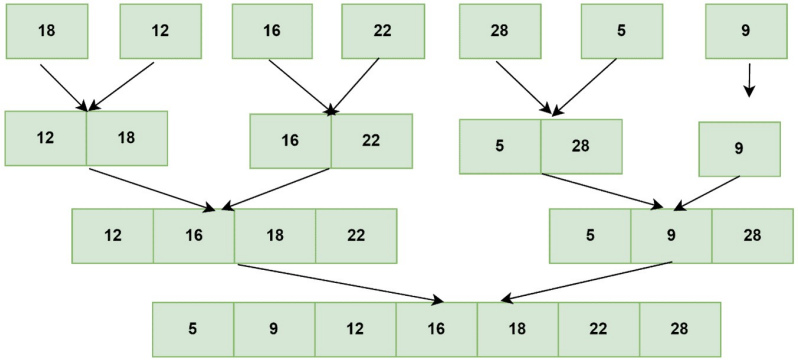


Figure 1: แผนภาพการทำงานของ Merge Sort

Algorithm 1 BottomUpMergeSort(A, n)

Require: A (a list of elements to be sorted), n (Number of elements in list)

```
if n < 2 then
    Return
end if
i ← 1                                          ▷ Initial subarray size
while i < n do
    j ← 0
    while j < n − i do
        if n < j + (2 × i) then
            merge(A, j, j + i, n)
                ▷ Merge subarray A[j ... j + i − 1] with A[j + i ... n − 1]
        else
            merge(A, j, j + i, j + (2 × i))
                ▷ Merge subarray A[j ... j + i − 1] with A[j + i ... j + 2 × i − 1]
        end if
        j ← j + 2 × i
    end while
    i ← i × 2
end while
```

งานของนักศึกษา

รับค่าจำนวนตัวเลขจากผู้ใช้ และตัวเลขแต่ละตัว จากนั้นทำการแสดงผลลำดับที่ถูกเรียงเรียบร้อยแล้ว

ข้อมูลนำเข้า (Input)

บรรทัดที่ 1	รับค่าจำนวนข้อมูล (n)
บรรทัดที่ 2	ค่าใน Array จำนวน n ตัว

ข้อมูลส่งออก (Output)

บรรทัดที่ 1	ข้อมูลที่รับเข้ามาที่ถูกเรียงลำดับจากน้อยไปมาก

ตัวอย่างข้อมูลนำเข้า ส่งออก (Examples of Input & Output)

Input	Output
6 -3 -8 3 -1 -2 -4	-8 -4 -3 -2 -1 3
10 152 -91 4 105 -15 46 173 166 14 163	-91 -15 4 14 46 105 152 163 166 173

Quicksort

Quicksort เป็น Sorting Algorithms ที่มี Strategy แบบ Divide-and-Conquer ที่เราจะทำการแบ่งออกเป็น 2 Subarray โดยการหาตำแหน่งของ Pivot โดยใช้ **Hoare's Partition** เรามาลองนำสิ่งที่เราเรียนจากอาจารย์มาเขียนโปรแกรมภาษา Java ดู

Algorithm 2 Quicksort(A)

Require: $A[l \dots r]$ (An array or subarray)
  if $l < r$ **then**
    $s \leftarrow$ HoarePartition($A[l \dots r]$) ▷ s is a split position
    Quicksort($A[l \dots s-1]$)
    Quicksort($A[s+1 \dots r]$)
  end if

Algorithm 3 HoarePartition(A)

Require: $A[l \dots r]$ (An array or subarray)
  $p \leftarrow A[l]$
  $i \leftarrow l$
  $j \leftarrow r+1$
  while $i < j$ **do**
    Repeat $i \leftarrow i+1$ until $A[i] \geq p$
    Repeat $j \leftarrow j-1$ until $A[j] \leq p$
    swap($A[i], A[j]$)
  end while
  swap($A[i], A[j]$) ▷ undo last swap when $i \geq j$
  swap($A[l], A[j]$)
  return j

งานของนักศึกษา

จงใช้วิธีการ Quicksort ในการเรียงตัวเลขใน Array จากน้อยไปมาก

ข้อมูลนำเข้า (Input)

บรรทัดที่ 1	รับค่าจำนวนข้อมูล (n)
บรรทัดที่ 2	ค่าใน Array จำนวน n ตัว

ข้อมูลส่งออก (Output)

