Unit 31.

# Divide-and-Conquer

# Learning objectives

- ✓ ຜູ້ຮຽນຈະສາມາດເຂົ້າໃຈວິທີການ divide-and-conquer ແລະ ແກ້ບັນຫາທາງສາມແຈດ້ວຍວິທີນີ້.
- ✓ ນັກຮຽນຈະສາມາດເຂົ້າໃຈວ່າ ການຄົ້ນຫາແບບ binary, ການຈັດລຽງແບບ merge ແລະ ການຈັດລຽງແບບ quick ແມ່ນຂັ້ນຕອນວິທີທີ່ນຳໃຊ້ວິທີການ divide-and-conquer.
- ✓ ນັກຮຽນຈະສາມາດອອກແບບການເອີ້ນໃຊ້ຕົນເອງຊ້ຳ ແລະ ສ້າງຂັ້ນຕອນວິທີ divide-and-conquer ໄດ້.

## Learning overview

- ✓ ຮຽນຮູ້ວິທີແກ້ໄຂບັນຫາເສັ້ນທາງຮູບສາມແຈໂດຍນຳໃຊ້ວິທີການ divide-and-conquer.
- ✓ ຮຽນຮູ້ວິທີການອອກແບບວິທີການ divide-and-conquer ໂດຍຜ່ານການຄົ້ນຫາແບບ binary, ການຈັດລຽງແບບ merge ແລະ ການຈັດລຽງແບບ quick.
- ✓ ຮຽນຮູ້ວິທີແກ້ໄຂບັນຫາໂດຍໃຊ້ວິທີການ divide-and-conquer.

## Concepts you will need to know from previous units

- ✓ ສາມາດເຂົ້າໃຈ ແລະ ສ້າງຄຳນິຍາມຂອງຟັງຊັນ recursive ແລະ ເງື່ອນໄຂໃນການອອກຈາກ recursive.
- ✓ ສາມາດແກ້ໄຂບັນຫາການຄົ້ນຫາແບບ binary ໂດຍໃຊ້ loops ແລະ recursions.
- ✓ ສາມາດແກ້ໄຂບັນຫາການຈັດລຽງໄດ້ໂດຍໃຊ້ການຈັດລຽງແບບ merge ແລະ quick.

# Keywords

- Divide-and-Conquer
- Recursion
- Triangle Path Problem

# Mission

# 1. Real world problem

## 1.1. ການຄົ້ນຫາເສັ້ນທາງໃນເຂົາວົງກົດ

- ພິຈາລະນາບັນຫາຂອງການຊອກຫາເສັ້ນທາງໃນເຂົາວົງກົດທີ່ສະຫຼັບຊັບຊ້ອນ.
- ສົມມຸດວ່າເຂົາວົງກົດໄດ້ຖືກກຳໜົດໃຫ້ຢູ່ໃນ Array ສອງມິຕິ, ພຽງແຕ່ສາມາດເຄື່ອນຍ້າຍໄປໃນທິດທາງດ້ານຂວາ ແລະ ລົງດ້ານລຸ່ມ ແລະ ຕ້ອງໄດ້ຈ່າຍເງິນຈຳນວນໃດໜຶ່ງເມື່ອຜ່ານແຕ່ລະຫ້ອງ.
- ສາມາດຊອກຫາເສັ້ນທາງທີ່ມີຄ່າໃຊ້ຈ່າຍໜ້ອຍທີ່ສຸດໄດ້ແນວໃດ?

# 2. Mission

## 2.1. ເສັ້ນທາງທີ່ສັ້ນທີ່ສຸດໃນຮູບສາມແຈ

- ສົມມຸດວ່າໃນຮູບສາມແຈທີ່ໃຫ້ປະກອບດ້ວຍຕົວເລກຖ້ວນບວກ, ສາມາດເລີ່ມຕົ້ນຢູ່ເທິງສຸດຂອງສາມແຈ ແລະ ລົງໄປຫາແຕ່ລະແຖວໃນເວລາໃດໜຶ່ງ.
- ຖ້າການເຄື່ອນຍ້າຍສາມາດເຮັດໄດ້ພຽງແຕ່ໄປຫາຕົວເລກຊ້າຍຫຼືຂວາທີ່ຕິດກັນ, ຜົນບວກຂອງຈຳນວນເສັ້ນທາງທີ່ນ້ອຍທີ່ສຸດຈົນກ່ວາມັນໄປຮອດລຸ່ມສຸດແມ່ນເທົ່າໃດ?
- ຕົວຢ່າງ, ໃນສາມແຈຂ້າງລຸ່ມນີ້, ຜົນບວກລວມນ້ອຍທີ່ສຸດຂອງເສັ້ນທາງແມ່ນ $2 + 3 + 5 + 1 = 11$.

# 3. ການແກ້ໄຂບັນຫາ

## 3.1. ວິທີການຊອກຫາເສັ້ນທາງໃນຮູບສາມແຈເຮັດວຽກ

ໂປຣແກຣມທີ່ຊອກຫາຜົນບວກຂອງເສັ້ນທາງສັ້ນທີ່ສຸດໃນຮູບສາມແຈແມ່ນໂປຣແກຣມທີ່ຊອກຫາ ແລະ ສະແດງຜົນບວກຂອງເສັ້ນທາງທີ່ສັ້ນທີ່ສຸດໃນຮູບສາມແຈນີ້ດ້ວຍລາຍການຂອງຕົວເລກ, ດັ່ງທີ່ສະແດງຢູ່ລຸ່ມນີ້.

```
1  triangle = [
2      [2],
3      [3, 4],
4      [6, 5, 7],
5      [4, 1, 8, 3]
6  ]
7  minimum = find_minimum(0, 0, triangle)
8  print("The minimum cost is ", minimum)
```

```
The minimum cost is  11
```

# 3. ภามแก้ไขบัมหา

## 3.2. Code สุดท้ายในภามถิ้มหาเส้มหาๆในรูบสามเจ

```
1  def find_minimum(row, col, triangle):
2      if row == len(triangle):
3          return 0
4      else:
5          minimum = min(find_minimum(row + 1, col, triangle),
6                        find_minimum(row + 1, col + 1, triangle))
7          return triangle[row][col] + minimum
```

# Key concept

# 1. Divide-and-Conquer

## 1.1. ວິທີການ Divide-and-Conquer

- ยุดทะสาด divide-and-conquer แบ่งตัวอย่างข้อมูนป้อนเข้าของบันหาใดหนึ่งออกเป้นสอง ซึ่ ซายส่วน.
- ถ้าคำตอบของข้อมุนที่ป้อนเข้าที่ได้แบ่งออกยังบ่สามาดได้ธับทันที, ให้แบ่งมันออกเป้นส่วนที่มีขะหนาดน้อยลิงไปเลื้อยๆ.
- ถ้าได้ธับคำตอบของตัวอย่างข้มุนที่ป้อนเข้าที่ได้แบ่งออก, สามาดลอมคำตอบเซ่านี้เพื่อซอกหาคำตอบต้นสะบับของตัวอย่างข้มุนที่ป้อนเข้า.

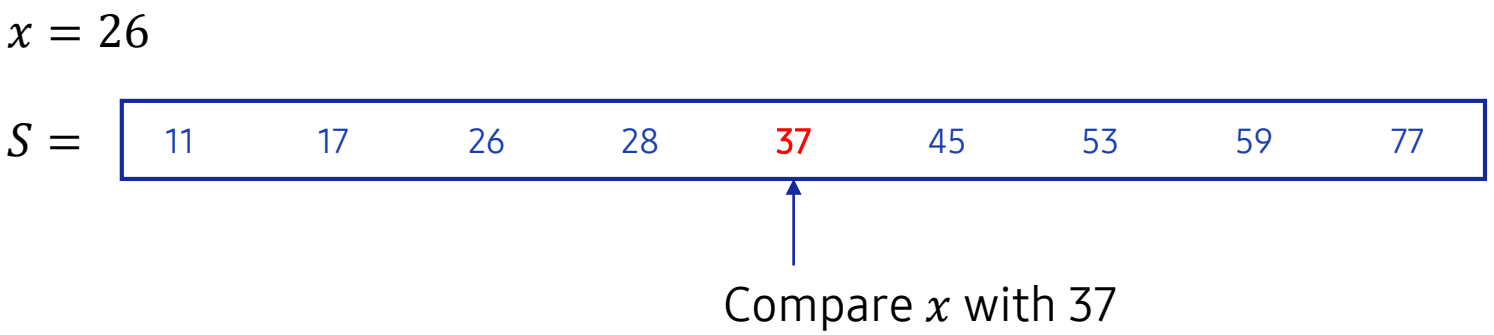
**Focus** ວິທີການ divide-and-conquer ແມ່ນວິທີການເພື່ອແກ້ໄຂບັນຫາໃນຂັ້ນຕອນຕໍ່ໄປນີ້

