

Unit 29.

ການຈັດລຽງແບບ Quick

● Learning objectives

- ✓ ເຂົ້າໃຈຂັ້ນຕອນວິທີການຈັດລຽງແບບ quick ແລະ ສາມາດແກ້ໄຂບັນຫາການຈັດລຽງໄດ້ໂດຍໃຊ້ການຈັດລຽງແບບ quick.
- ✓ ເຂົ້າໃຈ ແລະ ສາມາດອະທິບາຍຄວາມແຕກຕ່າງລະຫວ່າງການຈັດລຽງແບບ quick ແລະ ການຈັດລຽງແບບ merge.
- ✓ ເຂົ້າໃຈ ແລະ ສາມາດອະທິບາຍ ເວລາທີ່ໃຊ້ໃນການປະມວນຜົນຂອງການຈັດລຽງແບບ quick

● Learning overview

- ✓ ສ້າງການຈັດລຽງ quick ທີ່ແບ່ງລາຍການທີ່ບໍ່ໄດ້ຈັດລຽງໂດຍອີງຕາມ pivot ແລະ ຈັດລຽງພວກມັນເປັນແຕ່ລະສ່ວນ.
- ✓ ສ້າງຟັງຊັນແບ່ງສ່ວນທີ່ແບ່ງລາຍການທີ່ບໍ່ໄດ້ຈັດລຽງໂດຍອີງໃສ່ pivot ສໍາລັບການຈັດລຽງແບບ quick.
- ✓ ເຂົ້າໃຈວ່າ ການຈັດລຽງແບບ quick ກໍ່ເປັນວິທີການ divide-and-conquer ໂດຍໃຊ້ຟັງຊັນ recursive ເຊັ່ນດຽວກັນກັບຂັ້ນຕອນວິທີຈັດລຽງແບບ merge.

● Concepts you will need to know from previous units

- ✓ ການນໍາໃຊ້ຕົວປະຕິບັດການປຽບທຽບເພື່ອປຽບທຽບສອງຕົວເລກ.
- ✓ ການແບ່ງບັນຫາທີ່ໃຫ້ ແລະ ເອີ້ນໃຊ້ຟັງຊັນ recursive ຊໍ້າໆ.
- ✓ ການນໍາໃຊ້ຕໍາລາ Big O ເພື່ອວິເຄາະ ເວລາທີ່ໃຊ້ໃນການປະມວນຜົນຂອງ algorithm.

Keywords

Sorting Problem

Quick Sort

**Pivot and
Partitioning**

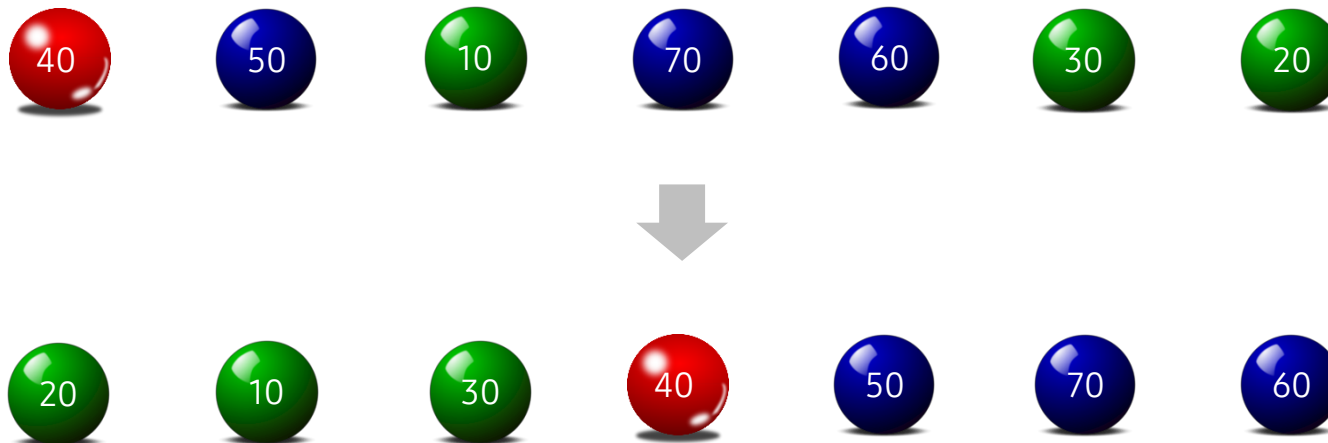
| Mission

1. Real world problem

1.1. ການລຽງລຳດັບກັບທີ່(In-Place Sorting)

| ໃນຫົວຂໍ້ທີ່ຜ່ານມາ, ພວກເຮົາໄດ້ສຶກສາຂັ້ນຕອນວິທີຈັດລຽງແບບ merge. ຢ່າງໃດກໍຕາມ, ເພື່ອນຳໃຊ້ການຈັດລຽງແບບ merge, ພວກເຮົາຕ້ອງໃຊ້ເນື້ອທີ່ຫນ່ວຍຄວາມຈຳເພີ່ມເທົ່າກັບຂະໜາດລາຍການທີ່ໃຫ້ມາ. ສາມາດອອກແບບຂັ້ນຕອນວິທີການຈັດລຽງລຳດັບທີ່ມີປະສິດທິພາບໂດຍບໍ່ໃຊ້ເນື້ອທີ່ຄວາມຈຳເພີ່ມເຕີມໄດ້ບໍ?

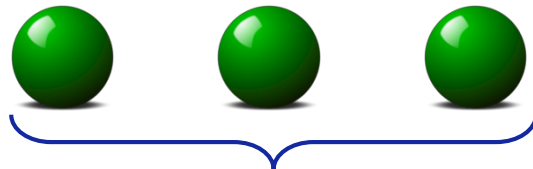
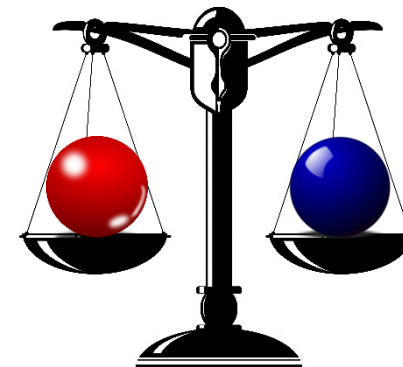
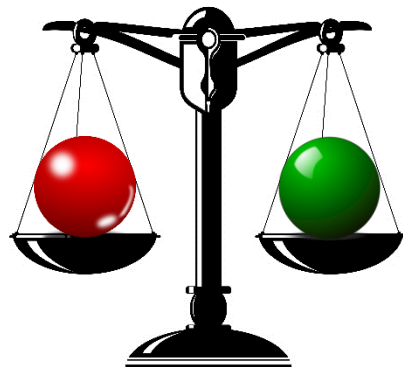
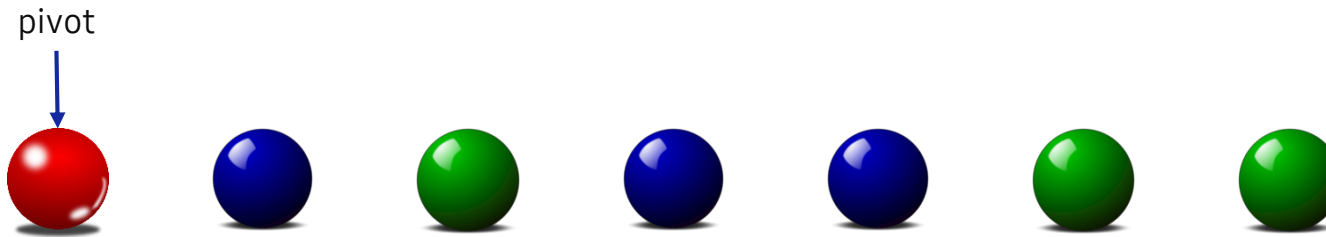
Ex ສົມມຸດວ່າມີຫນ່ວຍເຫຼັກ 7 ໜ່ວຍດັ່ງທີ່ສະແດງໃຫ້ເຫັນຂ້າງລຸ່ມນີ້. ໜ່ວຍເຫຼັກເຫຼົ່ານີ້ແມ່ນມີຮູບຮ່າງ ແລະ ຂະໜາດດຽວກັນ ແຕ່ມີນ້ຳໜັກທີ່ແຕກຕ່າງກັນ. ໂດຍພຽງແຕ່ປ່ຽນບ່ອນຫນ່ວຍເຫຼັກເຫຼົ່ານີ້, ໜ່ວຍເຫຼັກທີ່ເບົາກວ່າຫນ່ວຍເຫຼັກສີແດງໃຫ້ຍ້າຍໄປໄວ້ທາງດ້ານຊ້າຍ ແລະ ອັນທີ່ໜັກກວ່າ ໃຫ້ຍ້າຍໄປໄວ້ທາງດ້ານຂວາໄດ້ບໍ?