บรรทัดที่ 1	ข้อมูลที่รับเข้ามาที่ถูกเรียงลำดับจากน้อยไปมาก

ตัวอย่างข้อมูลนำเข้า ส่งออก (Examples of Input & Output)

Input	Output
6 -3 -8 3 -1 -2 -4	-8 -4 -3 -2 -1 3
10 152 -91 4 105 -15 46 173 166 14 163	-91 -15 4 14 46 105 152 163 166 173

2D Closest Pair

ในงานนี้จะให้ทุกคนได้ลองเขียน algorithm ที่ใช้ในการหาระยะทางของจุดที่สั้นที่สุดบนระนาบ 2 มิติ โดยที่มีเงื่อนไขว่าต้องใช้วิธี divide-and-conquer ในการคำนวณ

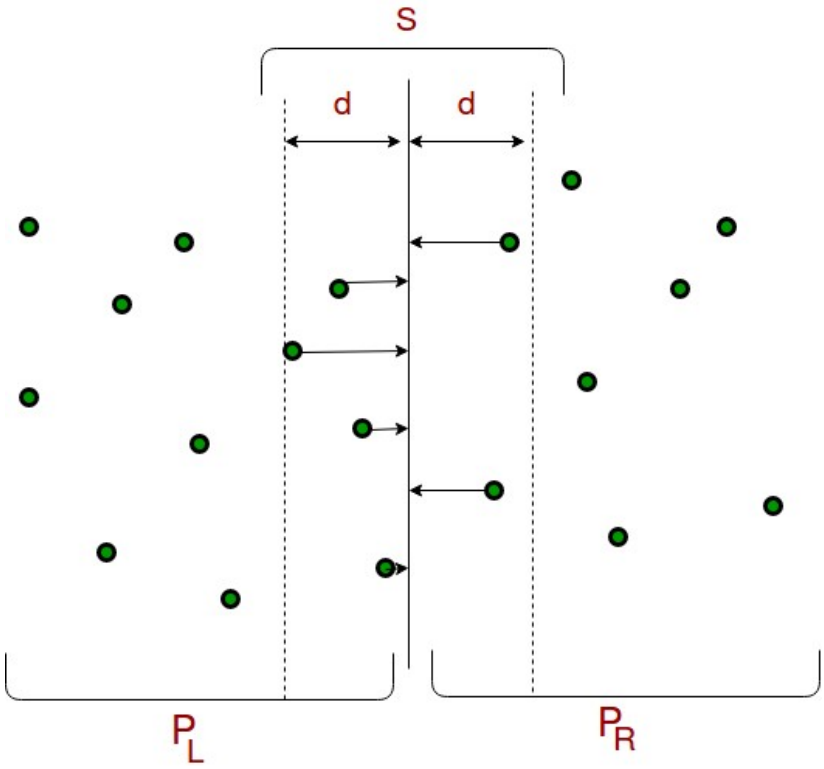


Figure 2: ตัวอย่างการทำงานของ 2D Closest Pair

1. แบ่งข้อมูล
    - เรียงลำดับจุดทั้งหมดตามแกน X
    - แบ่งจุดออกเป็นสองส่วนเท่า ๆ กัน โดยแบ่งตามเส้นแนวตั้งที่อยู่กลางพื้นที่พิจารณา แบ่งชุดจุดให้ได้สองกลุ่มซ้าย (Left) และขวา (Right)
2. เรียกทำซ้ำ (Recursion)
    - แก้ปัญหาในกลุ่มจุดด้านซ้าย (Left) และด้านขวา (Right) โดยหาคู่จุดที่ใกล้ที่สุดในแต่ละฝั่ง
    - เปรียบเทียบค่าระยะทางใกล้ที่สุดที่ได้จากทั้งสองกลุ่มว่าระยะทางขั้นต่ำของแต่ละฝั่งคือเท่าใด (เรียกว่าระยะทาง d)
3. การรวมผล (Conquer)
    - **ตรวจสอบจุดที่อยู่ใกล้เส้นแบ่ง (Mid-line):** ตรวจหาคู่จุดที่อยู่ใกล้กันแต่อยู่คนละฝั่งของเส้นแบ่ง โดยพิจารณาเฉพาะจุดที่มีระยะทางจากเส้นแบ่งไม่เกิน d
    - คำนวณระยะทางของคู่จุดในพื้นที่ตรงกลางระหว่างสองกลุ่มนี้ (Mid-region) และหาระยะทางที่สั้นที่สุดในพื้นที่นี้