- ภามแบ่งส่วน: แบ่งตัวอย่างภามป้อนข้มูนของบันหาออกเป้นสอง ซึ่ ซายตัวอย่าง.
- แก้ไขบันหา: แก้ไขแต่ละตัวอย่างภามป้อนข้มูนที่แบ่งออก. ถ้าตัวอย่างที่แบ่งออกยังบ่น้อยพวงพ, ใช้ recursion เพื่อแบ่งออกไปอีกเลื้อยๆ.
- ลอมผืนได้ธับเข้ากัน: ถ้าจำเป้น, ซอกหาคำตอบของตัวอย่างภามป้อนข้มูนต้นสะบับโดยภามลอมเอาคำตอบของตัวอย่างภามป้อนข้มูนขะหนาดน้อย.

# 1. Divide-and-Conquer

## 1.2. ການຄົ້ນຫາແບບ Binary ແລະ Divide-and-Conquer

- ການຄົ້ນຫາແບບ binary ທີ່ໄດ້ສຶກສາຢູ່ໃນຫົວຂໍ້ທີ 25 ແມ່ນຕົວຢ່າງອັນໜຶ່ງຂອງ divide-and-conquer.
- ການຄົ້ນຫາແບບ Binary ໃຊ້ວິທີການ divide-and-conquer ເພື່ອຊອກຫາອົງປະກອບຢູ່ໃນລາຍການທີ່ຈັດລຽງແລ້ວ.
- ຕົວຢ່າງ, ຕ້ອງການຊອກຫາ x = 26 ເປັນການຄົ້ນຫາແບບ binary ໃນລາຍການ S ທີ່ມີ 9 ອົງປະກອບດັ່ງທີ່ສະແດງຂ້າງລຸ່ມນີ້.
- ເບື້ອງຕົ້ນ, ພວກເຮົາປຽບທຽບ x ກັບອົງປະກອບເຄິ່ງກາງຄື 37 ໃນລາຍການນີ້.



# 1. Divide-and-Conquer

## 1.2. ການຄົ້ນຫາແບບ Binary ແລະ Divide-and-Conquer

- ເນື່ອງຈາກ x ຫນ້ອຍກວ່າ 37, ຖ້າ x ມີຢູ່ໃນ S, ມັນມີຢູ່ໃນສ່ວນດ້ານຊ້າຍຂອງລາຍການນີ້, ດັ່ງນັ້ນເນື້ອທີ່ຄົ້ນຫາສາມາດຫຼຸດລົງເຄິ່ງຫນຶ່ງ.

$x = 26$

$S =$ | 11 | 17 | 26 | 28 | 37 | 45 | 53 | 59 | 77 |

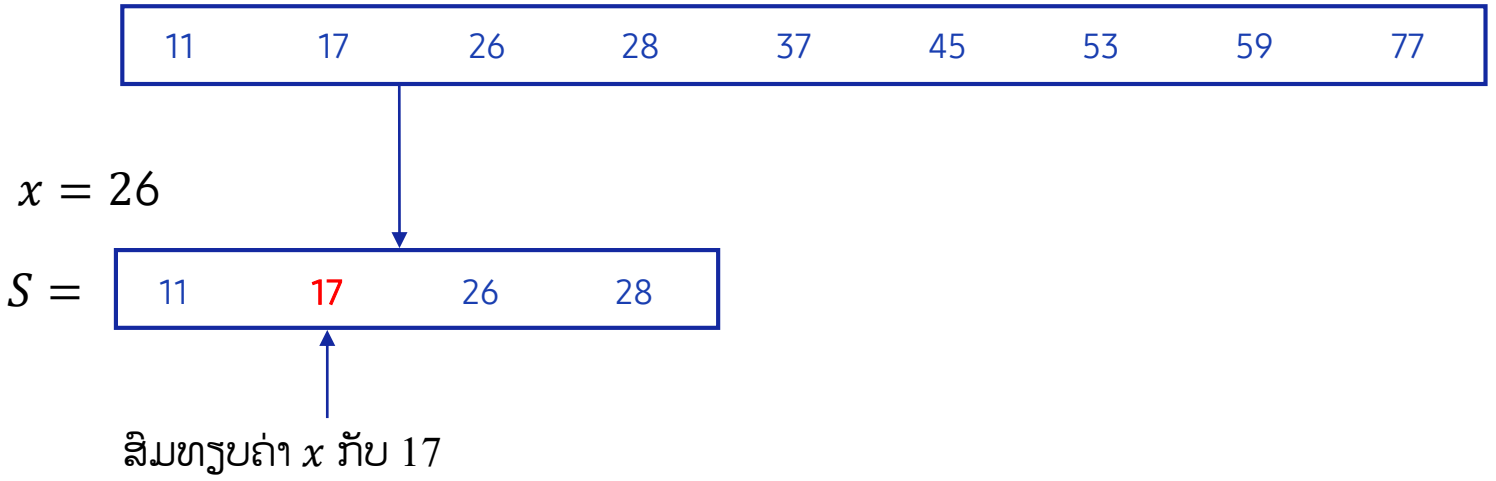
ເລືອກອາເຣ່ຍ່ອຍທາງດ້ານຊ້າຍ
ເພາະວ່າ $x < 37$

| 11 | 17 | 26 | 28 |

# 1. Divide-and-Conquer

## 1.2. ການຄົ້ນຫາແບບ Binary ແລະ Divide-and-Conquer

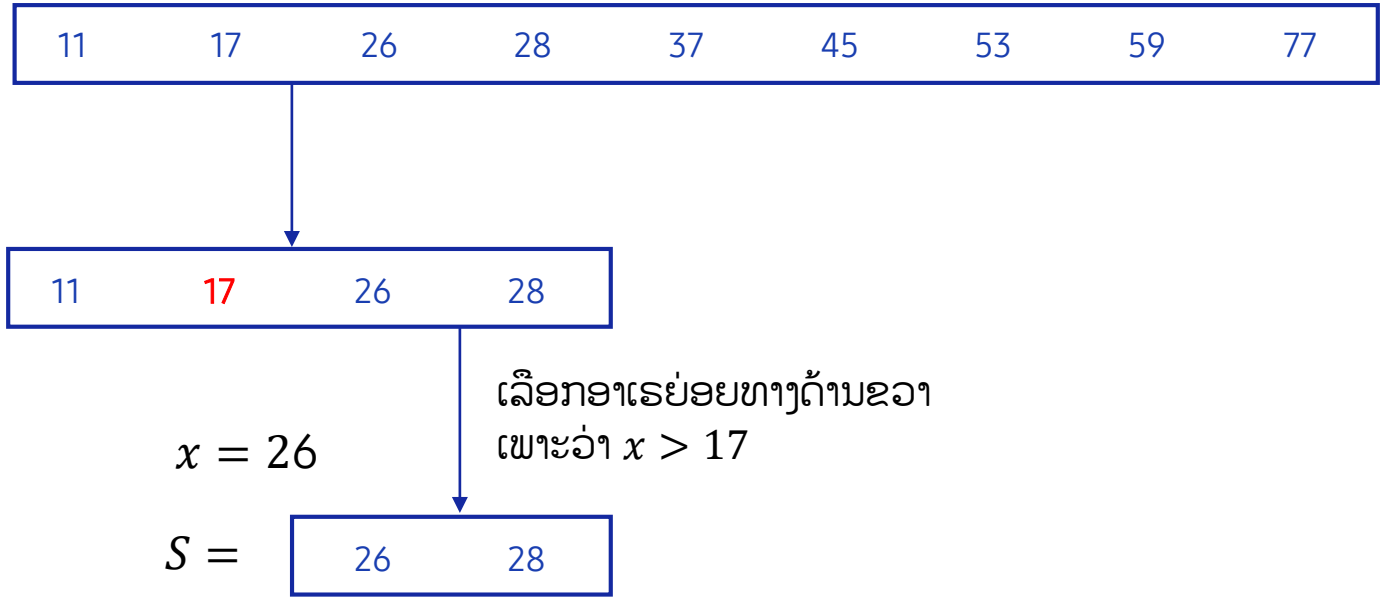
- ປຽບທຽບຄ່າຂອງອົງປະກອບກາງ 17 ແລະ x ອີກເທື່ອຫນຶ່ງໃນເນື້ອທີ່ຄົ້ນຫາທີ່ຫຼຸດລົງເຄິ່ງຫນຶ່ງ S.



