Editorial (Practice)

บริษัท

1 seconds, 256 megabytes

Abridged Problem Statement

โจทย์คือมีต้นไม้ต้นหนึ่ง ประกอบด้วยจุดยอด N จุด แต่ละจุดยอดมีตัวเลขอยู่ค่าหนึ่งโดยค่านั้นจะไม่ซ้ำกันเลย เราจะไม่ทราบค่าตัวเลขนั้น อย่างไรก็ตาม เราทราบว่า บนเส้นทางจากจุดยอดปัจจุบันถึงราก มีตัวเลขกี่ตัวที่มีค่าน้อยกว่าตัวเลขปัจจุบัน เราต้องการเรียงจุดยอดจากที่มีเลขสูงสุดไปหาต่ำสุด

Subtask 1

ทำการสุ่ม (หรือไล่หา) ทุกวิธีการจัดตัวเลขที่เป็นไปได้ ซึ่งมีอยู่ทั้งหมด $N!$ วิธี หลังจากนั้นไปตรวจสอบว่าวิธีเหล่านั้นทำให้เกิดผลลัพธ์ตามที่กำหนดหรือเปล่าด้วยอัลกอริทึม $\mathcal{O}(N^2)$

Time Complexity: $\mathcal{O}(N!)$

Subtask 2, 3

มี 2 วิธีหลัก ๆ วิธีแรกคือพิจารณาจากรากของต้นไม้ไปหาทุก ๆ จุดยอดที่เหลือ เพื่อพิจารณาว่าจุดยอดปัจจุบันจะต้องอยู่ส่วนใดของลำดับในคำตอบ (วิธีนี้เราจะไม่แทนค่าว่าแต่ละจุดยอดมีค่าค่าใดอยู่ แต่เราจะเก็บข้อมูลว่าจุดยอดใดจะต้องมีค่าน้อยกว่าจุดยอดใด จุดยอดใดต้องมีค่ามากกว่าจุดยอดใด เราเรียกข้อมูลนี้ว่า partially ordered set หรือเซตอันดับบางส่วน)

เริ่มจากการค้นหาจากรากมาสู่ลูกแต่ละตัว แล้วพิจารณาว่าจุดยอดปัจจุบันจะต้องน้อยกว่าจุดยอดใดบ้างบนเส้นทางจากจุดยอดปัจจุบันไปจนถึงราก แล้วทำการเก็บข้อมูล โดยเราอาจเก็บข้อมูลเป็นกราฟมีทิศทางเสริมที่มีเซตของเส้นเชื่อมนิยามโดย u ไปหา v ก็ต่อเมื่อ จุดยอด u บนต้นไม้ จะต้องมีค่ามากกว่าจุดยอด v อย่างแน่นอน สังเกตว่าหากทำ topological sort บนกราฟนี้จะได้คำตอบทันที หรือ อาจเก็บลำดับที่เป็นไปได้ (อาจมีได้หลายวิธี แต่เก็บวิธีเดียวไปเลย) เมื่อมีจุดยอดลูกเข้ามา เราก็สามารถไล่หาในลำดับนี้ได้ว่าควรแทรกจุดยอดลูกในตำแหน่งไหน เนื่องจากการพิจารณา subtree ที่ต่างกันนั้นอิสระต่อกัน วิธีนี้จึงไม่มีปัญหาเช่นกัน

วิธีต่อมาคือพิจารณาจาก subtree ต้นเล็กๆ ไปหา root โดยเราจะพิจารณาคำตอบที่เป็นไปได้ของ subtree แต่ละต้น สมมติพิจารณาจุดยอด u หาก v_i เป็นลูกของ u สำหรับทุก i ใน I แล้ว สมมติว่า subtree ที่ v_i ทุก subtree มีคำตอบอยู่แล้ว เมื่อนำคำตอบของแต่ละ subtree มาแทรกอย่างไรก็ได้ (อาจนำลำดับมาต่อกันเลยก็ได้) จะได้ว่าคำตอบของทุก subtree ยังถูกต้องอยู่ ที่เหลือคือการแทรก u ในลำดับคำตอบ ซึ่งสามารถพิจารณาได้ไม่ยากโดยการไล่หาว่าแทรกในจุดไหนได้บ้างภายในลำดับ แต่ส่วนนี้จะทำให้ไม่สามารถนำลำดับลูกมาต่อกันอย่างสุ่มได้ วิธีในการรักษา