งานของนักศึกษา

การหาระยะทางที่สั้นที่สุดของ 2 จุดใดๆบนระนาบ 2 มิติ โดยรับค่าจำนวนจุดจากผู้ใช้และรับค่าพิกัดของจุด x และ y ใด ๆ จากนั้นแสดงผลระยะที่สั้นที่สุด

ข้อมูลนำเข้า (Input)

บรรทัดที่ 1	รับค่าจำนวนจุดทั้งหมด (n)
บรรทัดที่ 2 ถึง $(n+1)$	ค่าพิกัด x, y ของแต่ละจุด n จุด

ข้อมูลส่งออก (Output)

บรรทัดที่ 1	ระยะทางที่สิ้นที่สุดทศนิยม 3 ตำแหน่ง (แสดงโดยการพิมพ์รูปแบบ `%.3f`)

ตัวอย่างข้อมูลนำเข้า ส่งออก (Examples of Input & Output)

Input	Output
5 10 1 0 6 8 3 2 0 9 4	1.414
6 10 1 0 6 8 2 2 0 4 4 6 6	2.236

ยันต์มงคล (Make a Wish)

หลังจากนักศึกษาได้ไปขอพรกับหลวงปู่วีให้สอบครั้งที่ 1 สำเร็จไปได้ด้วยดี ผลปรากฏว่าผลการสอบครั้งที่ 1 นั้นดีอย่างไม่น่าเชื่อ นักศึกษาจึงเลื่อมไสในหลวงปู่วีมาก ๆ กระทั่งได้ชวนเพื่อน ๆ ไปขอพรในการสอบครั้งที่ 2 ต่อไป รวมไปถึงเรื่องอื่น ๆ ที่ต้องการ เช่น ความรัก การเงิน การงาน ฯลฯ

ทีนี้นักศึกษาได้ยกโขยงกันไปเป็นจำนวน 83 คน ไปที่วัดพุทธบูชาตามเคย ได้พบกับหลวงปู่วีเหมือนเดิม ทีนี้หลวงปู่วีไม่ได้ให้สายลูกปัดเหมือนกับรอบที่แล้ว แต่ว่าให้กระดาษยันต์ที่ยาวมาก ๆ มาแผ่นหนึ่ง และยังได้บอกกับนักศึกษาทุกคนว่า

"หากโยมต้องการความเป็นสิริมงคลสูงสุด โยมต้องนำส่วนที่มีระดับความมงคลที่รวมกันได้มากที่สุดจำนวน 1 ผืน ตัดยังไงก็ได้ให้ผลรวมความมงคลได้มากสุด จึงจะได้ความโชคดี เจริญพร"

สมมติว่าหลวงปู่วีให้ผ้ายันต์ความยาว 10 เมตรมา แต่ละส่วนมีเขียนเลขดังนี้

-59	76	3	69	-26	-54	-79	-88	72	43

จะเห็นได้ว่า หากต้องการตัดผ้ายันต์ให้ได้ความมงคลสูงสุด นักศึกษาต้องตัดให้ได้เป็นรูปแบบดังนี้

-59	76	3	69	-26	-54	-79	-88	72	43

และจะได้ผลรวมความสิริมงคลคือ $76 + 3 + 69 = 148$ นั่นเอง

งานของนักศึกษา

จงหาว่า หากหลวงปู่วีให้ผ้ายันต์ที่มีความยาว n เมตร เราจะสามารถตัดอย่างไรให้ได้ผลรวมความเป็นสิริมงคลสูงที่สุดที่เป็นไปได้

ข้อมูลนำเข้า (Input)

บรรทัดที่ 1	จำนวนเต็ม n แทนความยาวผ้ายันต์ที่หลวงปู่วีให้มา โดยที่ $1 \leq n \leq 1,000,000$
บรรทัดที่ 2	จำนวนเต็ม l_i ทั้งหมด n ตัว แทนระดับความสิริมงคลของผ้ายันต์แต่ละจุด แต่ละตัวคั่นด้วยช่องว่าง 1 ช่อง โดยที่ $-100,000 \leq l_i \leq 100,000$

ข้อมูลส่งออก (Output)