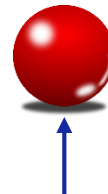
2. Mission

2.1. ການແບ່ງສ່ວນດ້ວຍ pivot

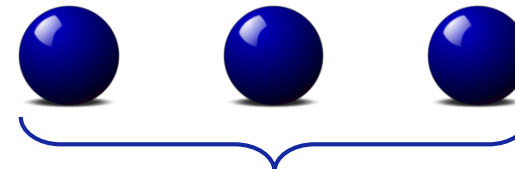
- | ນຳໃຊ້ຫຼັກການຄວາມສົມດຸນ. ຫຼັກການວັດແທກຄວາມສົມດຸນສາມາດດຳເນີນການດັ່ງຕໍ່ໄປນີ້:
 - ສາມາດປຽບທຽບນໍ້າໜັກໄດ້ໂດຍການເອົາໜ່ວຍເຫຼັກສອງໜ່ວຍອອກຈາກຖັງ.
 - ຕຳແໜ່ງຂອງໜ່ວຍເຫຼັກສອງໜ່ວຍສາມາດສັບປ່ຽນບ່ອນກັນ.
- | ພວກເຮົາມີ ໜ່ວຍເຫຼັກສີແດງເປັນຕົວອ້າງອີງ, ດັ່ງນັ້ນພວກເຮົາສາມາດຊອກຫາໜ່ວຍເຫຼັກທີ່ເບົາກວ່າ ແລະ ໜ່ວຍເຫຼັກທີ່ໜັກກວ່າ ໜ່ວຍສີແດງ ແລະ ສັບປ່ຽນພວກມັນ.
- | ດ້ວຍວິທີການນີ້, ຈຳນວນການປຽບທຽບຈຳນວນໜ່ວຍເຫຼັກທັງໝົດທີ່ຕ້ອງຖືກຈັດໃສ່ໃນບ່ອນຈັດລຽງ ຕາມນໍ້າໜັກແມ່ນເທົ່າໃດ?



ເບົາກວ່າ pivot



pivot



ໜັກກວ່າ pivot

| Key concept

1. ການຈັດລຽງແບບ Quick

1.1. ຕົວຢ່າງຂອງການຈັດລຽງແບບ quick

| ການຈັດລຽງແບບ Quick ແມ່ນຂັ້ນຕອນວິທີການຈັດລຽງທີ່ຈັດລຽງລາຍການ ທີ່ບໍ່ໄດ້ຈັດລຽງແບບຊ້າໆໂດຍການແບ່ງອອກເປັນສອງລາຍການຍ່ອຍ, ໂດຍອີງຕາມຄ່າຂອງ pivot.

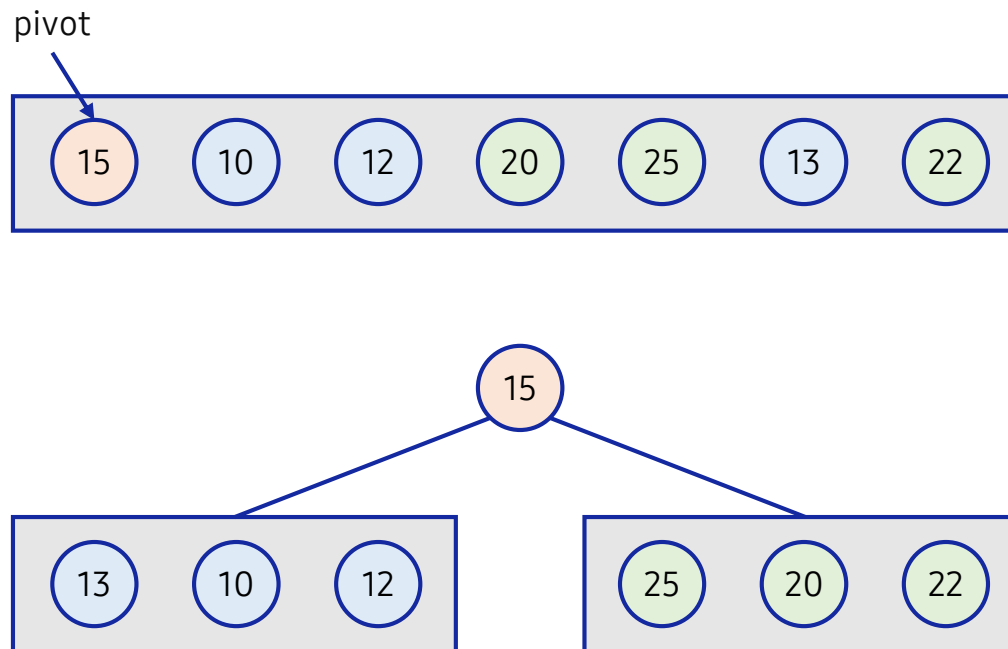
Ex ສົມມຸດວ່າພວກເຮົາຕ້ອງການທີ່ຈະຈັດລຽງລຳດັບ [15, 10, 12, 20, 25, 13, 22] ດັ່ງຕໍ່ໄປນີ້.



1. ການຈັດລຽງແບບ Quick

1.1. ຕົວຢ່າງຂອງການຈັດລຽງແບບ quick

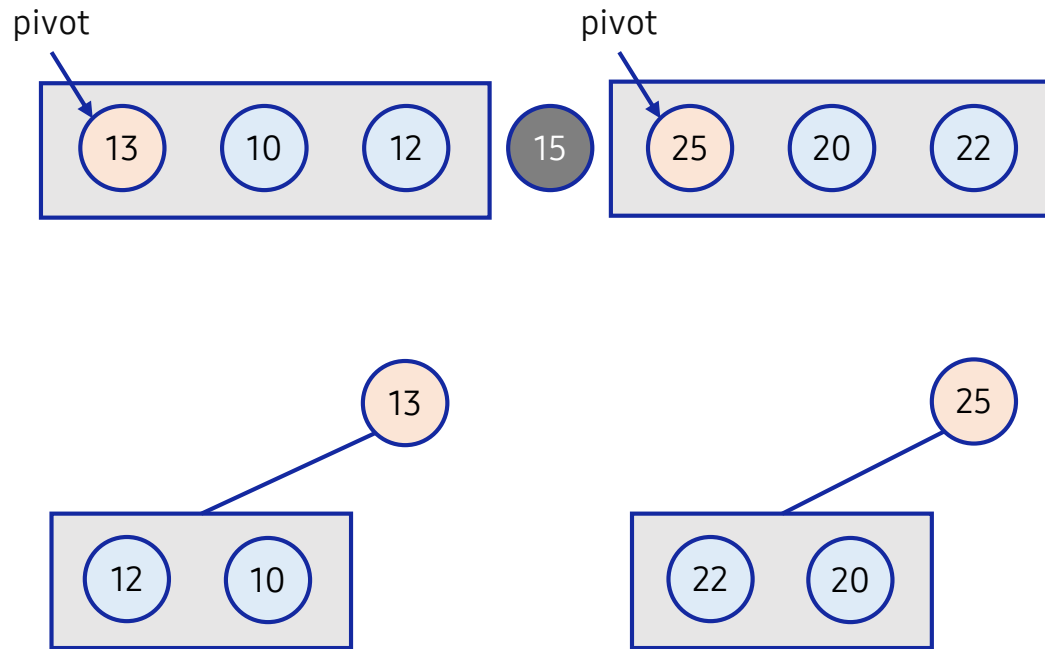
ເບື້ອງຕົ້ນ, ອົງປະກອບທຳອິດ ທີ່ມີຄ່າ 15 ຖືກກຳນົດເປັນ pivot. ອົງປະກອບທີ່ນ້ອຍກວ່າ pivot ຖືກຍ້າຍໄປທາງຊ້າຍຂອງ pivot ແລະ ອົງປະກອບທີ່ໃຫຍ່ກວ່າ pivot ແມ່ນຍ້າຍໄປທາງຂວາຂອງ pivot.



1. ການຈັດລຽງແບບ Quick

1.1. ຕົວຢ່າງຂອງການຈັດລຽງແບບ quick

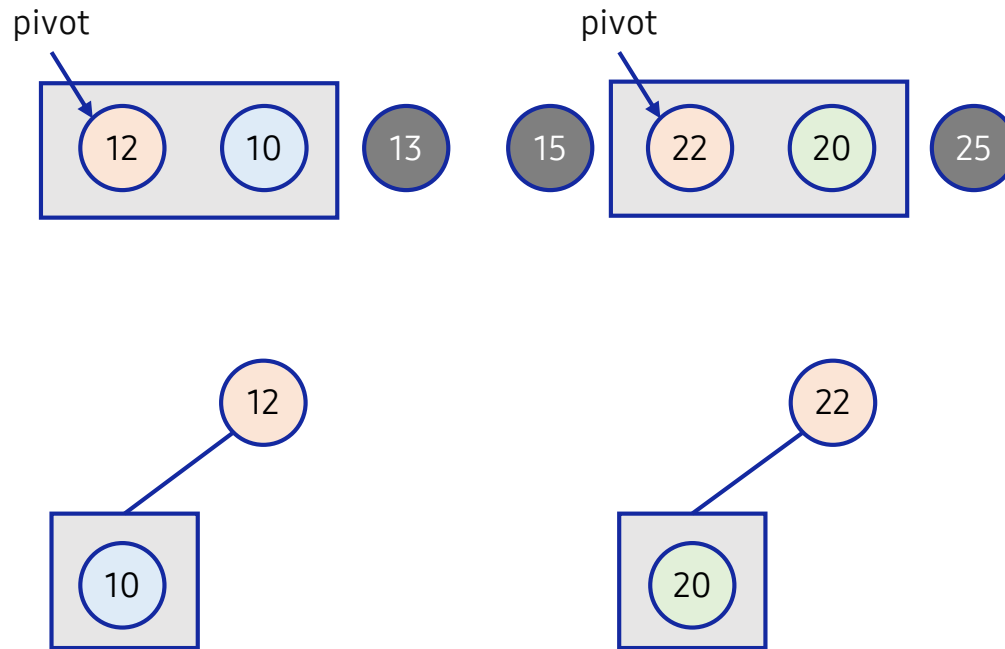
- ແບ່ງໂດຍການກຳນົດຕົວ pivot ເປັນອົງປະກອບທຳອິດໃນແຕ່ລະລາຍການ ທີ່ແບ່ງອອກ.
- ຕໍ່ໄປ, ແບ່ງລາຍການຍ່ອຍດ້ວຍ pivot ໃຫມ່ໃນແຕ່ລະລາຍການຍ່ອຍທີ່ໄດ້ແບ່ງອອກຕາມລຳດັບ.