ลำดับที่ง่าย คือการกำหนดจุด u ไว้ก่อนแล้วพิจารณาว่าจุดยอดลูกจุดใดบ้างที่ต้องอยู่ทางซ้าย จุดใดบ้างที่ต้องอยู่ทางขวา เท่านี้ก็จะทำให้สามารถจัดลำดับได้ตามต้องการ

ทั้งสองวิธีสามารถทำได้ใน $\mathcal{O}(N^3)$ และ $\mathcal{O}(N^2)$ ได้ ขึ้นอยู่กับวิธีการเขียน และการจัดการข้อมูล

Subtask 4

สืบเนื่องมาจากปัญหาย่อยที่ผ่านมา เราสามารถพัฒนาให้ดียิ่งขึ้นได้อีก สำหรับวิธีการที่ไล่จาก subtree ขึ้นมา อัลกอริทึมที่ดีพอจะใช้เนื้อหาเกินการแข่งขันระดับชาติ ในที่นี้จึงขอไม่อธิบายมากแต่จะให้ข้อมูลสำหรับผู้สนใจ สำหรับวิธีนี้เราสามารถใช้โครงสร้างข้อมูลชนิดหนึ่งเรียกว่า Sack ในการเก็บชุดลำดับจากแต่ละ subtree เอามารวมกันแล้วส่งขึ้นไปได้ โดยอาจดูเหมือนต้องเก็บข้อมูล $\mathcal{O}(N^2)$ แต่เราสามารถใช้เพียง $\mathcal{O}(N \log^2 N)$ หรือ $\mathcal{O}(N \log N)$ ได้ด้วยโครงสร้างข้อมูลนี้ (อ่านต่อได้ใน https://codeforces.com/blog/entry/44351)

Official Solution

เราจะพิจารณาจากบนลงล่างและทำการแทรกข้อมูลลงในลำดับเรื่อย ๆ โดยเราต้องการโครงสร้างข้อมูลบางอย่างที่ทำให้เราสามารถแทรกจุดยอดปัจจุบันลงไปในที่ที่มันควรอยู่ได้ กล่าวคือ ต้องการโครงสร้างข้อมูลที่รองรับการดำเนินการดังนี้:

1. แทรกจุด u ก่อน ตัวชี้ตำแหน่งค่าหนึ่ง
2. หาตัวชี้ตำแหน่งของ v
3. หาว่า บนเส้นทางจาก u ไปถึงรากนั้น หากเรียงลำดับตามมูลค่าแล้วจุดยอดลำดับ i คือจุดยอดใด

ด้วยเวลา $\mathcal{O}(\log N)$ สำหรับทั้ง 3 การดำเนินการ

สังเกตว่าเราสามารถใช้ต้นไม้ค้นหาทวิภาค (binary search tree) เพื่อรองรับการดำเนินการที่ 3 ได้ หากเราค่อยๆ เดินจากรากไปยังแต่ละจุดยอด เมื่อลงข้างล่างก็นำค่าปัจจุบันใส่ binary search tree ด้วยวิธีการเปรียบเทียบค่าแบบเปรียบเทียบมูลค่าของจุดยอดนั้นๆ เมื่อเดินขึ้นข้างบนก็นำค่าออกจาก binary search tree นอกจากนั้นต้องเก็บค่าไว้ด้วยว่าใน subtree ของ binary search tree นั้นจะมีขนาดเท่าไร เพื่อให้สามารถหาว่าตัวที่ i ตามลำดับ in-order นั้นอยู่ที่จุดยอดใดใน binary search tree ภายในเวลาอันรวดเร็ว