บรรทัดที่ 1	ผลรวมความเป็นสิริมงคลที่มากที่สุด

ตัวอย่างข้อมูลนำเข้า ส่งออก (Examples of Input & Output)

Input	Output
10 -59 76 3 69 -26 -54 -79 -88 -72 43	148
25 -10 7 10 4 12 11 29 -8 15 18 -4 9 24 7 29 30 -8 1 20 14 -9 -2 -3 20 -5	226

Presorted Uniqueness

ในการแก้ปัญหาหลาย ๆ อย่างนั้น สามารถทำได้หลายวิธี และรูปแบบการแก้ปัญหาแตกต่างกัน แต่มีวิธีการหนึ่งที่เปลี่ยนรูปแบบข้อมูลเพื่อให้สามารถแก้ปัญหาได้ดีขึ้น คือ Transform and Conquer ปัญหาของการตรวจสอบจำนวนซ้ำ หรือ Uniqueness หากแก้ปัญหาด้วยการวนเพื่อหาจำนวนซ้ำไปเรื่อย ๆ วิธีปกติคือ $O(n^2)$ ซึ่งมีวิธีการที่สามารถแก้ปัญหานี้ได้อย่างเร็วขึ้นคือการเรียงลำดับก่อน จากนั้นวนเพื่อนับจำนวนซ้ำจากลำดับที่ถูกเรียง ซึ่งวิธีนี้หากใช้ sorting algorithm ที่มีประสิทธิภาพ จะสามารถลด complexity เหลือเพียง $O(n \log n)$ ได้ ซึ่ง algorithm นี้เรียกว่า PresortElementUniqueness

Algorithm 4 PresortElenentUniqueness(A)

Require: A, an arbitrary array
  Sort the array A ▷ You can sort array by any method
  for i from 0 to Length(A) -2 **do**
    if $A[i] = A[i+1]$ **then**
      Return False ▷ Duplicates found, not all elements are unique
    end if
  end for
  Return True ▷ No duplicates found

โดยในโจทย์ข้อนี้จะให้นักศึกษาได้ประยุกต์ใช้ `PresortElementUniqueness(A)` ในการลำดับของตัวเลขทั้งหมดไม่มีจำนวนซ้ำ โดยขั้นตอนของการ เรียงลำดับนั้นสามารถใช้ algorithm ไหนก็ได้แต่ต้องทำให้ได้ algorithm ที่มีขนาดน้อยกว่า $O(n^2)$ และ ห้ามใช้ function สำเร็จรูปในการเรียงลำดับทุกกรณีรวมถึง `merge()` แต่ยกเว้น `swap()`

งานของนักศึกษา

ให้เขียนโปรแกรมเพื่อรับค่าลำดับจากผู้ใช้ จากนั้นแสดงตัวเลขในลำดับที่ลบจำนวนซ้ำออกไป เช่น 3 4 4 5 จะได้ 3 4 5 โดยที่ 4 ที่ซ้ำอีกตัวจะถูกลบออกจากลำดับ

ข้อมูลนำเข้า (Input)

บรรทัดที่ 1	จำนวนของลำดับที่รับเข้าทั้งหมด (n)
บรรทัดที่ 2	ลำดับตัวเลข n_i

ข้อมูลส่งออก (Output)

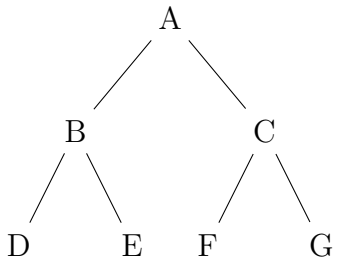
บรรทัดที่ 1	ตัวเลขในลำดับที่ไม่เกิดการซ้ำกันที่เรียงจากน้อยไปมาก

ตัวอย่างข้อมูลนำเข้า ส่งออก (Examples of Input & Output)

Input	Output
10 1 1 1 2 2 2 2 3 4 5	1 2 3 4 5
5 5 5 5 4 3	3 4 5
6 6 6 6 6 6 6	6

Heap Sort

Binary Heap เป็น Data Structure ประเภทหนึ่งที่ประยุกต์หลักการของ **Complete Binary Tree** เพื่อเก็บข้อมูลต่าง ๆ *(หวังว่าเรายังจำเรื่อง Binary Tree จากวิชา CPE112 ได้นะ)*