1. ການຈັດລຽງແບບ Quick

1.1. ຕົວຢ່າງຂອງການຈັດລຽງແບບ quick

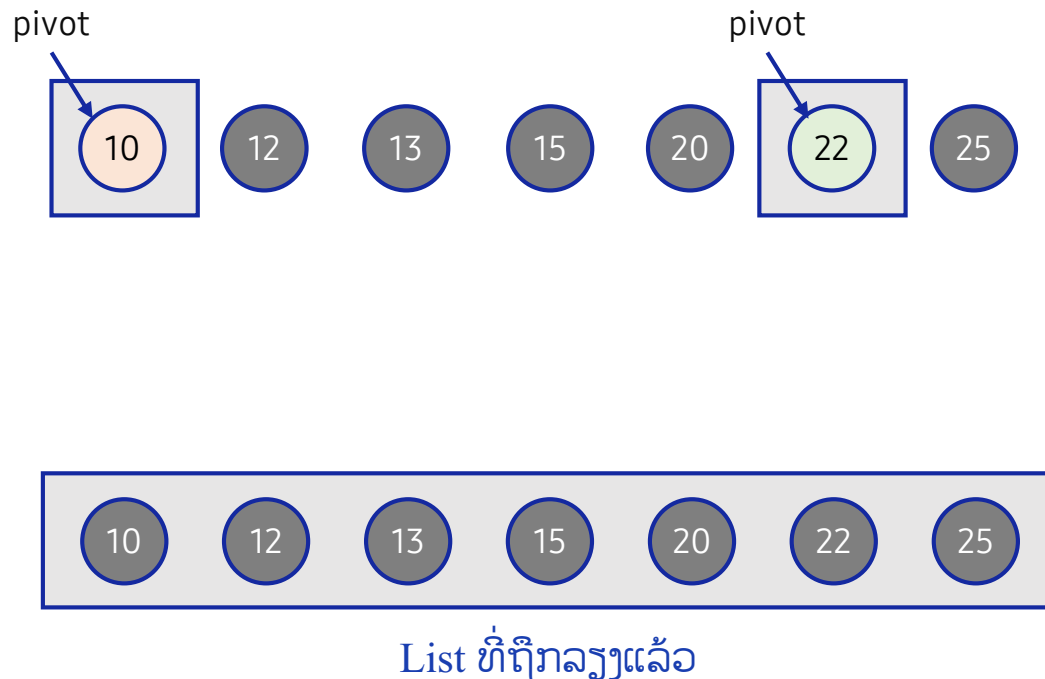
- ແບ່ງໂດຍການກຳໜົດ pivot ເປັນອົງປະກອບທຳອິດໃນແຕ່ລະລາຍການ ທີ່ແບ່ງອອກ.
- ຕໍ່ໄປ, ແບ່ງລາຍການຍ່ອຍດ້ວຍ pivot ໃຫມ່ໃນແຕ່ລະລາຍການຍ່ອຍທີ່ໄດ້ແບ່ງອອກຕາມລຳດັບ.



1. ການຈັດລຽງແບບ Quick

1.1. ຕົວຢ່າງຂອງການຈັດລຽງແບບ quick

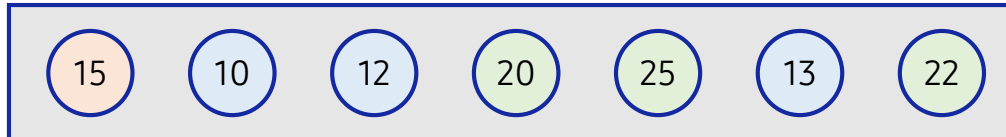
ຖ້າມີພຽງແຕ່ອົງປະກອບດຽວ, ຈະຢຸດການເອີ້ນໃຊ້ recursive ເນື່ອງຈາກວ່າມັນຖືກຈັດລຽງແລ້ວ. ໃນເວລານີ້, ຂໍ້ມູນທັງໝົດພາຍໃນລາຍການ ແມ່ນຖືກຈັດລຽງແລ້ວ.



1. ການຈັດລຽງແບບ Quick

1.2. ການແບ່ງສ່ວນໂດຍໃຊ້ pivot

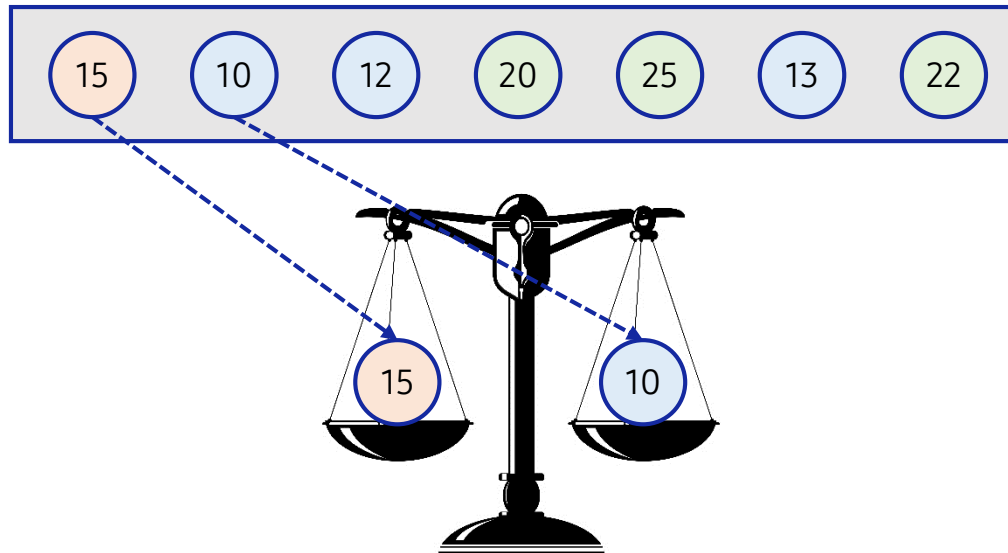
ຊອກຫາວິທີການແບ່ງທາງດ້ານຊ້າຍ ແລະ ຂວາ ໂດຍອີງໃສ່ pivot ໂດຍບໍ່ຕ້ອງໃຊ້ເນື້ອທີ່ໜ່ວຍຄວາມຈຳຈັດເກັບຂໍ້ມູນ ແຍກຕ່າງຫາກ.ນຳໃຊ້ຫຼັກການຄວາມສົມດຸນ.



1. ການຈັດລຽງແບບ Quick

1.2. ການແບ່ງສ່ວນໂດຍໃຊ້ pivot

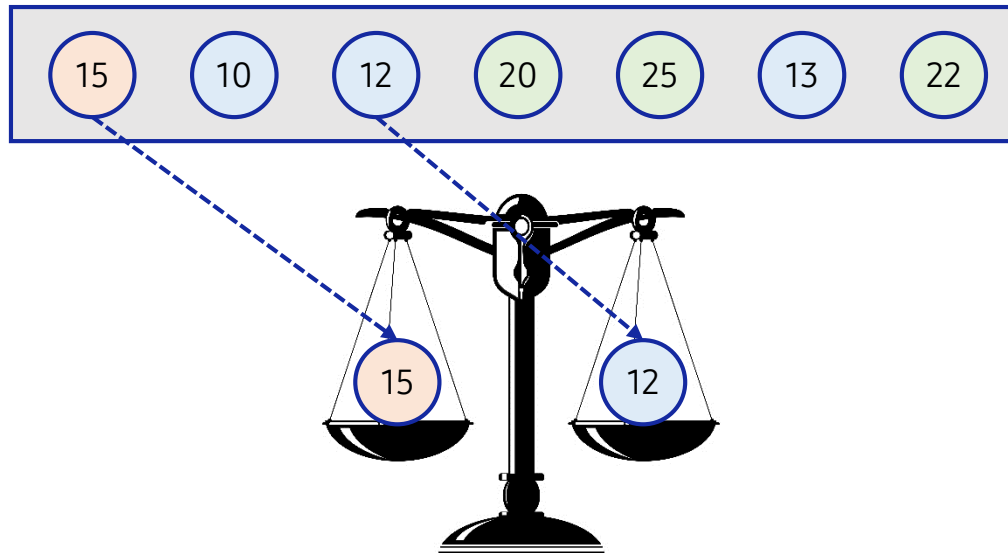
ກ່ອນອື່ນ, ຊອກຫາຄ່າທີ່ໃຫຍ່ກວ່າ pivot ຈາກຊ້າຍໄປຂວາໃນບັນດາອົງປະກອບທີ່ບໍ່ລວມເອົາ pivot. ເນື່ອງຈາກ 10 ໜ້ອຍກວ່າ 15, ມັນຈຶ່ງໄປຫາອົງປະກອບຕໍ່ໄປ.