ที่เหลือคือการดำเนินการ 2 อย่างแรก ซึ่งเราสามารถใช้ต้นไม้ค้นหาทวิภาคอีกต้นได้ เพื่อเก็บลำดับที่เป็นไปได้

เนื่องจาก binary search tree นั้นอาจช้า เราจึงจำเป็นต้องพัฒนาด้วย self balancing binary search tree หรือ randomized data structures ทางผู้เขียนแนะนำให้ใช้ treap เนื่องจากสามารถจัดการ split/merge ได้ไม่ยาก

R Sequence

1 second, 256 megabytes

Abridged Problem Statement

เราต้องการนับว่ามีวิธีการตัดส่วนหนึ่งของอาเรย์ได้กี่วิธีที่จะทำให้อาเรย์ย่อยที่ถูกตัดนั้นเป็น R Sequence

Subtask 1 - 2

สำหรับ Sequence หนึ่ง เราสามารถตรวจสอบว่าเป็น R Sequence หรือไม่ด้วยวิธีการดังนี้

1. หากลำดับปัจจุบันเป็นลำดับว่าง จะถือว่าเป็น R Sequence
2. หากลำดับไม่ว่าง และ ตัวแรกสุดมีค่าเท่ากับตัวท้ายสุด ให้ตรวจสอบว่าลำดับปัจจุบันที่ลบตัวแรกกับตัวท้ายออกไปนั้นเป็น R Sequence หรือไม่
3. หากลำดับไม่ว่าง และ ตัวแรกสุดมีค่าไม่เท่ากับตัวท้ายสุด ถือว่าไม่เป็น R Sequence

เราทำการไล่หาทุกลำดับย่อยที่เป็นไปได้ (ซึ่งมี N^2 ลำดับย่อย) หลังจากนั้นใช้วิธีข้างต้นในการตรวจสอบว่าเป็น R Sequence หรือไม่ ภายในเวลา $\mathcal{O}(N)$

Time Complexity: $\mathcal{O}(N^3)$

Subtask 3

ก่อนอื่นจะต้องพูดถึงวิธีอื่นที่สามารถตรวจสอบได้เช่นกันว่าลำดับหนึ่งเป็น R Sequence หรือไม่ดังนี้

1. สร้าง stack มาก่อน
2. ค่อย ๆ พิจารณาจากซ้ายไปขวา หาก stack ไม่ว่างและตัวปัจจุบันมีค่าเท่ากับตัวบนสุดของ stack ให้ลบตัวบน stack ออก แต่ในทางกลับกัน หาก stack ว่าง หรือตัวปัจจุบันไม่เท่ากับตัวบนสุด ให้นำตัวปัจจุบันวางลงบน stack
3. ทำจนจบ หาก stack ว่าง ถือว่าเป็น R Sequence แต่หากไม่ว่างจะถือว่า ไม่เป็น R Sequence

เราเริ่มต้นจากการกำหนดค่าเริ่มต้น L ตั้งแต่ 1 ถึง N แล้วพิจารณาลำดับย่อยจาก L ถึง R สำหรับ R จาก L ถึง N จะได้ว่าเราสามารถตรวจสอบไปด้วยได้เลยว่าลำดับย่อยจาก L ถึง R เป็น R Sequence หรือไม่ โดยที่เดินต่อไปด้วยพร้อม ๆ กัน

Time Complexity: $\mathcal{O}(N^2)$

Subtask 4

กลายเป็นโจทย์คณิตศาสตร์ ถามว่ามีลำดับย่อยกี่แบบที่ยาวเป็นเลขคู่ สามารถคิดด้วยวิธีการทางคอมบินาทอริกส์ได้ดังนี้ สำหรับเลขคู่ i จาก 2 ถึง N พิจารณาว่ามีกี่ลำดับย่อยที่ยาว i เราจะสังเกตได้ว่ามีอยู่ทั้งหมด $N-i+1$ ลำดับย่อยพอดี