โดย Binary Heap จะมี 2 ประเภท นั่นคือ

- Min Heap
- Max Heap

Min Heap คือการที่โหนดแต่ละโหนดมีเงื่อนไขที่ว่า Parent Node ต้องมีค่าน้อยกว่าหรือเท่ากับ Child Node ของมันเสมอ ($A \leq B$ และ $A \leq C$) ส่วน Max Heap คือการที่โหนดแต่ละโหนดมีเงื่อนไขว่า Parent Node ต้องมีค่ามากกว่าหรือเท่ากับ Child Node ของมันเสมอ ($A \geq B$ และ $A \geq C$)

หากเราสังเกตจากนิยามด้านบนแล้ว ค่าที่มากที่สุดใน Max Heap จะต้องเป็น Root ของต้นไม้ กล่าวคือ A จะต้องมีค่ามากที่สุด

ในทำนองเดียวกัน ค่าที่น้อยที่สุดใน Min Heap ก็จะต้องเป็น Root ของต้นไม้เหมือนกัน หากว่าต้นไม้ด้านบนเป็น Min Heap จะได้ว่า A คือค่าที่น้อยที่สุดใน Heap นั่นเอง

โดยหากเราได้รับ Array ที่มีสมาชิกทั้งหมด n ตัว หากเราต้องการแปลง Array ให้กลายเป็น Max Heap เราสามารถใช้ขั้นตอนวิธีการดังนี้

Algorithm 5 MaxHeapBottomUp($A[1 \dots n]$)

Require: $A[1 \dots n]$, an arbitrary array (which index **starts at 1**)

```
for i ← ⌊n/2⌋ to 1 do
    k ← i
    v ← A[k]
    heap ← False
    while not heap & (2 × k) ≤ n do
        j ← (2 × k)
        if j < n then                ▷ This mean there are two children of this node
            if A[j] < A[j + 1] then
                j ← j + 1
            end if
        end if
        if v ≥ A[j] then
            heap ← True
        else
            A[k] ← A[j]
            k ← j
        end if
    end while
    A[k] ← v
end for
```

ยกตัวอย่างเช่น เรามีสมาชิกใน Array ทั้งหมด 10 ตัว ดังนี้ $\{52, 60, 45, 23, 1, 13, 70, 48, 90, 84\}$ เมื่อเรานำ Array ไปทำ Heapify ให้ได้ Max Heap เราจะได้ผลการทำ Heapify ดังนี้ $\{90, 84, 70, 60, 52, 13, 45, 48, 23, 1\}$

จากเงื่อนไขด้านบนแล้ว เราสามารถเรียงลำดับ Array จากน้อยไปมากหรือมากไปน้อยได้ โดยการประยุกต์ใช้ Heap เข้าช่วย ซึ่งจะได้วิธีการ Sort แบบใหม่ เรียกว่า **Heap Sort** นั่นเอง โดยขั้นตอนของการทำ Heap Sort จะมีขั้นตอนดังนี้

1. เริ่มจากให้ตัวแปร i มีค่าเป็น n เมื่อ n คือจำนวนสมาชิกใน Array
2. ทำการ **Heapify** Array ของเรา โดยให้ขอบเขตการมองเห็นสมาชิก Array คือ 1 ถึง i (*สังเกตว่าเราจะได้สมาชิกที่มีค่ามากที่สุดเป็นตัวแรกแล้ว ก็คือ* `arr[0]`)
3. ทำการ**สลับ** `arr[0]` และ `arr[i]` (*สังเกตว่าเราจะทำให้ตัวที่มีค่ามากที่สุด ณ ตอนนี้ สลับไปอยู่ท้ายสุด*)
4. ลดค่า i ทีละหนึ่ง
5. ทำซ้ำข้อ 2 - 4 เมื่อค่า i ยังมากกว่า 0

ท้ายที่สุดแล้วเราก็จะได้ Array ที่เรียงจากน้อยไปมากแล้วนั่นเอง

งานของนักศึกษา

จงนำสิ่งที่ได้เรียนรู้มาใน Lab Sheet มาปรับให้เล็กน้อย โดยให้นักศึกษาทำการ Heap Sort Array จาก**มากไปน้อย** โดยใช้การ Heapify ให้กลายเป็น Min Heap *(เปลี่ยน Pseudocode ด้านบน เพียงเล็กน้อยเท่านั้น)*

ข้อมูลนำเข้า (Input)