1. ການຈັດລຽງແບບ Quick

1.2. ການແບ່ງສ່ວນໂດຍໃຊ້ pivot

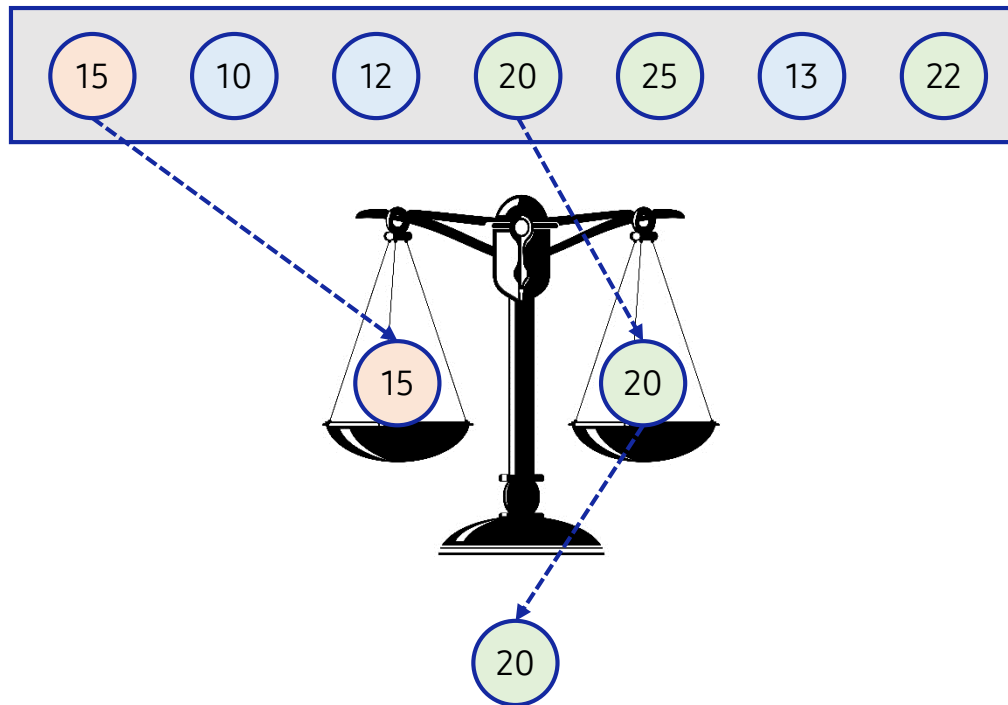
ເນື່ອງຈາກ 12 ນ້ອຍກວ່າ 15, ມັນຈຶ່ງໄປຫາອົງປະກອບຕໍ່ໄປ.



1. ການຈັດລຽງແບບ Quick

1.2. ການແບ່ງສ່ວນໂດຍໃຊ້ pivot

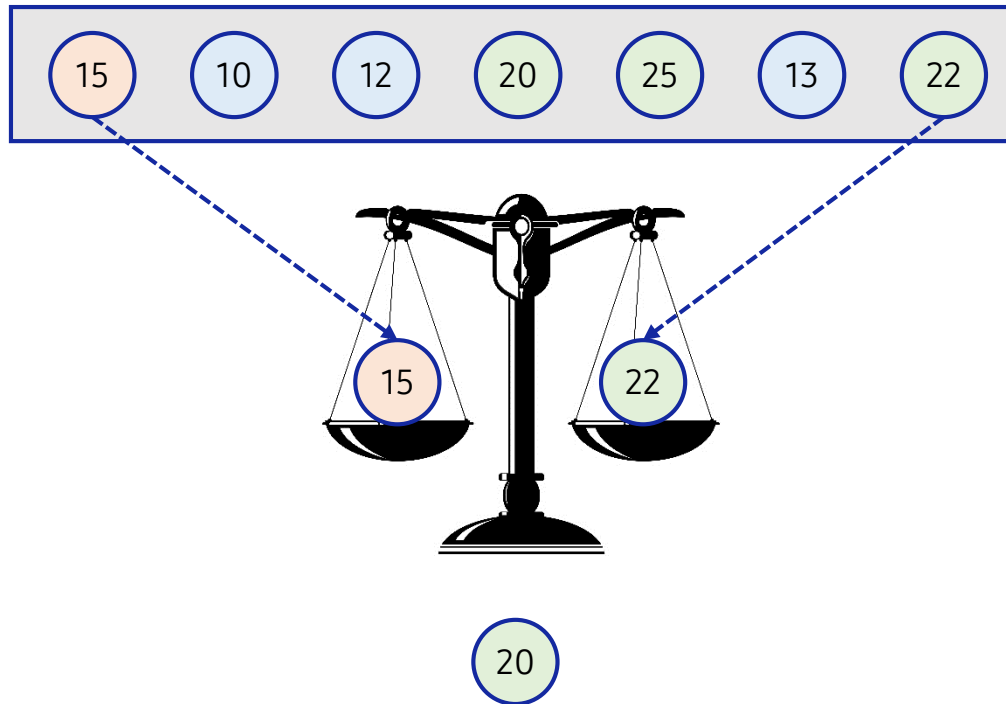
ເນື່ອງຈາກ 20 ໃຫຍ່ກວ່າ 15, 20 ແມ່ນອົງປະກອບທຳອິດໃນລາຍການທຳອິດທີ່ໃຫຍ່ກວ່າ pivot.



1. ການຈັດລຽງແບບ Quick

1.2. ການແບ່ງສ່ວນໂດຍໃຊ້ pivot

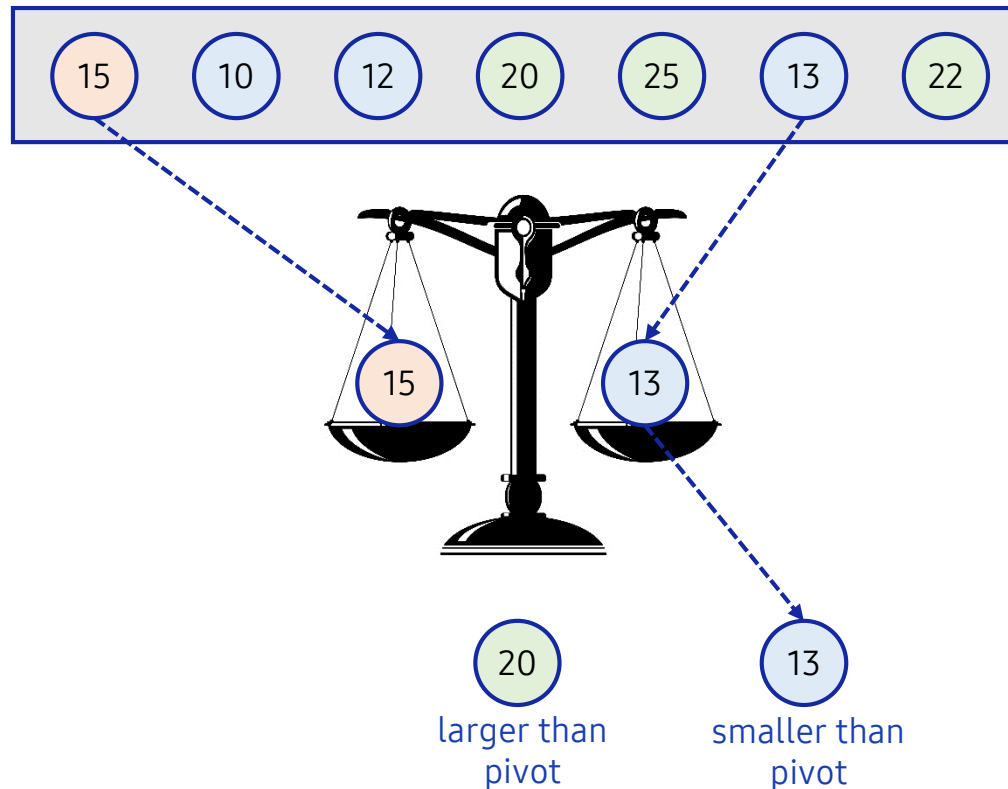
ຕໍ່ໄປ, ຊອກຫາອົງປະກອບທຳອິດທີ່ນ້ອຍກວ່າ pivot ຈາກຂວາຫາຊ້າຍ. ເນື່ອງຈາກ 22 ແມ່ນໃຫຍ່ກວ່າ 15, ມັນຈຶ່ງໄປທີ່ອົງປະກອບຕໍ່ໄປ.