Subtask 5, 6, 7

เราต้องการหาวิธีที่ดีกว่า $\mathcal{O}(N^2)$ ที่ทำได้คล้าย ๆ กัน เราเริ่มต้นได้ด้วยการสังเกต

สังเกตว่าหากทำตามอัลกอริทึมในปัญหาย่อยที่ 3 เราจะมี stack ทั้งหมด N อัน แต่ละอันเริ่มจากพิกัดทางซ้ายที่ต่างกันออกไป แต่ในความเป็นจริงแล้วเราสามารถใช้เพียงอันเดียวโดยเริ่มที่ตำแหน่งเริ่มต้นได้เลย ค่อย ๆ ไล่มาทางขวาไปเรื่อย ๆ เหมือนเดิม แต่เปลี่ยนลักษณะคำถามใหม่ จากเดิมที่ว่า stack โล่งหรือไม่ คำถามจะกลายเป็น "หากด้านขวาของลำดับย่อยอยู่ตำแหน่งปัจจุบัน จะมีด้านซ้ายได้กี่แบบที่ทำให้ stack โล่งหากคิดแบบเดิม" เราสังเกตว่า หากมี stack S อยู่ ผ่านชุดการดำเนินการ P แล้วยังคงข้อมูลเป็น S เหมือนเดิม แสดงว่าหากนำ stack โล่งมาผ่านชุดการดำเนินการ P แล้วจะทำให้ stack โล่งเช่นกัน รวมถึงในทางกลับกันด้วย

คำถามตอนนี้คือ หากเราทราบหน้าตาของ stack หลังผ่านแต่ละจุดไปแล้ว ตั้งแต่จุดเริ่มต้น (0) ถึง ผ่าน N นั้น มีอยู่กี่คู่ที่ลักษณะเหมือนกันทุกประการ

การเปรียบเทียบโดยตรงก็จะยังเป็น $\mathcal{O}(N^3)$ อยู่ เพราะเทียบทุกคู่ แต่ละคู่ใช้เวลา $\mathcal{O}(N)$ ในการเทียบ

เราสามารถพัฒนาปรับปรุงได้เป็น $\mathcal{O}(N^2)$ โดยเก็บเป็นโครงสร้างลักษณะ map of stack กล่าวคือเก็บว่า stack หน้าตาแบบนี้พบแล้วกี่ครั้ง แล้วทำจากซ้ายไปขวาตามปกติ

วิธีที่จะทำให้เหลือแค่ $\mathcal{O}(N)$ คือ แทนที่จะเก็บ stack ใส่ map เราจะใช้ rolling hash มาเก็บค่าของ stack แทน เมื่อทำการเทียบจึงสามารถทำได้โดยง่าย ว่าพบเจอ stack ลักษณะปัจจุบันกี่ครั้งแล้ว เนื่องจากจำนวน stack ที่เป็นไปได้ทั้งหมด มีถึง $\mathcal{O}(N)$ แบบ จึงมีโอกาสสูงที่จะเกิดข้อผิดพลาดหากทำ hash โดยตรง ทางผู้แต่งจึงแนะนำให้ใช้วิธี double hash หรือ triple hash เพื่อให้มั่นใจว่าถูกต้อง

Alternative Solution

นอกจากการใช้ stack แล้ว ยังมีวิธีอื่นที่แตกต่างออกไปเลย คือการสร้าง Trie แล้วเดินบน Trie ดังนี้ หากพบเจอจำนวนอะไรก็ให้เดินไปหาลูกที่แทนค่าจำนวนนั้น ยกเว้นว่า parent edge จะเป็นจำนวนเดียวกัน ให้ขึ้นด้านบน ทุกครั้งที่ย้ายที่ ให้นำตัวเลขที่อยู่บนจุดยอดไปบวกในคำตอบ แล้วเพิ่มค่าตัวเลขบนจุดยอดปัจจุบันไปอีก 1 ทำซ้ำจนหมดทั้งลำดับแล้ว จะได้คำตอบ

ง่าย?