# 1. Divide-and-Conquer

## 1.2. ການຄົ້ນຫາແບບ Binary ແລະ Divide-and-Conquer

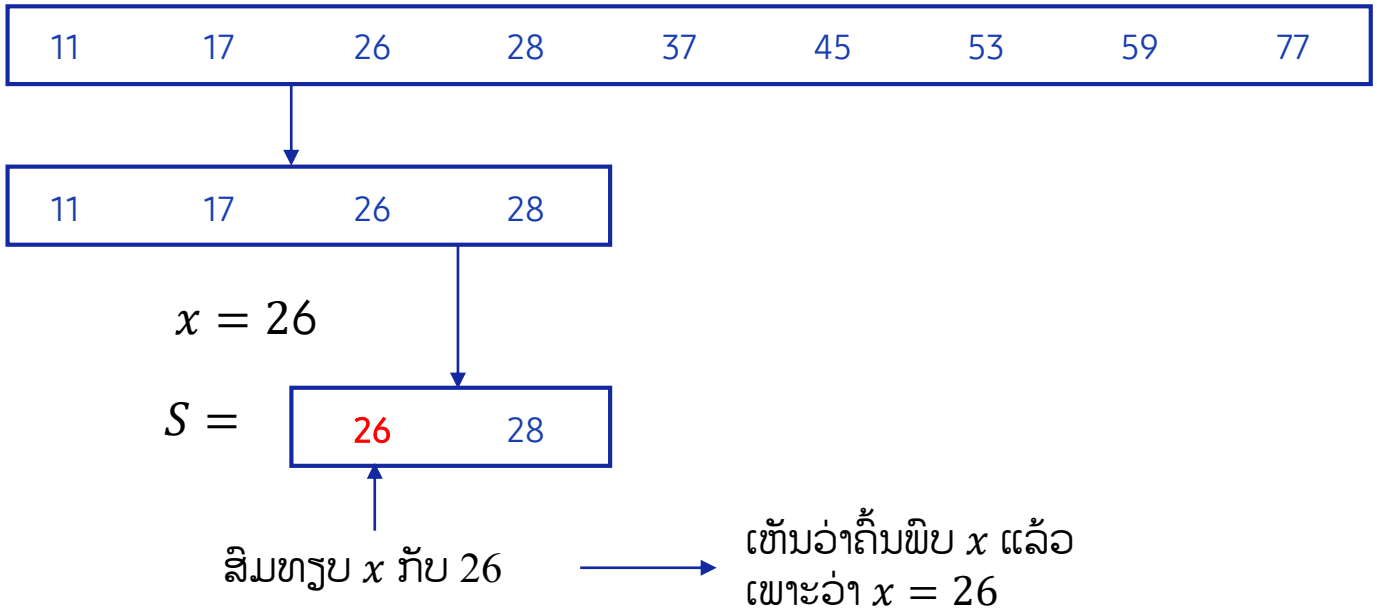
- ເນື່ອງຈາກ x ຫຼາຍກວ່າ 17, ຖ້າ x ມີຢູ່ໃນລາຍການນີ້, ມັນຈະມີຢູ່ໃນສ່ວນທາງດ້ານຂວາຂອງ S, ດັ່ງນັ້ນ ສາມາດຫຼຸດເນື້ອທີ່ຄົ້ນຫາລົງໄດ້ອີກ.



# 1. Divide-and-Conquer

## 1.2. ການຄົ້ນຫາແບບ Binary ແລະ Divide-and-Conquer

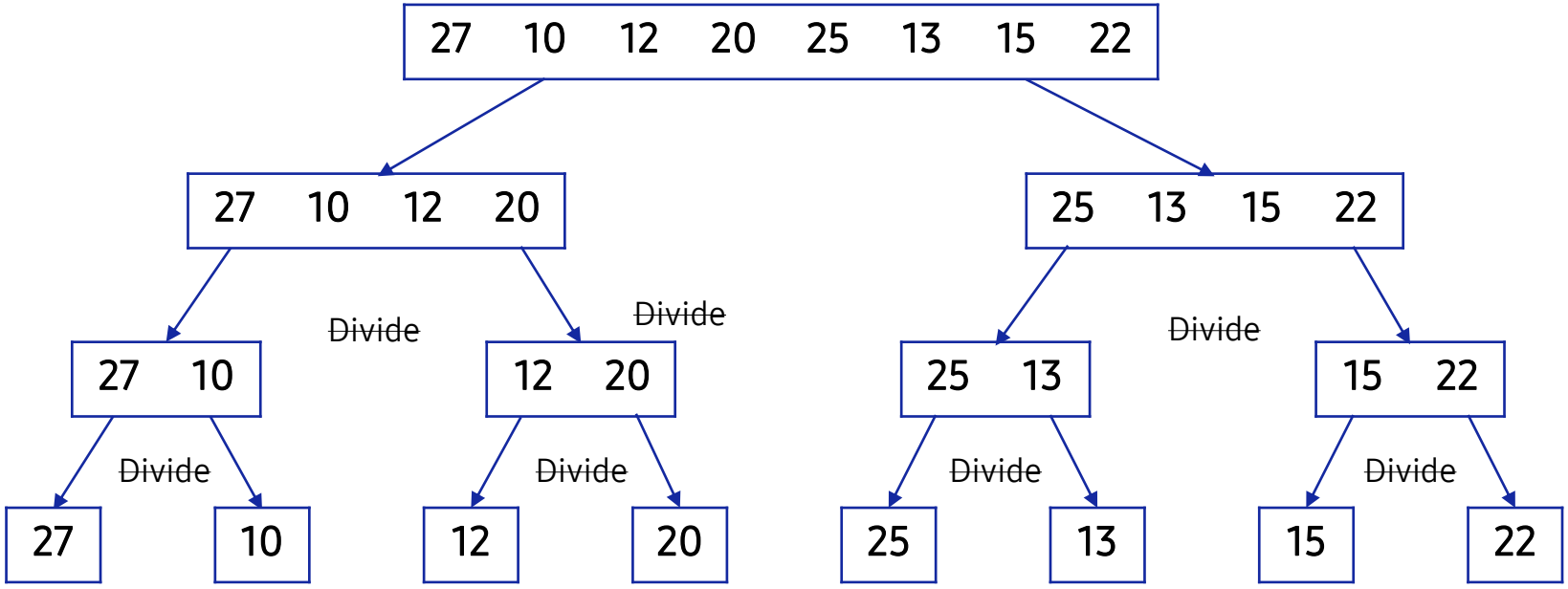
- ຖ້າອົງປະກອບກາງຂອງ S ເທົ່າກັບ x, ຈະຢຸດການຄົ້ນຫາ ສະແດງວ່າພົບຂໍ້ມູນທີ່ຕ້ອງການແລ້ວ.
- ຖ້າ S ສືບຕໍ່ຄົ້ນຫາແບບ binary ໄປເລື້ອຍໆ ຈົນກວ່າຄ່າ low ໃຫຍ່ກວ່າຫຼືເທົ່າກັບຄ່າ high ສະແດງວ່າບໍ່ມີ x ຢູ່ໃນລາຍການ S ໃຫ້ຢຸດການຄົ້ນຫາ.