บรรทัดที่ 1	จำนวนของข้อมูล (n) โดยที่ $1 \leq n \leq 1,000,000$
บรรทัดที่ 2	ลำดับตัวเลข A_i

ข้อมูลส่งออก (Output)

บรรทัดที่ 1	ผลจากการทำ Min Heapify **ครั้งแรก**
บรรทัดที่ 2	ลำดับของเลขที่เรียงจาก**มากไปน้อย**

ตัวอย่างข้อมูลนำเข้า ส่งออก (Examples of Input & Output)

Input	Output
10 52 60 45 23 1 -13 70 48 90 84	-13 1 45 23 60 52 70 48 90 84 90 84 70 60 52 48 45 23 1 -13
5 5 4 3 2 1	1 2 3 5 4 5 4 3 2 1
1 60	60 60

อธิบายตัวอย่าง

ในตัวอย่างแรก เมื่อข้อมูลนำเข้าเป็น `52 60 45 23 1 -13 70 48 90 84` ในขั้นตอนแรกของ Heap Sort คือการนำ Array ไปทำ Heapify ผลลัพธ์ในบรรทัดที่ 1 คือผลที่ได้ของ Array จากการทำ Min Heapify ครั้งแรกสุด ซึ่งจะได้ `-13 1 45 23 60 52 70 48 90 84` และบรรทัดที่ 2 คือ Array ที่เกิดจากการเรียงจากมากไปน้อยโดยการใช้ Heap Sort ก็คือ `90 84 70 60 52 48 45 23 1 -13`

หวย (Lottery)

ในกิจกรรม CPE Games 2026 ที่จะจัดขึ้นปีหน้า ภาควิชาได้มีการขายสลากกระดาษที่ประกอบด้วยตัวอักษรยาวเหยียด โดยจะมีการออกรางวัล 1 ครั้ง นักศึกษาที่ถูกหวยตัวนี้จะได้รับรางวัลเป็นเงิน 10 ล้านบาทจาก อ.วี

นักศึกษาเข้าร่วมกิจกรรมนี้โดยซื้อหวยดังกล่าวจำนวน 1 ใบ ที่มีความยาว n ตัวอักษร การออกรางวัลคือการประกาศสายอักขระที่ถูกหากรางวัล หากในหวยที่นักศึกษาซื้อไปมีสายอักขระที่ถูกรางวัลอยู่ในนั้น ก็จะถือว่านักศึกษาถูกหวยนั่นเอง

ตัวอย่างเช่น หากนักศึกษาซื้อหวยที่มีสายอักขระดังนี้

DKRLSMDNRJTKHNF

และผลประกาศคือสายอักขระ **SMD** นักศึกษาจะสังเกตได้ว่า นักศึกษาถูกหวย โดย Pattern ดังกล่าวเริ่มที่ตัวอักษรที่ 4 *(เมื่อให้ตัวอักษรแรกเป็นลำดับที่* 0*)*

เพื่อความรวดเร็วในการตรวจหวย นักศึกษาจึงประยุกต์ใช้วิธีการของ **Horspool** ในการตรวจว่าถูกหวยหรือไม่ ซึ่งจะมีการเลื่อนตัวอักษรตรวจจำนวน 2 ครั้ง

งานของนักศึกษา

จงใช้ขั้นตอนของ Horspool ในการตรวจสอบว่านักศึกษาถูกหวยหรือไม่ เมื่อให้ String ความยาว n และ String ที่ถูกรางวัลความยาว m หาว่านักศึกษาจะต้องใช้การเลื่อน (Shift) ทั้งหมดกี่ครั้ง และลำดับตัวอักษรตัวแรกที่ทำให้นักศึกษาถูกรางวัล คือตำแหน่งเท่าใด

ข้อมูลนำเข้า (Input)

บรรทัดที่ 1	จำนวนเต็ม m และ n แทนความยาวของ String หวยที่ซื้อ และ String หวยที่ถูกตามลำดับ โดยที่ $m \geq n$ เสมอ
บรรทัดที่ 2	String ของหวยที่นักศึกษาซื้อ
บรรทัดที่ 3	String ของหวยที่ถูกรางวัล

ข้อมูลส่งออก (Output)