2 seconds, 256 megabytes

Abridged Problem Statement

ให้จำนวนขนาดใหญ่ A กับ B แล้วให้หา $A + B$

Subtask 1

เป็นการเขียนโปรแกรมพื้นฐาน สามารถใช้ตัวแปรประเภท `int` เก็บจำนวนได้

Subtask 2

จำนวนใหญ่เกินขอบเขต `int` แต่เรามีประเภท `long long` ที่สามารถเก็บค่าได้ถึง 64-bit ซึ่งสามารถใช้เก็บค่าได้

Subtask 3

เนื่องจาก $A = 0$ เราสามารถรับค่า B แล้วส่งออกได้เลย โดยอาจรับค่าเป็นสตริง (string) ได้

Subtask 4, 5

เราทำการบวกเลขโดยตรง ด้วยวิธีคล้ายการบวกเลขแบบทดเลข คือดูเลขหลักหน่วย จับบวกกัน ทดเลข ต่อมาดูเลขหลักสิบ จับบวกกัน และบวกกับเลขที่ทดด้วย

หากเขียนดีจะสามารถทำตรง ๆ ใน $\mathcal{O}(a+b)$ ได้ แต่หากระบบทดไม่ดีอาจจะต้องใช้เวลา $\mathcal{O}((a+b)^2)$ ได้ เมื่อ a แทนจำนวนหลักของ A และ b แทนจำนวนหลักของ B หรืออาจกล่าวได้ว่า $a \in \mathcal{O}(\log A)$ และ $b \in \mathcal{O}(\log B)$

จัดแถวหุ่นยนต์

2 seconds, 256 megabytes

Abridged Problem Statement

มีชุดตัวเลข A ยาว N เรียงจากมากไปน้อย และมีชุดตัวเลขเป็นเซต B

เราต้องการนำ B ไปแทรกใน A โดยที่ทำให้จำนวนครั้งในการสลับน้อยสุด มีค่ามากที่สุดเท่าที่เป็นไปได้ ถามว่าจะใช้จำนวนครั้งในการสลับเท่าใด

Subtask 1

หาก $M = 1$ คำถามจะเป็น ควรแทรกตัวเลขตัวเดียวไว้ที่จุดไหน

Observation 1

สังเกตว่า การแทรกภายใน A ระหว่างจำนวนใด ๆ นั้นจะทำให้จำนวนครั้งที่ต้องสลับ ไม่มากเท่า การนำไปไว้ด้านหน้าสุดหรือหลังสุดของชุดตัวเลข A

เราจึงสามารถพิจารณาเพียงแค่ว่าจำนวนใหม่นั้นควรอยู่ด้านหน้าหรือด้านหลัง แล้วนับจำนวนครั้งในการสลับให้เรียง

Subtask 2

เนื่องจากมีจำนวนเก่าแค่ตัวเดียว นั่นเทียบเท่ากับว่าเราสามารถจัดวางอย่างไรก็ได้ วิธีการที่ทำให้จำนวนครั้งในการสลับสูงสุด คือนำชุดจำนวนรวมกันแล้ว เรียงจากน้อยไปมาก สังเกตว่าเราสามารถแทรกจำนวนใหม่ให้กลายเป็นแบบที่ต้องการได้เสมอ

Subtask 3

เราสามารถไล่หาวิธีการจัดเรียงทุกแบบที่เป็นไปได้ โดยการเขียน Recursive function เพื่อค้นหาทุกแบบ