# 1. Divide-and-Conquer

## 1.2. ການຈັດລຽງແບບ merge ແລະ Divide-and-Conquer

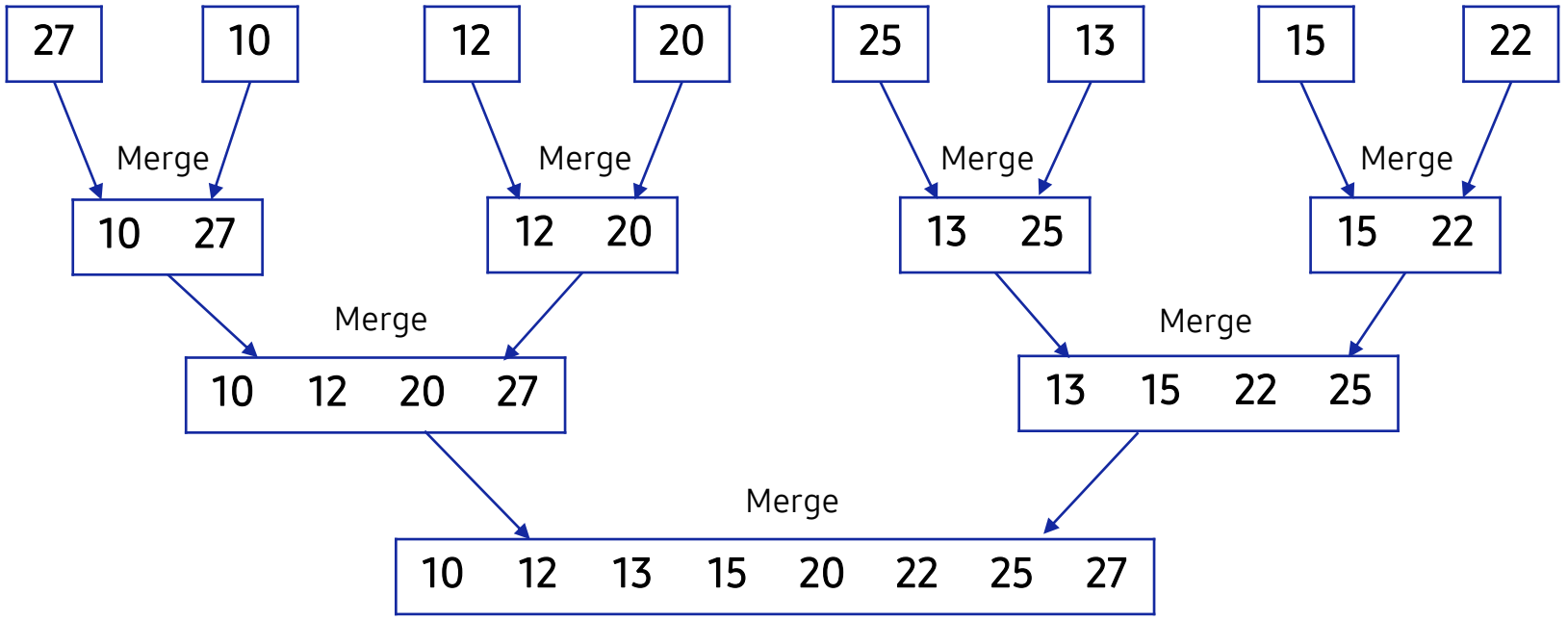
- ການຈັດລຽງແບບ merge ທີ່ສຶກສາຢູ່ໃນຫົວຂໍ້ທີ 28 ກໍ່ໃຊ້ divide-and-conquer.
- ໃນເວລານີ້, ເຫັນວ່າການຄົ້ນຫາແບບ binary ຈະແກ້ໄຂບັນຫາພຽງແຕ່ພາກສ່ວນຫນຶ່ງໃນບັນບັນດາພາກສ່ວນທີ່ແບ່ງອອກເປັນສອງສ່ວນ, ໃນຂະນະທີ່ການຈັດລຽງແບບ merge ແກ້ໄຂບັນຫາແຕ່ລະພາກສ່ວນທີ່ໄດ້ແບ່ງອອກ.



# 1. Divide-and-Conquer

## 1.2. ການຈັດລຽງແບບ merge ແລະ Divide-and-Conquer

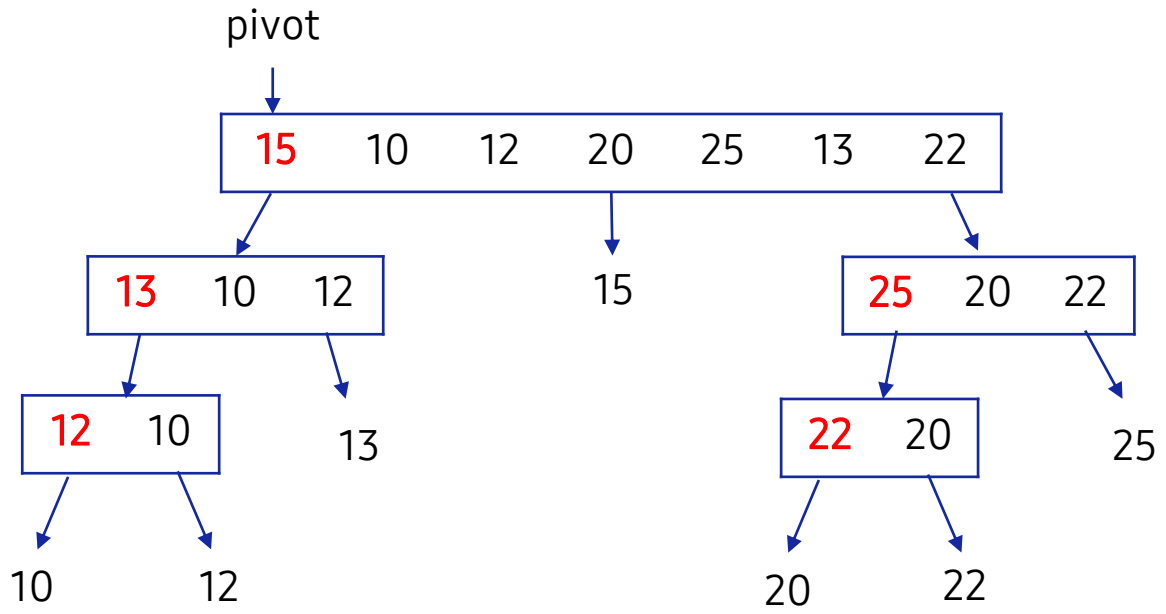
ນອກຈາກນັ້ນ, ໃຫ້ສັງເກດວ່າ ການຈັດລຽງແບບ merge ຈຳເປັນຕ້ອງມີຂັ້ນຕອນແຍກຕ່າງຫາກສຳລັບການລວມເອົາສອງລາຍການທີ່ຈັດລຽງຫຼັງຈາກແບ່ງບັນຫາອອກເປັນສອງສ່ວນຍ່ອຍທີ່ມີຂະໜາດເທົ່າກັນ ແລະ ຈັດລຽງພວກມັນ.



# 1. Divide-and-Conquer

## 1.2. ການຈັດລຽງແບບ Quick ແລະ Divide-and-Conquer

- ການຈັດລຽງແບບ Quick ທີ່ໄດ້ສຶກສາຢູ່ໃນຫົວຂໍ້ທີ 29, ກໍ່ໃຊ້ divide-and-conquer.
- ໃນຂະນະນີ້, ການຈັດລຽງແບບ quick ຈະແບ່ງບັນຫາທີ່ໃຫ້ອອກເປັນສອງກໍລະນີ, ແຕ່ຂະໜາດຂອງກໍລະນີທີ່ແບ່ງອອກທັງສອງແມ່ນບໍ່ເທົ່າກັນ.
- ນອກຈາກນັ້ນ, ໃຫ້ສັງເກດວ່າ ການຈັດລຽງແບບ quick ບໍ່ມີຂະບວນການລວມສອງລາຍການທີ່ຈັດລຽງ.



# Let’s code

# 1. Divide-and-Conquer

## 1.1. ການຄົ້ນຫາແບບ Binary ດ້ວຍການເຮັດຊ້ຳຄືນ

- ໃຫ້ເບິ່ງຄືນອີກເທື່ອຫນຶ່ງກ່ຽວກັບການສ້າງຟັງຊັນ bin_search() ໃນຫົວຂໍ້ 25.
- ຟັງຊັນນີ້ໄດ້ໃຊ້ວິທີການເພື່ອຫຼຸດຂອບເຂດຂອງການຄົ້ນຫາໂດຍໃຊ້ຕົວບົ່ງບອກ low ແລະ high.

```
1  def bin_search(nums, x):
2      low, high = 0, len(nums) - 1
3      while low <= high:
4          mid = (low + high) // 2
5          if nums[mid] == x:
6              return mid
7          elif nums[mid] > x:
8              high = mid - 1
9          else:
10             low = mid + 1
11     return -1
```

### Line 1-11

- ຂັ້ນຕອນວິທີ ຄົ້ນຫາແບບ binary ໂດຍໃຊ້ປະໂຫຍດການເຮັດຊ້ຳຄືນຊ່ວຍຫຼຸດຂອບເຂດຂອງການຄົ້ນຫາ ໂດຍໃຊ້ຕົວບົ່ງບອກ low ແລະ high.