1. ການຈັດລຽງແບບ Quick

1.2. ການແບ່ງສ່ວນໂດຍໃຊ້ pivot

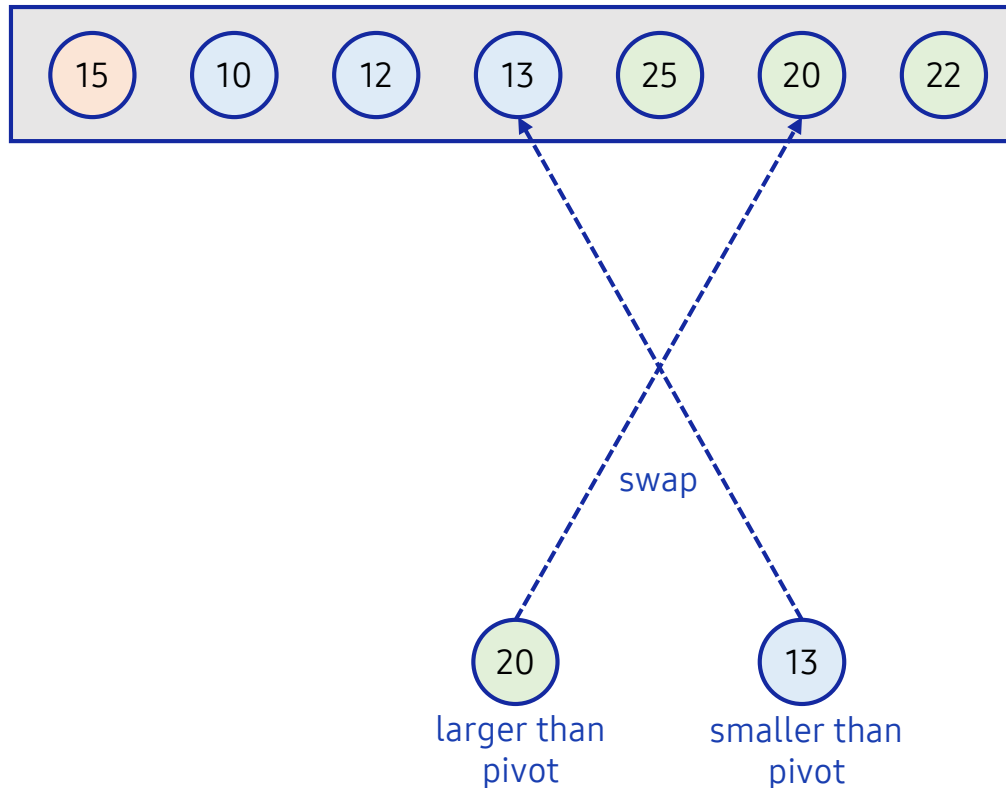
ເນື່ອງຈາກ 13 ນ້ອຍກວ່າ 15, 13 ແມ່ນອົງປະກອບສຸດທ້າຍທີ່ນ້ອຍກວ່າ 15.



1. ການຈັດລຽງແບບ Quick

1.2. ການແບ່ງສ່ວນໂດຍໃຊ້ pivot

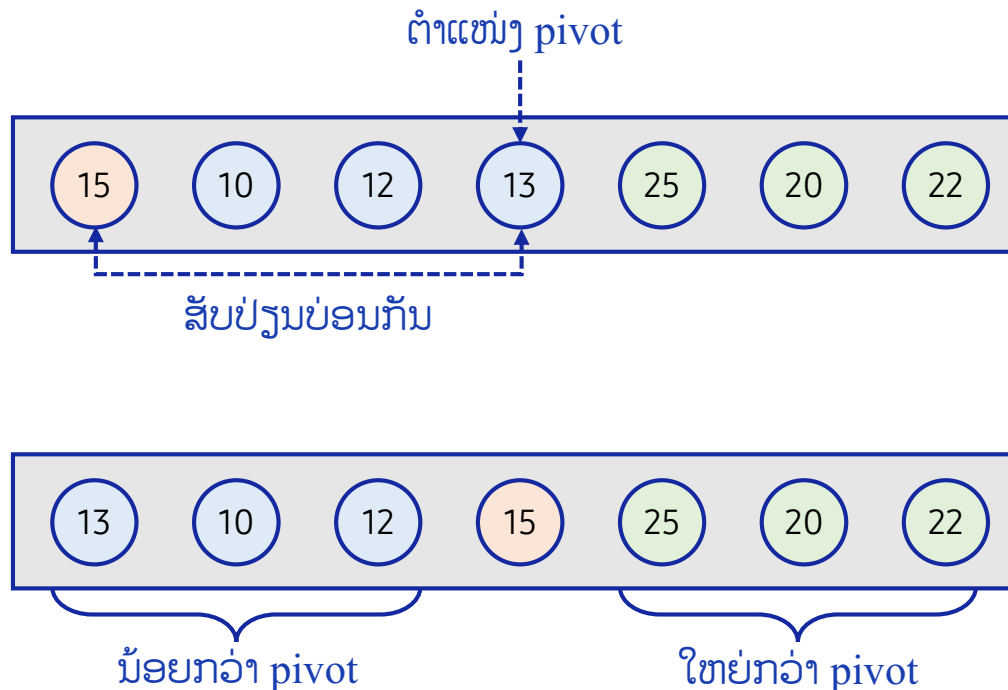
ອົງປະກອບທຳອິດທີ່ໃຫຍ່ກວ່າ pivot ແລະ ອົງປະກອບສຸດທ້າຍທີ່ນ້ອຍກວ່າ pivot ຖືກສະຫຼັບປ່ອນກັນ. ດ້ວຍການເຮັດຊື່າຂະບວນການນີ້, ອົງປະກອບທີ່ນ້ອຍກວ່າ pivot ແລະ ອົງປະກອບທີ່ໃຫຍ່ກວ່າ pivot ຖືກແບ່ງອອກເປັນສອງສ່ວນ.



1. ການຈັດລຽງແບບ Quick

1.2. ການແບ່ງສ່ວນໂດຍໃຊ້ pivot

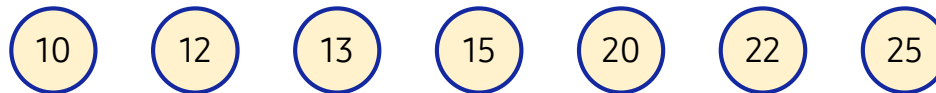
ໃນລາຍການທີ່ມີສອງສ່ວນ, ເມື່ອອົງປະກອບຢູ່ໃນຕຳແໜ່ງ pivot ແລະ ຕົວ pivot ຖືກສັບປ່ຽນບ່ອນເຊິ່ງກັນແລະກັນ, pivot ແມ່ນຖືກຈັດໃສ່ໃນຕຳແໜ່ງຂອງ pivot, ອົງປະກອບທີ່ນ້ອຍກວ່າ pivot ແມ່ນວາງໄວ້ທາງຊ້າຍ ແລະ ອົງປະກອບທີ່ໃຫຍ່ກວ່າ pivot ແມ່ນຖືກຈັດວາງຢູ່ເບື້ອງຂວາ.



1. ການຈັດລຽງແບບ Quick

1.3. ການເລືອກ pivot

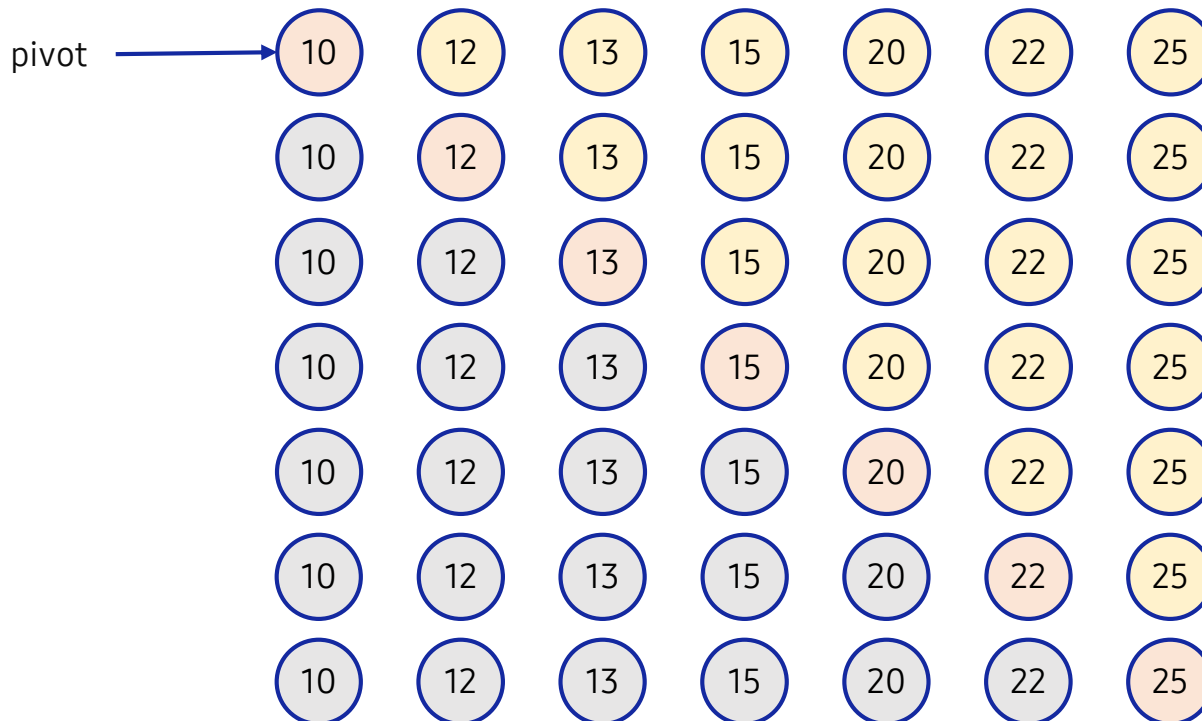
ກ່ອນຫນ້ານີ້, ພວກເຮົາໄດ້ໃຊ້ວິທີການຂອງການກຳນົດ pivot ເພື່ອເລືອກລາຍການທຳອິດໃນລາຍການ. ຈະເກີດຫຍັງຂຶ້ນຖ້າພວກເຮົາໃຊ້ວິທີການນີ້ຢູ່ໃນລາຍການທີ່ຈັດລຽງໄວ້ແລ້ວ ດັ່ງນີ້:



1. ການຈັດລຽງແບບ Quick

1.3. ການເລືອກ pivot

ເນື່ອງຈາກແຕ່ລະຄັ້ງທີ່ລາຍການທຳອິດໃນລາຍການ ແມ່ນອົງປະກອບທີ່ນ້ອຍທີ່ສຸດ, ຟັງຊັນ partition() ຈະບໍ່ສາມາດແບ່ງໄດ້. ພວກເຮົາຕ້ອງແບ່ງ N ຄັ້ງດັ່ງຕໍ່ໄປນີ້.



1. ການຈັດລຽງແບບ Quick

1.3. ການເລືອກ pivot

| ຈະເກີດຫຍັງຂຶ້ນຖ້າພວກເຮົາກຳນົດ pivot ເປັນອົງປະກອບໃດກໍ່ໄດ້? ໃນກໍລະນີທີ່ບໍ່ດີທີ່ສຸດ, ລໍາດັບທີ່ຈັດລຽງແລ້ວຖືກເລືອກ, ແຕ່ຄວາມເປັນໄປໄດ້ຂອງກໍລະນີດັ່ງກ່າວແມ່ນບໍ່ຄ່ອຍເປັນໄປໄດ້, ດັ່ງນັ້ນໃນກໍລະນີໂດຍສະເລ່ຍ, ມັນມີຄ່າ $O(N \log N)$.

| Let's code

1. ການສ້າງການຈັດລຽງແບບ Quick ດ້ວຍການເລືອກຕົວທຳອິດເປັນ Pivot

1.1. ການຈັດລຽງແບບ Quick ດ້ວຍວິທີການ divide-and-conquer

Quick sort ໃຊ້ຍຸດທະສາດ divide-and-conquer ເຊັ່ນດຽວກັນກັບການຈັດລຽງແບບ merge. ຄວາມແຕກຕ່າງຈາກການຈັດລຽງແບບ merge ມັນແບ່ງອອກເປັນສອງລາຍການ ໂດຍອີງໃສ່ pivot, ຂະໜາດຂອງສອງລາຍການອາດຈະແຕກຕ່າງກັນ.

```
1 def quicksort1(S, low, high):
2     if low < high:
3         print(S)
4         pivotpoint = partition1(S, low, high)
5         quicksort1(S, low, pivotpoint - 1)
6         quicksort1(S, pivotpoint + 1, high)
```

Line1~3

- ການຈັດລຽງແບບ Quick ຈັດລຽງອີງປະກອບຈາກ low ຫາ high ຢູ່ໃນ S ທີ່ໃຫ້ມາ.
- ຖ້າ low ເທົ່າກັບຫຼືຫນ້ອຍກວ່າ high, ການຈັດລຽງແມ່ນຖືກຢຸດເຊົາເພາະວ່າມັນຢູ່ໃນສະຖານະທີ່ຈັດລຽງແລ້ວ.

1. ການສ້າງການຈັດລຽງແບບ Quick ດ້ວຍການເລືອກຕົວທຳອິດເປັນ Pivot

1.1. ການຈັດລຽງແບບ Quick ດ້ວຍວິທີການ divide-and-conquer

■ ອີງຕາມ pivot, ຟັງຊັນ partition1() ແບ່ງອົງປະກອບຂອງ S ເປັນອົງປະກອບທີ່ນ້ອຍກວ່າ pivot ແລະ ອົງປະກອບທີ່ໃຫຍ່ກວ່າ pivot. ລາຍການທີ່ແບ່ງອອກສອງອັນນີ້ແມ່ນຖືກຈັດລຽງໂດຍຜ່ານການເອີ້ນໃຊ້ recursive.

```
1 def quicksort1(S, low, high):  
2     if low < high:  
3         print(S)  
4         pivotpoint = partition1(S, low, high)  
5         quicksort1(S, low, pivotpoint - 1)  
6         quicksort1(S, pivotpoint + 1, high)
```

Line4~6

- pivotpoint ສິ່ງຄືນຕຳແໜ່ງຂອງ pivot ໃນຟັງຊັນ partition1().
- ອົງປະກອບທີ່ນ້ອຍກວ່າ pivot ແມ່ນຖືກວາງໄວ້ທາງຊ້າຍຂອງຈຸດ pivot ແລະ ອົງປະກອບທີ່ໃຫຍ່ກວ່າ pivot ແມ່ນຖືກວາງໄວ້ທາງຂວາຂອງຈຸດ pivot.
- ອົງປະກອບທາງຊ້າຍ ແລະ ຂວາ ຖືກຈັດລຽງໂດຍອີງໃສ່ຈຸດ pivot, ຕາມລຳດັບ.

1. ການສ້າງການຈັດລຽງແບບ Quick ດ້ວຍການເລືອກຕົວທຳອິດເປັນ Pivot

1.2. ການແບ່ງສ່ວນດ້ວນການເລືອກອົງປະກອບທຳອິດເປັນ pivot

ຟັງຊັນ partition1() ແບ່ງອົງປະກອບຈາກ low ຫາ high ໃນ S ທີ່ໃຫ້ໄວ້ໂດຍອີງໃສ່ pivot. ສົມມຸດວ່າຍຸດທະສາດ pivot ແມ່ນເພື່ອເລືອກອົງປະກອບທຳອິດຂອງອົງປະກອບທີ່ໃຫ້ເປັນ pivot.

```

1 def partition1(S, low, high):
2     pivot = S[low]
3     left, right = low + 1, high
4     while left < right:
5         print(S)
6         while left <= right and S[left] <= pivot:
7             left += 1
8         while left <= right and S[right] >= pivot:
9             right -= 1
10        if left < right:
11            S[left], S[right] = S[right], S[left]
12        pivotpoint = right
13        S[low], S[pivotpoint] = S[pivotpoint], S[low]
14        return pivotpoint

```

Line1~2

- ຟັງຊັນ partition1() ໃຊ້ S, low ແລະ high ເປັນພາຣາມິເຕີ ຂໍ້ມູນປ້ອນເຂົ້າ.
- ອົງປະກອບທຳອິດໃນຂອບເຂດທີ່ໃຫ້ S[low] ຖືກກຳນົດເປັນ pivot.

1. ການສ້າງການຈັດລຽງແບບ Quick ດ້ວຍການເລືອກຕົວທຳອິດເປັນ Pivot

1.2. ການແບ່ງສ່ວນດ້ວນການເລືອກອົງປະກອບທຳອິດເປັນ pivot

ການປະຕິບັດການແບ່ງສ່ວນເລືອກຍຸດທະສາດເພື່ອສັບປ່ຽນອົງປະກອບໃນຂອບເຂດຂອງ S ໂດຍການປຽບທຽບພວກມັນຢູ່ທາງດ້ານຊ້າຍແລະຂວາ, ຕາມລຳດັບ.

```

1 def partition1(S, low, high):
2     pivot = S[low]
3     left, right = low + 1, high
4     while left < right:
5         print(S)
6         while left <= right and S[left] <= pivot:
7             left += 1
8         while left <= right and S[right] >= pivot:
9             right -= 1
10        if left < right:
11            S[left], S[right] = S[right], S[left]
12    pivotpoint = right
13    S[low], S[pivotpoint] = S[pivotpoint], S[low]
14    return pivotpoint

```

Line3~11

- ໃນບັນດາອົງປະກອບທັງໝົດທີ່ບໍ່ລວມ pivot, ດ້ານຊ້າຍແລະຂວາແມ່ນໄດ້ຖືກຄັດເລືອກເປັນອົງປະກອບຊ້າຍສຸດແລະຂວາສຸດ.
- ຊອກຫາອົງປະກອບທຳອິດຈາກຊ້າຍທີ່ໃຫຍ່ກວ່າ pivot ແລະ ອົງປະກອບທຳອິດຈາກຂວາທີ່ນ້ອຍກວ່າ pivot.
- ສະຫຼັບອົງປະກອບທາງຊ້າຍ ແລະ ຂວາ.