Time Complexity: $\mathcal{O}\left(\frac{(N+M)!}{N!}\right)$

Subtask 4

Observation 2

จากข้อสังเกตแรก หากเราแบ่งการจัดวางของในเซต B ออกเป็น P_L กับ P_R แทนเซตของของที่วางไว้ด้านหน้าของตัวแรก กับเซตของของที่วางไว้หลังสุด เราสังเกตได้ไม่ยากว่าเราควรทำให้ P_L เรียงจากน้อยไปมาก และ P_R เรียงจากน้อยไปมากเช่นกัน เมื่อทำแล้ว เราจะสามารถนับจำนวนครั้งในการสลับได้ใน $\mathcal{O}(N^2)$ ด้วย Bubble Sort

Observation 3

การแบ่ง B ออกเป็น P_L กับ P_R นั้นก็ทำได้หลายวิธีเช่นกัน แต่สังเกตได้อีกว่าวิธีที่ดีที่สุดในการแบ่ง จะมี $\max(P_L) < \min(P_R)$ เสมอ เพราะหากมีค่า $x_L \in P_L$ กับ $x_R \in P_R$ ที่ $x_L > x_R$ จะได้ว่าการสลับ x_L กับ x_R ตั้งแต่แรกจะทำให้จำนวนครั้งในการสลับนั้นมากขึ้น

เราจึงสามารถไล่การแบ่ง P_L กับ P_R ได้ทั้งหมด $\mathcal{O}(N)$ วิธี

Time Complexity: $\mathcal{O}(M(N+M)^2)$

Subtask 5

Definition

นิยาม Inversion ของชุดตัวเลข A ซึ่งมีความยาว N เขียนแทนด้วยสัญลักษณ์ $Inv(A)$ ว่า ขนาดของเซตนิยามโดย

$$\{(i,j) \in \mathbb{Z}^2 | 1 \leq i < j \leq N \text{ and } A_i < A_j\}$$

Remark

นิยามดังกล่าวใช้เฉพาะกับข้อนี้ โดยปกติเราจะนิยาม Inversion อีกแบบดังนี้

นิยาม Inversion ของชุดตัวเลข A ซึ่งมีความยาว N ว่า ขนาดของเซตนิยามโดย

$$\{(i,j) \in \mathbb{Z}^2 | 1 \leq i < j \leq N \text{ and } A_i > A_j\}$$

แต่เนื่องจากจะทำให้ยุ่งยากต่อการอธิบายจึงทำการสลับข้างของการเปรียบเทียบ

Observation 4

จำนวนครั้งในการสลับที่น้อยที่สุด มีค่าเท่ากับจำนวน inversion

พิสูจน์. ก่อนอื่นจะพิสูจน์ว่าจำนวนครั้งในการสลับที่น้อยที่สุด $\leq$ จำนวน inversion

ตราบใดที่ชุดตัวเลขนั้นยังไม่ได้เรียงจากมากไปน้อย เราจะสังเกตได้ว่ามีอย่างน้อยบางตำแหน่ง i ที่ $A_i < A_{i+1}$ หากสลับตำแหน่ง i กับ $i+1$ จะได้ว่าจำนวน inversion จะลดลง 1 แต่เราทราบว่าชุดตัวเลขที่เรียงแล้วมีจำนวน inversion เป็น 0 เราจึงพิสูจน์ได้แล้วว่า จำนวนครั้งในการสลับที่น้อยที่สุด มีค่า *ไม่เกิน* จำนวน inversion (เพราะเราสามารถสลับตามอัลกอริทึมข้างต้นไปเท่ากับจำนวน inversion ครั้งแล้วจำนวน inversion จะเหลือศูนย์)