# 1. Divide-and-Conquer

## 1.2. ການຄົ້ນຫາແບບ Binary ດ້ວຍການໃຊ້ Divide-and-Conquer

ບັນຫາ binary search ຖືກສ້າງແບບເອີ້ນໃຊ້ຕົວເອງແບບຊ້ຳຄືນ, ດັ່ງທີ່ສະແດງຂ້າງລຸ່ມນີ້, ໂດຍການນຳໃຊ້ວິທີການ divide-and-conquer.

```
1  def bin_search2(nums, x, low, high):
2      if low > high:
3          return -1
4      else:
5          mid = (low + high) // 2
6          if nums[mid] == x:
7              return mid
8          elif nums[mid] > x:
9              return bin_search2(nums, x, low, mid - 1)
10         else:
11             return bin_search2(nums, x, mid + 1, high)
```

Line 2-3

- ຟັງຊັນ bin_search2() ຊອກຫາຄ່າ x ໃນ nums ແລະ ສົ່ງຄືນຄ່າດັດຊະນີທີ່ສອດຄ່ອງກັນ.
- ເງື່ອນໄຂການສິ້ນສຸດຂອງຟັງຊັນ recursive ແມ່ນໃນເວລາທີ່ຄ່າຂອງ low ຫຼາຍກ່ວາຄ່າຂອງ high.
- ໃນກໍລະນີນີ້, ເນື່ອງຈາກບໍ່ພົບ x ຢູ່ໃນ S, ໃຫ້ສົ່ງຄືນຄ່າ -1.

# 1. Divide-and-Conquer

## 1.2. ການຄົ້ນຫາແບບ Binary ດ້ວຍການໃຊ້ Divide-and-Conquer

```
1  def bin_search2(nums, x, low, high):
2      if low > high:
3          return -1
4      else:
5          mid = (low + high) // 2
6          if nums[mid] == x:
7              return mid
8          elif nums[mid] > x:
9              return bin_search2(nums, x, low, mid - 1)
10         else:
11             return bin_search2(nums, x, mid + 1, high)
```

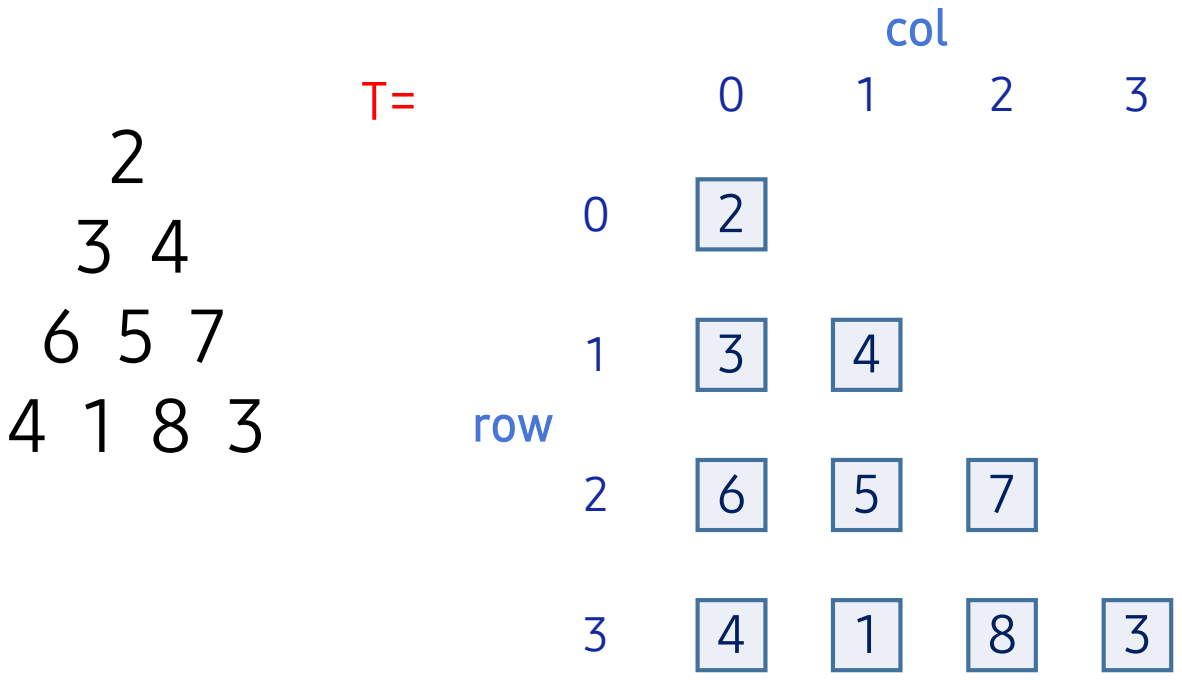
Line 2-3

- mid ແມ່ນອົງປະກອບເຄິ່ງກາງຂອງຊ່ວງການຄົ້ນຫາ ແລະ ສົ່ງຄືນຄ່າດັດຊະນີກາງຖ້າ nums[mid] ເທົ່າກັບຄ່າ ຂອງ x.
- ຖ້າ x ໜ້ອຍກວ່າ nums[mid], ໃຫ້ຄົ້ນຫາໃນຊ່ວງ [low, mid -1] ໂດຍການເຮັດຊ້ຳຄືນຂັ້ນຕອນເດີມ.
- ຖ້າ x ຫຼາຍກວ່າ nums[mid], ໃຫ້ຊອກຫາຊ່ວງ [mid + 1, high] ໂດຍການເຮັດຊ້ຳຄືນຂັ້ນຕອນເດີມ.

# 2. ບັນຫາເສັ້ນທາງໃນຮູບສາມແຈ

## 2.1. ຄົ້ນຫາເສັ້ນທາງທີ່ສັ້ນທີ່ສຸດໃນຮູບສາມແຈ ໂດຍໃຊ້ Divide-and-Conquer

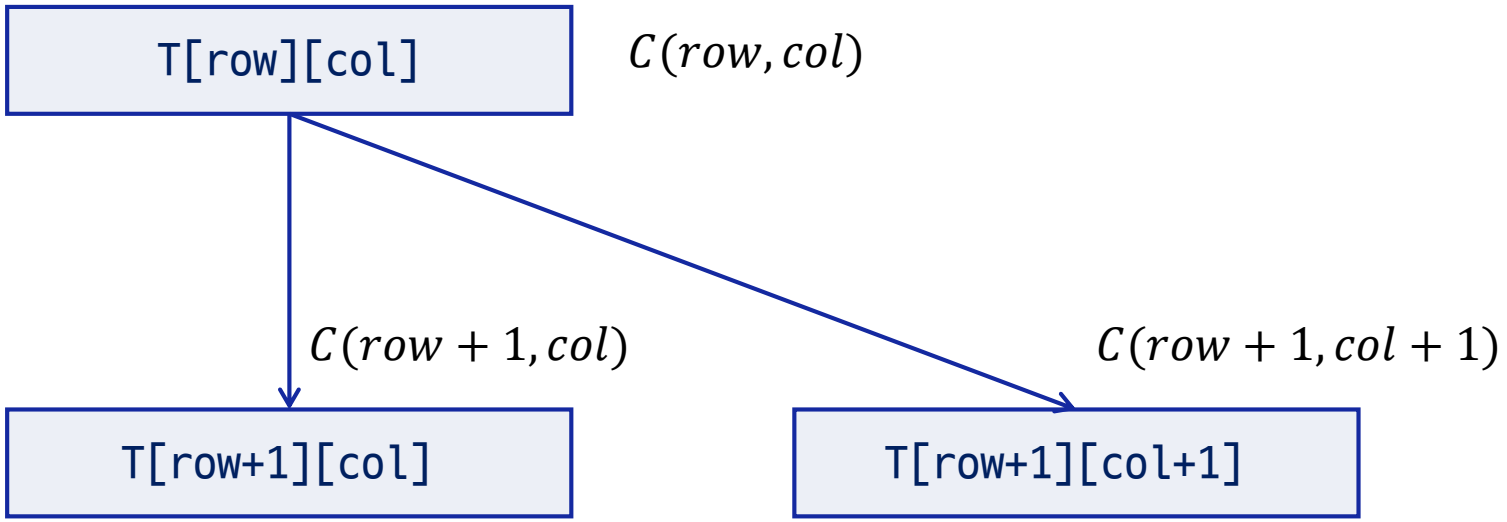
- ໃຫ້ແກ້ໄຂບັນຫາການຄົ້ນຫາເສັ້ນທາງທີ່ສັ້ນທີ່ສຸດຂອງການເດີນທາງໃນຮູບສາມແຈ ໂດຍໃຊ້ Divide-and-Conquer.
- ກ່ອນອື່ນໝົດ, ຮູບສາມແຈທີ່ໃຫ້ສາມາດສະແດງອອກໃນຮູບແບບຂອງລາຍການຊື່ T ດັ່ງທີ່ສະແດງຢູ່ໃນຮູບດ້ານຂວາຕໍ່ໄປນີ້.