1. ການສ້າງການຈັດລຽງແບບ Quick ດ້ວຍການເລືອກຕົວທຳອິດເປັນ Pivot

1.2. ການແບ່ງສ່ວນດ້ວຍການເລືອກອົງປະກອບທຳອິດເປັນ pivot

ເມື່ອການປະຕິບັດການສັບປ່ຽນສິ້ນສຸດລົງ, ອົງປະກອບທຳອິດທີ່ເລືອກໂດຍ pivot ໄດ້ຖືກຍ້າຍໄປຢູ່ໃນຕຳແໜ່ງກາງ.

```

1 def partition1(S, low, high):
2     pivot = S[low]
3     left, right = low + 1, high
4     while left < right:
5         print(S)
6         while left <= right and S[left] <= pivot:
7             left += 1
8         while left <= right and S[right] >= pivot:
9             right -= 1
10        if left < right:
11            S[left], S[right] = S[right], S[left]
12        pivotpoint = right
13        S[low], S[pivotpoint] = S[pivotpoint], S[low]
14        return pivotpoint

```

Line12-14

- ໃນ Line3-11, ເມື່ອການປະຕິບັດການສັບປ່ຽນສຳເລັດ, ອົງປະກອບ S [pivotpoint] ເປັນອົງປະກອບສຸດທ້າຍຂອງອົງປະກອບທີ່ນ້ອຍກວ່າອົງປະກອບ pivot.
- ຖ້າອົງປະກອບ pivot ຢູ່ໃນຕຳແໜ່ງ low ແລະ ອົງປະກອບໃນຕຳແໜ່ງ pivotpoint ຖືກສັບປ່ຽນເຊິ່ງກັນແລະກັນ, ການດຳເນີນງານການແບ່ງສ່ວນແມ່ນສຳເລັດ.

1. ການສ້າງການຈັດລຽງແບບ Quick ດ້ວຍການເລືອກຕົວທຳອິດເປັນ Pivot

1.3. Running quick sort

ເມື່ອຂັ້ນຕອນວິທີ quick sort ທີ່ໄດ້ສ້າງກ່ອນໜ້ານັ້ນຖືກປະຕິບັດ, quick sort ແມ່ນດຳເນີນໂດຍຜ່ານຂະບວນການຕໍ່ໄປນີ້.

```
1 S = [15, 10, 12, 20, 25, 13, 22]
2 quicksort1(S, 0, len(S) - 1)
3 print(S)
```

```
[15, 10, 12, 20, 25, 13, 22]
[15, 10, 12, 20, 25, 13, 22]
[15, 10, 12, 13, 25, 20, 22]
[13, 10, 12, 15, 25, 20, 22]
[13, 10, 12, 15, 25, 20, 22]
[12, 10, 13, 15, 25, 20, 22]
[10, 12, 13, 15, 25, 20, 22]
[10, 12, 13, 15, 25, 20, 22]
[10, 12, 13, 15, 22, 20, 25]
[10, 12, 13, 15, 20, 22, 25]
```


1. ການສ້າງການຈັດລຽງແບບ Quick ດ້ວຍການເລືອກຕົວທຳອິດເປັນ Pivot

1.3. Running quick sort

ກໍລະນີທີ່ບໍ່ດີທີ່ສຸດຂອງ quick sort ແມ່ນເມື່ອ S ທີ່ໃຫ້ຖືກຈັດລຽງຕາມລຳດັບຈາກໃຫຍ່ຫນ້ອຍ. ໃນກໍລະນີນີ້, ມັນສາມາດຢືນຢັນໄດ້ຢ່າງແທ້ຈິງວ່າ $N-1$ ອົງປະກອບ ຖືກແບ່ງອອກໃນແຕ່ລະຄັ້ງທີ່ເອີ້ນໃຊ້ recursive.

```
1 S = [25, 22, 20, 15, 13, 12, 10]
2 quicksort1(S, 0, len(S) - 1)
3 print(S)
```

```
[25, 22, 20, 15, 13, 12, 10]
[25, 22, 20, 15, 13, 12, 10]
[10, 22, 20, 15, 13, 12, 25]
[10, 22, 20, 15, 13, 12, 25]
[10, 22, 20, 15, 13, 12, 25]
[10, 22, 20, 15, 13, 12, 25]
[10, 12, 20, 15, 13, 22, 25]
[10, 12, 20, 15, 13, 22, 25]
[10, 12, 20, 15, 13, 22, 25]
[10, 12, 20, 15, 13, 22, 25]
[10, 12, 13, 15, 20, 22, 25]
[10, 12, 15, 13, 20, 22, 25]
```

2. ການສ້າງການຈັດລຽງແບບ Quick ດ້ວຍການເລືອກ Pivot ແບບສຸ່ມ

2.1. Quick sort ດ້ວຍການກຳນົດ pivot ແບບສຸ່ມ

ຂັ້ນຕອນວິທີ quicksort1() ເລືອກອົງປະກອບທຳອິດຂອງລາຍການເປັນ pivot. ຂັ້ນຕອນວິທີ quicksort2() ເລືອກອົງປະກອບແບບສຸ່ມເປັນ pivot. ການເລືອກ pivot ຖືກປະຕິບັດຢູ່ໃນຟັງຊັນ partition2().

```
1 def quicksort2(S, low, high):  
2     if low < high:  
3         pivotpoint = partition2(S, low, high)  
4         quicksort2(S, low, pivotpoint - 1)  
5         quicksort2(S, pivotpoint + 1, high)
```

2. ການສ້າງການຈັດລຽງແບບ Quick ດ້ວຍການເລືອກ Pivot ແບບສຸ່ມ

2.2. ການເລືອກ pivot ແບບສຸ່ມ

ຟັງຊັນ partition2() ເລືອກອົງປະກອບແບບສຸ່ມຈາກອົງປະກອບຂອງ S ເປັນ pivot.

```

1 from random import randint
2
3 def partition2(S, low, high):
4     rand = randint(low, high)
5     S[low], S[rand] = S[rand], S[low]
6     pivot, left, right = S[low], low, high
7     print(S, left, right, "pivot = ", pivot)
8     while left < right:
9         while left < high and S[left] <= pivot:
10             left += 1
11         while right > low and pivot <= S[right]:
12             right -= 1
13         if left < right:
14             S[left], S[right] = S[right], S[left]
15     S[low], S[right] = S[right], S[low]
16     return right

```

Line4~6

- ຕຳແໜ່ງ Random ພາຍໃນຂອບເຂດ low ແລະ high ທີ່ກຳນົດໄວ້ເປັນຄ່າ rand.
- ເມື່ອ S[low] ແລະ S[rand] ຖືກສັບປ່ຽນ, ອົງປະກອບທຳອິດຖືກປ່ຽນເປັນອົງປະກອບແບບສຸ່ມ.

2. ການສ້າງການຈັດລຽງແບບ Quick ດ້ວຍການເລືອກ Pivot ແບບສຸ່ມ

2.3. Running quick sort ດ້ວຍການກຳນົດ pivot ແບບສຸ່ມ

ສັງເກດເຫັນວ່າ ຂັ້ນຕອນວິທີ quicksort2() ບໍ່ແມ່ນກໍລະນີທີ່ບໍ່ດີທີ່ສຸດເຖິງແມ່ນວ່າກໍລະນີການປ້ອນຂໍ້ມູນທີ່ຈັດລຽງຕາມລຳດັບຈາກໃຫຍ່ໄປຫນ້ອຍ, ບໍ່ເໝືອນກັບຂັ້ນຕອນວິທີ quicksort1().

```
1 S = [25, 22, 20, 15, 13, 12, 10]
2 quicksort2(S, 0, len(S) - 1)
3 print(S)
```

```
[12, 22, 20, 15, 13, 25, 10] 0 6 pivot = 12
[10, 12, 25, 15, 13, 20, 22] 2 6 pivot = 25
[10, 12, 22, 15, 13, 20, 25] 2 5 pivot = 22
[10, 12, 15, 20, 13, 22, 25] 2 4 pivot = 15
[10, 12, 13, 15, 20, 22, 25]
```

| Pop quiz

Q1. ຕາມ list ຂ້າງລຸ່ມນີ້, ຂຽນຜົນໄດ້ຮັບຫຼັງຈາກປະຕິບັດງານທີ່ຂອງ partition1().

```
1 S = [15, 10, 12, 20, 25, 13, 22]
2 partition1(S, 0, len(S) - 1)
3 print(S)
```

| Pair programming



Pair Programming Practice

ແນວທາງ, ກິນໄກ ແລະ ແຜນສຸກເສີນ

ການກະກຽມການສ້າງໂປຣແກຣມຮ່ວມກັນເປັນຄູ່ກ່ຽວຂ້ອງກັບການໃຫ້ຄໍາແນະນໍາແລະກິນໄກເພື່ອຊ່ວຍໃຫ້ນັກຮຽນຈັບຄູ່ຢ່າງຖືກຕ້ອງແລະ ໃຫ້ເຂົາເຈົ້າເຮັດວຽກເປັນຄູ່. ຕົວຢ່າງ, ນັກຮຽນຄວນປ່ຽນ "ເຮັດ." ການກະກຽມທີ່ມີປະສິດທິຜົນຕ້ອງໃຫ້ມີແຜນການສຸກເສີນໃນກໍລະນີທີ່ຄູ່ຮ່ວມງານຫນຶ່ງບໍ່ຢູ່ຫຼືຕັດສິນໃຈທີ່ຈະບໍ່ເຂົ້າຮ່ວມດ້ວຍເຫດຜົນໃດຫນຶ່ງ ຫຼືດ້ວຍເຫດຜົນອື່ນ. ໃນກໍລະນີເຫຼົ່ານີ້, ມັນເປັນສິ່ງສໍາຄັນທີ່ຈະເຮັດໃຫ້ມັນຊັດເຈນວ່ານັກຮຽນທີ່ມີປະຕິບັດໜ້າທີ່ຢ່າງຫ້າວຫັນຈະບໍ່ຖືກລົງໂທດຍ້ອນວ່າການຈັບຄູ່ບໍ່ໄດ້ຜິດ.