บรรทัดที่ 1	หากถูกรางวัล ให้พิมพ์ **YES** ตามด้วยจำนวนครั้งของการเลื่อน และ ตำแหน่งแรกที่ทำให้นักศึกษาถูกหวย แต่ถ้าไม่ถูกรางวัล ให้พิมพ์ **NO** ตามด้วยจำนวนครั้งของการเลื่อน และตัวเลข **-1** ทั้งหมดคั่นด้วยช่องว่าง 1 ช่อง (ดูตัวอย่างประกอบ)

ตัวอย่างข้อมูลนำเข้า ส่งออก (Examples of Input & Output)

Input	Output
15 3 DKRLSMDNRJTKHNF SMD	YES 2 4
35 13 CPETHREESEVENKMUTTVERYCUTEBUTSINGLE AJWEEHANDSOME	NO 3 -1
25 7 TAGTAGCAGTAGTAGTAGTAGCAGA TAGCAGA	YES 4 18

Hash Table

ตารางแฮช (Hash Table) เป็นตารางที่มีหน้าที่ในการเก็บค่าต่าง ๆ ซึ่งเป็นเบื้องหลังการเก็บข้อมูลแบบ Dictionary โดยการ Hashing คือการกระจาย Key ออกภายในอาร์เรย์ 1 มิติโดยเราจะใช้ **Hash Function** ในการคำนวณว่า Key แบบนี้จะถูกเก็บไว้ในช่องใด

สมมติว่าเราต้องการเก็บข้อความที่มีทั้งหมด n ข้อความ ดังต่อไปนี้ `{WEE, ALGO, CPE, KMUTT}` เราจะต้องนิยามฟังก์ชัน Hash ก่อน โดยเรานิยาม Hash Function ดังนี้

$$h(S) = \left(\sum_{i=0}^{len(S)-1} order(s_i) \right) \mod Z$$

เมื่อให้ S คือสตริงข้อความ, $order(s_i)$ เป็นลำดับของตัวอักษรที่ i ในคำนั้น เช่น A คือ 1, B คือ 2, ..., Z คือ 26 และ Z คือขนาดของ Hash Table

สมมติให้ $Z = 10$

ดังนั้น $order(\texttt{WEE}) = (23+5+5) \mod 10 = 3$ ดังนั้นคำว่า **WEE** จะไปอยู่ในช่องที่ 3 ของ Hash Table นั่นเอง

ในขณะที่ **ALGO** จะอยู่ช่องที่ 5, **CPE** จะอยู่ช่องที่ 4 ส่วน **KMUTT** การคำนวณเมื่อเข้า Hash Function จะได้เท่ากับ 5 แต่ว่าช่องที่ 5 มี **ALGO** อยู่ในนั้นแล้ว ดังนั้นมันจะถูกถัดไปเช็คอีก 1 ช่อง ถ้าว่างจะเข้าไปอยู่ช่องนั้น ถ้าไม่ว่างก็จะถัดไปอีกช่อง ทำนองนี้ไปเรื่อย ๆ เราสังเกตได้ว่าสิ่งนี้คือการทำ **Closed Hashing** นั่นเอง

ดังนั้น Hash Table ของข้อความนี้คือ

`{NULL NULL NULL WEE CPE ALGO KMUTT NULL NULL NULL}`

งานของนักศึกษา

จงสร้างตาราง Hash แบบ **Closed Hashing** ที่มีทั้งหมด Z ช่อง เพื่อเก็บข้อมูลคำจำนวนทั้งหมด n คำ

ข้อมูลนำเข้า (Input)

บรรทัดที่ 1	จำนวนเต็ม Z และ n แทนจำนวนช่องของ Hash Table และจำนวนคำที่จะใส่ตามลำดับ โดยที่ $Z \geq n$ เสมอ
บรรทัดที่ 2 ถึง $n+1$	String ของคำ

ข้อมูลส่งออก (Output)

บรรทัดที่ 1	ผลของ Hash Table หลังจากเก็บข้อมูลทั้งหมด ช่องไหนไม่มีคำในนั้นให้แสดงเป็น **NULL** แต่ละช่องคั่นด้วยช่องว่าง 1 ช่อง

ตัวอย่างข้อมูลนำเข้า ส่งออก (Examples of Input & Output)

Input	Output
10 4 WEE ALGO CPE	NULL NULL NULL WEE CPE ALGO KMUTT NULL NULL NULL
7 6 AA B CD EF GH III	CD GH AA B EF NULL III