# 2. ບັນຫາເສັ້ນທາງໃນຮູບສາມແຈ

## 2.1. ຄົ້ນຫາເສັ້ນທາງທີ່ສັ້ນທີ່ສຸດໃນຮູບສາມແຈ ໂດຍໃຊ້ Divide-and-Conquer

ໃຫ້ C (row, col) ເປັນຜົນບວກຂອງເສັ້ນທາງນ້ອຍທີ່ສຸດຈາກແໜ່ງຂອງ row ແລະ col ໃນລາຍການ T ຂອງ list.



# 2. ບັນຫາເສັ້ນທາງໃນຮູບສາມແຈ

## 2.1. ຄົ້ນຫາເສັ້ນທາງທີ່ສັ້ນທີ່ສຸດໃນຮູບສາມແຈ ໂດຍໃຊ້ Divide-and-Conquer

- ຜົນບວກຂອງເສັ້ນທາງທີ່ນ້ອຍທີ່ສຸດ C(row, col) ມີການພົວພັນແບບ recursive ດັ່ງຕໍ່ໄປນີ້.
  - ຖ້າຄ່າຂອງ row ເທົ່າກັບ n, ຜົນບວກຂອງເສັ້ນທາງແມ່ນສູນເພາະວ່າມັນມາຮອດລຸ່ມສຸດຂອງຮູບສາມແຈ.
  - ຖ້າຄ່າຂອງ row ບໍ່ເທົ່າ n, ຕົວເລກ T[row][col] ຂອງທີ່ຢູ່ປັດຈຸບັນຈະເຮັດໃຫ້ໄດ້ຄ່າຄວາມຍາວສູງຂຶ້ນ.
  - ດັ່ງນັ້ນ, ເສັ້ນທາງທີ່ມີຄ່າຄວາມຍາວໜ້ອຍກວ່າອາດຈະຖືກເລືອກຈາກຜົນບວກຂອງເສັ້ນທາງນ້ອຍສຸດໃນຕຳແໜ່ງດ້ານລຸ່ມ ແລະ ລຸ່ມດ້ານຂວາຂອງຕຳແໜ່ງປະຈຸບັນ.

$$C(row, col) = T[row][col] + \min(C(row+1, col), C(row+1, col+1))$$

$$C(n, col) = 0$$

# 2. ບັນຫາເສັ້ນທາງໃນຮູບສາມແຈ

## 2.2. ການສ້າງ ແລະ ຂຽນໂປຣແກຣມ

ການນຳໃຊ້ການພິວພັນແບບ recursive ທີ່ໄດ້ວິເຄາະຂ້າງເທິງ, ຟັງຊັນ find_minimum() ທີ່ການເອີ້ນໃຊ້ຕົວເອງຊ້ຳຄືນໄດ້ຖືກໃຊ້ງານດັ່ງທີ່ສະແດງຂ້າງລຸ່ມນີ້.

```
1  def find_minimum(row, col, triangle):
2      if row == len(triangle):
3          return 0
4      else:
5          minimum = min(find_minimum(row + 1, col, triangle),
6                        find_minimum(row + 1, col + 1, triangle))
7          return triangle[row][col] + minimum
```

### Line 1-3

- ພາລາມິເຕີການປ້ອນຂໍ້ມູນຂອງຟັງຊັນ find_minimum() ແມ່ນລາຍການ 'triangle' ທີ່ຊີ້ບອກທີ່ຢູ່ປັດຈຸບັນຂອງ row, col ແລະ list.
- ເນື່ອງຈາກເງື່ອນໄຂການສິ້ນສຸດເຮັດວຽກຂອງຟັງຊັນ recursive ມາຮອດດ້ານລຸ່ມສຸດຂອງຮູບສາມແຈ, ຄ່າຂອງແຖວແລະຄວາມຍາວແມ່ນເທົ່າກັນ.
- ໃນກໍລະນີນີ້, ຈະສົ່ງຄືນຄ່າ 0 ເປັນຜົນບວກຂອງເສັ້ນທາງ.

# 2. ບັນຫາເສັ້ນທາງໃນຮູບສາມແຈ

## 2.2. ການສ້າງ ແລະ ຂຽນໂປຣແກຣມ

```
1  def find_minimum(row, col, triangle):
2      if row == len(triangle):
3          return 0
4      else:
5          minimum = min(find_minimum(row + 1, col, triangle),
6                        find_minimum(row + 1, col + 1, triangle))
7          return triangle[row][col] + minimum
```

### Line 4-6

- ຖ້າຍັງບໍ່ທັນມາຮອດທາງດ້ານລຸ່ມຂອງຮູບສາມແຈ, ຜົນບວກຂອງສອງເສັ້ນທາງແມ່ນຄຳນວນໂດຍການເອີ້ນໃຊ້ຕົວເອງເຮັດວຽກຄືນ.
- ໃນຂະນະນີ້, ຈະຕ້ອງເລືອກຄ່າທີ່ນ້ອຍກວ່າຈາກຜົນບວກຂອງເສັ້ນທາງເມື່ອເຄື່ອນຍ້າຍລົງໄປລຸ່ມສຸດ ແລະ ລຸ່ມສຸດດ້ານຂວາ.

# 2. ບັນຫາເສັ້ນທາງໃນຮູບສາມແຈ

## 2.2. ການສ້າງ ແລະ ຂຽນໂປຣແກຣມ

```
1  def find_minimum(row, col, triangle):
2      if row == len(triangle):
3          return 0
4      else:
5          minimum = min(find_minimum(row + 1, col, triangle),
6                        find_minimum(row + 1, col + 1, triangle))
7          return triangle[row][col] + minimum
```

### Line 7

- ໂດຍຜ່ານການເອີ້ນໃຊ້ຕົວເອງ, ຄ່າທີ່ນ້ອຍກວ່າໄດ້ຖືກເລືອກເປັນຄ່ານ້ອຍສຸດລະຫວ່າງທາງລຸ່ມ ຫຼື ລຸ່ມຂວາ.
- ຜົນບວກຂອງເສັ້ນທາງນ້ອຍສຸດໃນຕຳແໜ່ງປະຈຸບັນແມ່ນຄ່າທີ່ເພີ່ມຕື່ມ triangle [row][col] ຂອງ ເສັ້ນທາງປັດຈຸບັນໃຫ້ກັບຄ່ານ້ອຍສຸດນີ້.