ต่อมาจะพิสูจน์ว่าจำนวนครั้งในการสลับที่น้อยที่สุด $\geq$ จำนวน inversion

จะพิสูจน์ด้วยข้อขัดแย้ง สมมติว่าจำนวนครั้งในการสลับที่น้อยที่สุด (เพื่อความง่ายจะเรียกค่านี้ว่า k) มีค่าน้อยกว่าจำนวน inversion นั่นคือ จากชุดตัวเลข A นั้นเราสามารถเรียงให้ inversion เหลือ 0 ได้โดย $k < Inv(A)$ สังเกตว่าการสลับตัวที่อยู่ติดกันภายใน A จะเพิ่มหรือลดจำนวน inversion ได้มากสุดแค่ 1 จึงได้ว่าการสลับของที่อยู่ติดกันไป k รอบ ไม่ว่าจะเป็นวิธีแบบใด จะได้จำนวน inversion ใหม่ (เพื่อความง่ายจะขอนิยามว่า $Inv(A')$) มีสมบัติดังนี้ $Inv(A) - k \leq Inv(A') \leq Inv(A) + k$ แต่เราทราบว่า $k < Inv(A)$ แสดงว่า $Inv(A) - k > 0$ กล่าวคือ $Inv(A') > 0$ นั่นคือไม่สามารถทำให้ inversion เหลือ 0 ได้ภายใน k รอบ □

ในปัญหาย่อยนี้ เราสามารถไล่การแบ่ง B ออกเป็น P_L และ P_R ทั้ง N แบบ แล้วนับจำนวน inversion ที่มีค่ามากที่สุด หากนับด้วยวิธีตรง ๆ จะใช้ $\mathcal{O}(N^2)$ เหมือนเดิม แต่เราสามารถใช้วิธี Divide and Conquer ในการนับ Inversion ได้ จะเหลือเพียง $\mathcal{O}(N \log N)$

Time Complexity: $\mathcal{O}(M(N+M)\log(N+M))$

Subtask 6

จากข้อสังเกต 4 เราสามารถทำอะไรได้มากกว่า การใช้วิธีจากปัญหาย่อย 3 แล้วหาเพียง inversion

สำหรับแต่ละตัวใน B เราจะพิจารณาทีละตัวแล้วดูว่าควรนำไปใส่ใน P_L หรือ P_R โดยก่อนอื่นสังเกตว่าจำนวน inversion ระหว่างข้างใน P_L, ระหว่างข้างใน P_R, ระหว่าง P_L กับ P_R สามารถทำให้มีค่ามากสุดได้โดยไม่ต้องสนใจ A ด้วยการเรียงของจากน้อยไปมาก ทั้งใน P_L และ P_R หลังจากนั้นเราจะสนใจแค่เพียง "ระหว่างการนำ x ไปใส่ P_L กับการนำ x ไปใส่ P_R" แบบไหนจะทำให้จำนวน inversion เพิ่มขึ้นมากกว่ากัน โดยจะอาศัยข้อสังเกตต่อไปดังนี้

Observation 5

ให้ m แทน มัธยฐาน (median) ของ A จะได้ว่า หาก $x > m$ ควรนำไปใส่ใน P_R แต่หาก $x < m$ ควรนำไปใส่ P_L

พิสูจน์. เราจะเปลี่ยนมุมมอง เป็นคำถามว่า หากนำ x ใส่ P_L จะเพิ่มค่า inversion เท่าไร และหากนำใส่ P_R จะเพิ่มค่าเท่าไรเช่นกัน

หากนำใส่ P_L จำนวน inversion ที่จะเพิ่มขึ้น คือจำนวนของใน A ที่มากกว่า x ส่วนหากนำใส่ P_R จำนวน inversion ที่จะเพิ่มขึ้นจะเป็นจำนวนของใน A ที่น้อยกว่า x

ในจุดนี้ จะเห็นได้ชัดว่าหากมีค่ามากกว่ามัธยฐาน เมื่อนำไปใส่ P_L จะได้จำนวน inversion มากกว่านำไปใส่ P_R และเป็นทำนองเดียวกันกรณีมีค่าน้อยกว่ามัธยฐาน □

Time Complexity: $\mathcal{O}(N + M \log N)$