# 2. ບັນຫາເສັ້ນທາງໃນຮູບສາມແຈ

## 2.2. ການສ້າງ ແລະ ຂຽນໂປຣແກຣມ

- ເມື່ອຕົວປ່ຽນ triangle ຖືກໃສ່ເຂົ້າໄປເປັນລາຍການໜຶ່ງຂອງລາຍການຕໍ່ໄປນີ້, ຕຳແໜ່ງເທິງຈະເປັນ triangle[0][0], ດັ່ງນັ້ນຄ່າຂອງ row ແລະ col ແມ່ນ 0, 0 ຕາມລຳດັບ, ແລະ ຟັງຊັນ find_minimum() ຈະຖືກເອີ້ນໃຊ້.

```
1 triangle = [
2     [2],
3     [3, 4],
4     [6, 5, 7],
5     [4, 1, 8, 3]
6 ]
7 minimum = find_minimum(0, 0, triangle)
8 print("The minimum cost is ", minimum)
```

```
The minimum cost is  11
```

# Pop quiz

Q1. ສົມມຸດວ່າມີແປດຫຼຽນທີ່ມີລັກສະນະຄືກັນນັບຈາກ 1 ຫາ 8. ໃນຈຳນວນນີ້, ມີພຽງຫຼຽນດຽວທີ່ໜັກກວ່າອີກຫຼຽນໜຶ່ງ. ຖ້າທ່ານປະເມີນນ້ຳໜັກຂອງຫຼຽນດ້ວຍແຂນ, ໃຫ້ອອກແບບຂັ້ນຕອນວິທີເພື່ອເລືອກເອົາຫນຶ່ງຫຼຽນທີ່ຫນັກກວ່າ. ຢ່າງໜ້ອຍຕ້ອງຈັບຫຼຽນຂຶ້ນມາປະເມີນຈັກເທື່ອເພື່ອແຍກເອົາຫຼຽນອອກມາ?

1 2 3 4

5 6 7 8

Q2. ໃນຄຳຖາມທີ່ຜ່ານມາ, ສົມມຸດວ່າມີເກົ້າຫຼຽນທີ່ມີລັກສະນະຄືກັນນັບຈາກ 1 ຫາ 9.
ໃນກໍລະນີນີ້, ອອກແບບຂັ້ນຕອນວິທີເພື່ອເລືອກເອົາຫຼຽນທີ່ຫນັກກວ່າ.
ຢ່າງໜ້ອຍຕ້ອງຈັບຫຼຽນຂຶ້ນມາປະເມີນຈັກເທື່ອເພື່ອແຍກເອົາຫຼຽນອອກມາ?

1 2 3

4 5 6

7 8 9

# Pair programming

# Pair Programming Practice

**ແນວທາງ, ກົນໄກ ແລະ ແຜນສຸກເສີນ**

ການກະກຽມການສ້າງໂປຣແກຣມຮ່ວມກັນເປັນຄູ່ກ່ຽວຂ້ອງກັບການໃຫ້ຄຳແນະນຳແລະກົນໄກເພື່ອຊ່ວຍໃຫ້ນັກຮຽນຈັບຄູ່ຢ່າງຖືກຕ້ອງແລະ ໃຫ້ເຂົາເຈົ້າເຮັດວຽກເປັນຄູ່. ຕົວຢ່າງ, ນັກຮຽນຄວນປ່ຽນ "ເຮັດ." ການກະກຽມທີ່ມີປະສິດຕິຜົນຕ້ອງໃຫ້ມີແຜນການສຸກເສີນໃນກໍລະນີທີ່ຄູ່ຮ່ວມງານຫນຶ່ງບໍ່ຢູ່ຫຼືຕັດສິນໃຈທີ່ຈະບໍ່ເຂົ້າຮ່ວມດ້ວຍເຫດຜົນໃດຫນຶ່ງ ຫຼືດ້ວຍເຫດຜົນອື່ນ. ໃນກໍລະນີເຫຼົ່ານີ້, ມັນເປັນສິ່ງສໍາຄັນທີ່ຈະເຮັດໃຫ້ມັນຊັດເຈນວ່ານັກຮຽນທີ່ມີປະຕິບັດໜ້າທີ່ຢ່າງຫ້າວຫັນຈະບໍ່ຖືກລົງໂທດ ຍ້ອນວ່າການຈັບຄູ່ບໍ່ໄດ້ຜົນດີ.

**ການຈັບຄູ່ທີ່ຄ້າຍຄືກັນ, ບໍ່ຈຳເປັນຕ້ອງເທົ່າທຽມກັນ, ຄວາມສາມາດເປັນຄູ່ຮ່ວມງານ**

ການຂຽນໂປຣແກຣມຄູ່ຈະມີປະສິດທິພາບເມື່ອນັກຮຽນຕັ້ງໃຈຮ່ວມກັນເຮັດວຽກ, ຊຶ່ງວ່າບໍ່ຈຳເປັນຕ້ອງມີຄວາມຮູ້ເທົ່າທຽມກັນ, ແຕ່ຕ້ອງມີຄວາມສາມາດເຮັດວຽກເປັນຄູ່ຮ່ວມງານ. ການຈັບຄູ່ນັກຮຽນທີ່ບໍ່ສາມາດເຂົ້າກັນໄດ້ມັກຈະເຮັດໃຫ້ການມີສ່ວນຮ່ວມທີ່ບໍ່ສົມດຸນກັນ. ຄູສອນຕ້ອງເນັ້ນຫນັກວ່າການຂຽນໂປຣແກຣມຄູ່ບໍ່ແມ່ນຍຸດທະສາດ “divide-and-conque”, ແຕ່ຈະເປັນຄວາມພະຍາຍາມຮ່ວມມືເຮັດວຽກທີ່ແທ້ຈິງໃນທຸກໆດ້ານສໍາລັບໂຄງການທັງຫມົດ. ຄູຄວນຫຼີກເວັ້ນການຈັບຄູ່ນັກຮຽນທີ່ອ່ອນຫຼາຍກັບນັກຮຽນທີ່ເກັ່ງຫຼາຍ.

**ກະຕຸ້ນນັກຮຽນໂດຍການໃຫ້ສິ່ງຈູງໃຈພິເສດ**

ການສະເໜີແຮງຈູງໃຈພິເສດສາມາດຊ່ວຍກະຕຸ້ນນັກຮຽນໃຫ້ຈັບຄູ່, ໂດຍສະເພາະກັບນັກຮຽນທີ່ມີຄວາມສາມາດຫຼາຍ. ຈະເຫັນວ່າມັນເປັນປະໂຫຍດທີ່ຈະໃຫ້ນັກຮຽນຈັບຄູ່ເຮັດວຽກຮ່ວມກັນພຽງແຕ່ຫນຶ່ງຫຼືສອງວຽກເທົ່ານັ້ນ.

# Pair Programming Practice

**ປ້ອງກັນການໂກງໃນການເຮັດວຽກຮ່ວມກັນ**

ສິ່ງທ້າທາຍສຳລັບຄູແມ່ນເພື່ອຊອກຫາວິທີທີ່ຈະປະເມີນຜົນໄດ້ຮັບຂອງບຸກຄົນ, ໃນຂະນະທີ່ນຳໃຊ້ຜົນປະໂຫຍດຂອງການຮ່ວມມື. ຈະຮູ້ໄດ້ແນວໃດວ່ານັກຮຽນຕັ້ງໃຈເຮັດວຽກ ຫຼື ກົ້ງແຮງງານຜູ້ຮ່ວມງານ? ຜູ້ຊ່ຽວຊານແນະນຳໃຫ້ທົບທວນຄືນການອອກແບບຫຼັກສູດ ແລະ ການປະເມີນ ພ້ອມທັງປຶກສາຫາລືຢ່າງຈະແຈ້ງ ແລະ ຊັດເຈນກ່ຽວກັບພຶດຕິກຳຂອງນັກຮຽນທີ່ຈະຖືກຕີຄວາມວ່າຂີ້ຕົວະ. ຜູ້ຊ່ຽວຊານເນັ້ນໜັກໃຫ້ຄູເຮັດການມອບໝາຍໃຫ້ມີຄວາມໝາຍຕໍ່ນັກຮຽນ ແລະ ອະທິບາຍຄຸນຄ່າຂອງສິ່ງທີ່ນັກຮຽນຈະຮຽນຮູ້ໂດຍການເຮັດສຳເລັດ.

**ສະພາບແວດລ້ອມການຮຽນຮູ້ໃນການຮ່ວມມື**

ສະພາບແວດລ້ອມການຮຽນຮູ້ໃນຮ່ວມກັນເກີດຂຶ້ນໄດ້ທຸກເວລາທີ່ຜູ້ສອນຮຽກຮ້ອງໃຫ້ນັກຮຽນເຮັດວຽກຮ່ວມກັນໃນກິດຈະກຳການຮຽນຮູ້. ສະພາບແວດລ້ອມການຮຽນຮູ້ຮ່ວມກັນສາມາດມີສ່ວນຮ່ວມທັງກິດຈະກຳທີ່ເປັນທາງການ ແລະ ບໍ່ເປັນທາງການ ແລະ ອາດຈະບໍ່ລວມເຖິງການປະເມີນໂດຍກົງ. ຕົວຢ່າງ, ນັກສຶກສາຄູ່ເຮັດວຽກມອບໝາຍຮ່ວມກັນໃນການຂຽນໂປຣແກຣມ; ນັກສຶກສາກຸ່ມນ້ອຍໆສົນທະນາຄຳຕອບທີ່ເປັນໄປໄດ້ຕໍ່ກັບຄຳຖາມຂອງອາຈານໃນລະຫວ່າງການບັນຍາຍ; ແລະ ນັກຮຽນເຮັດວຽກຮ່ວມກັນນອກຫ້ອງຮຽນເພື່ອຮຽນຮູ້ແນວຄວາມຄິດໃໝ່. ການຮຽນຮູ້ການຮ່ວມມືແມ່ນແຕກຕ່າງຈາກໂຄງການທີ່ນັກຮຽນ “divide and conquer.” ເມື່ອນັກຮຽນແບ່ງວຽກກັນ, ແຕ່ລະຄົນຮັບຜິດຊອບພຽງແຕ່ສ່ວນໜຶ່ງຂອງການແກ້ໄຂບັນຫາ ແລະ ຈະບໍ່ຄ່ອຍມີບັນຫາຫຍັງໃນການເຮັດວຽກຮ່ວມກັບຄົນອື່ນໃນທີມ. ໃນສະພາບແວດລ້ອມການເຮັດວຽກຮ່ວມກັນ, ນັກຮຽນມີສ່ວນຮ່ວມໃນການສົນທະນາປຶກສາຫາລືເຊິ່ງກັນແລະກັນ.

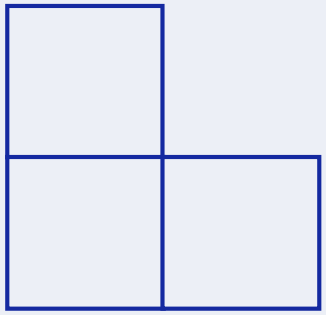
## Q1. ອອກແບບຂັ້ນຕອນວິທີ divide-and-conquer ທີ່ແກ້ໄຂບັນຫາ Tromino.

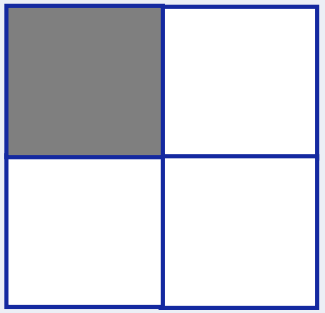
- ສາມຮູບຈະຕຸລັດທີ່ຕິດກັນເອີ້ນວ່າ tromino.
- ມີກະດານ checkerboard ທີ່ມີ m ຮູບຈະຕຸລັດເຊື່ອມຕໍ່ກັນຕາມແນວນອນແລະແນວຕັ້ງ ແລະ ຊ່ອງຫນຶ່ງແມ່ນຖືກຫມາຍດ້ວຍ X.
- ໃນທີ່ນີ້, ພວກເຮົາສົມມຸດວ່າ m ແມ່ນກໍາລັງຂອງ 2.
- ພວກເຮົາຕ້ອງການຕື່ມໃສ່ກະດານ checkerboard ດ້ວຍ tromino ເພື່ອຕອບສະຫນອງເງື່ອນໄຂຕໍ່ໄປນີ້.
  - ກະດານ checkerboard ທັງຫມົດຄວນຖືກເຕີມເຕັມໄປດ້ວຍ tromino.
  - ຖັນທີ່ມີເຄື່ອງໝາຍ X ແມ່ນບໍ່ສາມາດເອົາ tromino ເຂົ້າໄປໃສ່ໄດ້.
  - Trominoes ບໍ່ສາມາດທັບຊ້ອນກັນໄດ້.
  - Tromino ບໍ່ສາມາດເກີນອອກຈາກ checkerboard ໄດ້.

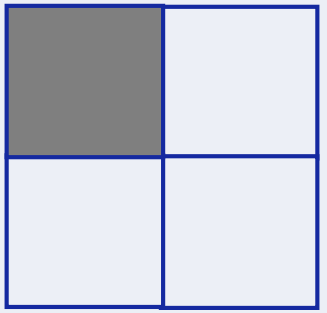
## Q1. ອອກຂັ້ນຕອນວິທີ divide-and-conquer ທີ່ແກ້ໄຂບັນຫາ Tromino.

**Hint**

- Tromino ປະກອບດ້ວຍສາມຮູບຈະຕຸລັດ. ໂດຍໃຫ້ກະດານຂະໜາດ N*N, ຊອກຫາວິທີທີ່ຈະຕື່ມໃສ່ຊ່ອງຫວ່າງທັງໝົດດ້ວຍ tromino.
- ໃນກໍລະນີນີ້, ສ່ວນຫນຶ່ງທີ່ X ຖືກໝາຍໄວ້ແມ່ນບໍ່ສາມາດເອົາ tromino ໄປໃສ່ໄດ້.



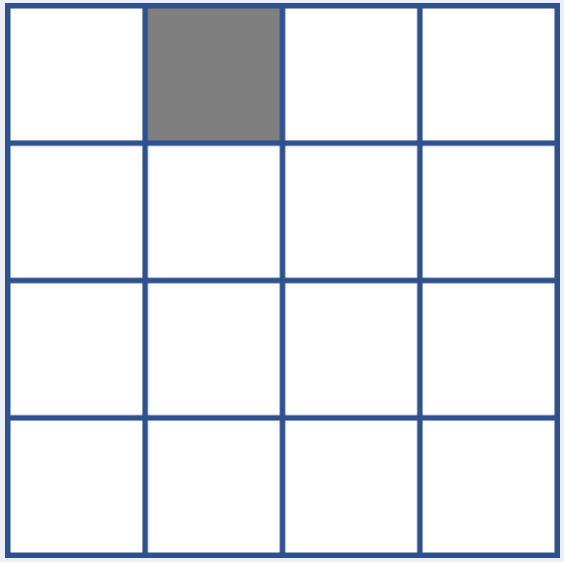


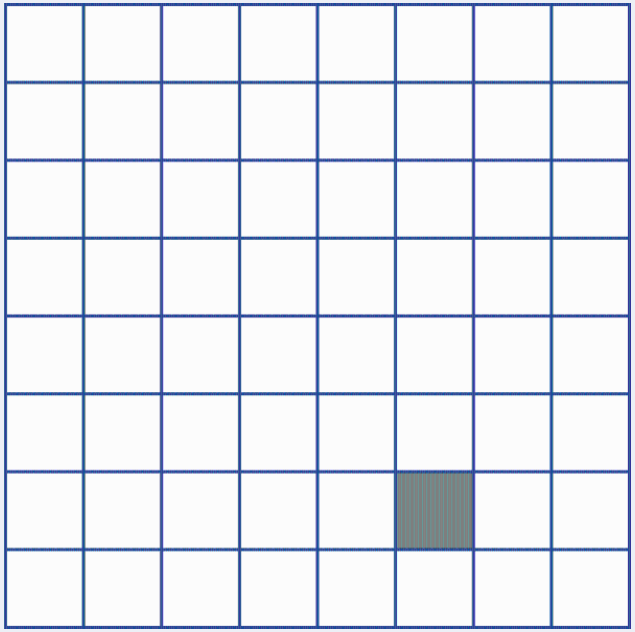


## Q1. ອອກແບບຂັ້ນຕອນວິທີ divide-and-conquer ທີ່ແກ້ໄຂບັນຫາ Tromino.

**Hint**

- ຕົວຢ່າງ, ພິຈາລະນາວິທີການປົກຄຸມຊ່ອງທັງຫມົດດ້ວຍ tromino ໃນກະດານ 4 * 4 ແລະ ກະດານ 8 * 8.
- ລອງຄິດວ່າ ຖ້າແບ່ງກະດານທີ່ໄດ້ຮັບເປັນສີ່ສ່ວນ.





## Q1. ອອກແບບຂັ້ນຕອນວິທີ divide-and-conquer ທີ່ແກ້ໄຂບັນຫາ Tromino.

**Hint**

ກະດານທີ່ໃຫ້ແມ່ນຖືກປົກຄຸມດ້ວຍ tromino ດັ່ງທີ່ສະແດງຂ້າງລຸ່ມນີ້.