ການຈັບຄູ່ທີ່ຄ້າຍຄືກັນ, ບໍ່ຈໍາເປັນຕ້ອງເທົ່າທຽມກັນ, ຄວາມສາມາດເປັນຄູ່ຮ່ວມງານ

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

ກະຕຸ້ນນັກຮຽນໂດຍການໃຫ້ສິ່ງຈູງໃຈພິເສດ

ການສະເໜີແຮງຈູງໃຈພິເສດສາມາດຊ່ວຍກະຕຸ້ນນັກຮຽນໃຫ້ຈັບຄູ່. ໂດຍສະເພາະກັບນັກຮຽນທີ່ມີຄວາມສາມາດຫຼາຍ. ຈະເຫັນວ່າມັນເປັນປະໂຫຍດທີ່ຈະໃຫ້ນັກຮຽນຈັບຄູ່ເຮັດວຽກຮ່ວມກັນພຽງແຕ່ຫນຶ່ງຫຼືສອງວຽກເທົ່ານັ້ນ.



Pair Programming Practice

| ປ້ອງກັນການໂກງໃນການເຮັດວຽກຮ່ວມກັນ

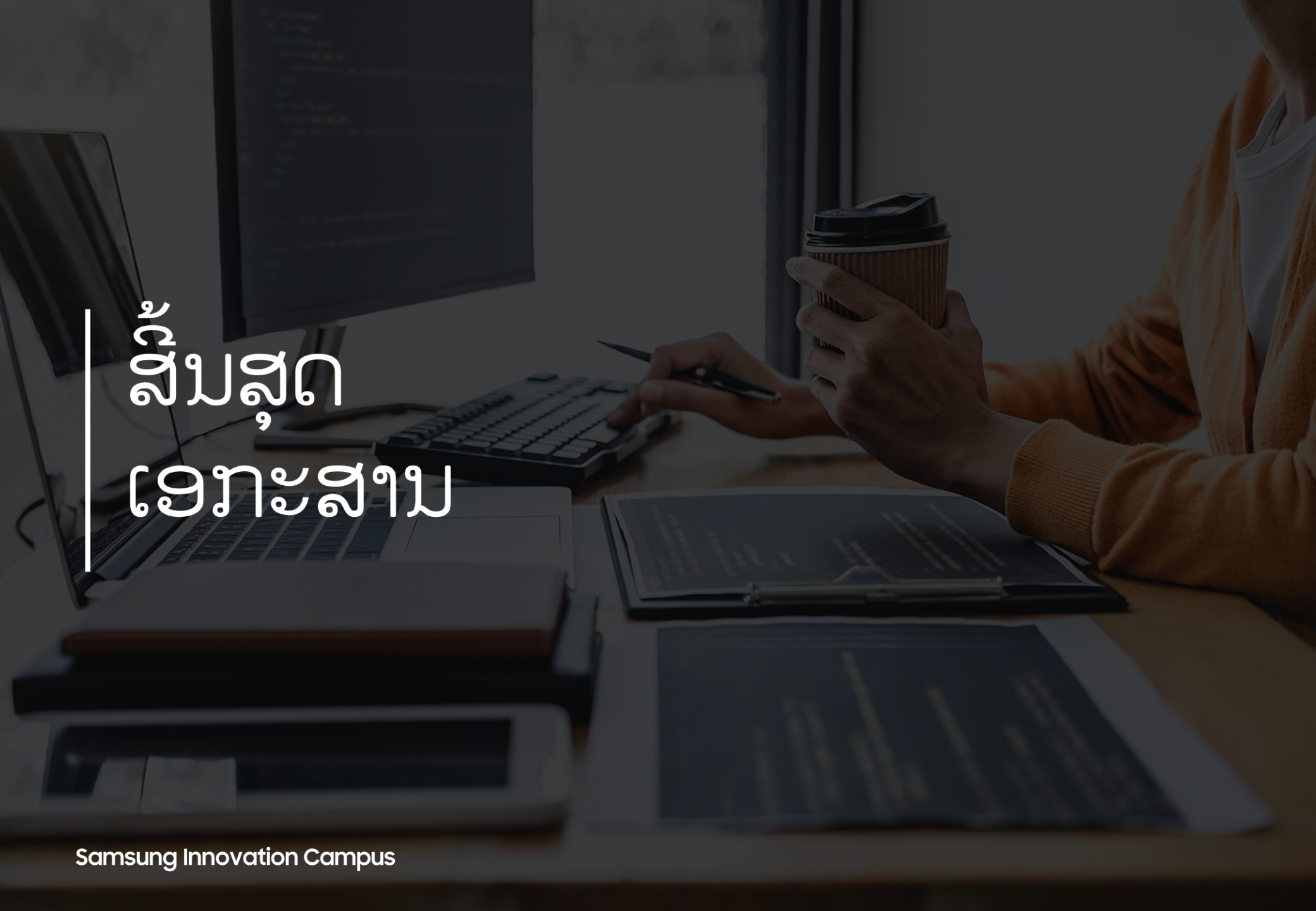
ສິ່ງທ້າທາຍສໍາລັບຄູແມ່ນເພື່ອຊອກຫາວິທີທີ່ຈະປະເມີນຜົນໄດ້ຮັບຂອງບຸກຄົນ, ໃນຂະນະທີ່ນໍາໃຊ້ຜົນປະໂຫຍດຂອງການຮ່ວມມື. ຈະຮູ້ໄດ້ແນວໃດວ່າ ນັກຮຽນຕັ້ງໃຈເຮັດວຽກ ຫຼື ກິດແຮງງານຜູ້ຮ່ວມງານ? ຜູ້ຊ່ຽວຊານແນະນໍາໃຫ້ທົບທວນຄືນການອອກແບບຫຼັກສູດ ແລະ ການປະເມີນ ພ້ອມທັງປຶກສາຫາລື ຢ່າງຈະແຈ້ງ ແລະ ຊັດເຈນກ່ຽວກັບພຶດຕິກຳຂອງນັກຮຽນທີ່ຈະຖືກຕີຄວາມວ່າຂີ້ຕົວະ. ຜູ້ຊ່ຽວຊານເນັ້ນໜັກໃຫ້ຄູເຮັດການມອບໝາຍໃຫ້ມີຄວາມໝາຍຕໍ່ ນັກຮຽນ ແລະ ອະທິບາຍຄຸນຄ່າຂອງສິ່ງທີ່ນັກຮຽນຈະຮຽນຮູ້ໂດຍການເຮັດສໍາເລັດ.

| ສະພາບແວດລ້ອມການຮຽນຮູ້ໃນການຮ່ວມມື

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

Q1. ຂຽນຂັ້ນຕອນວິທີທີ່ຊອກຫາອົງປະກອບທີ່ໃຫຍ່ທີ່ສຸດອັນດັບທີ K, ໃຫ້ N ອົງປະກອບທີ່ບໍ່ມີລຳດັບ.
ຫຼັງຈາກການແກ້ໄຂບັນຫາດ້ວຍສອງວິທີການຂ້າງເທິງນີ້, ວິເຄາະວ່າວິທີການໃດທີ່ມີປະສິດທິພາບຫຼາຍກວ່າ.

- ▶ ທ່ານສາມາດສິ່ງຄືນອົງປະກອບທີ K ຫຼັງຈາກໃຊ້ຟັງຊັນການຈັດລຽງ.
- ▶ ທ່ານສາມາດນຳໃຊ້ຟັງຊັນ `partition()` ເພື່ອເຮັດໃຫ້ການເອີ້ນໃຊ້ recursive ຈົນກ່ວາ pivot ເປັນອົງປະກອບທີ K.

A person is sitting at a desk in a dimly lit office or home workspace. They are wearing an orange sweater and holding a brown paper coffee cup with a black lid in their left hand. Their right hand is holding a pen and pointing at a laptop screen. On the desk, there is a laptop, a keyboard, a mouse, and some papers. The background is dark and out of focus.

ສິ້ນສຸດ ເອກະສານ



SAMSUNG

Together for Tomorrow!
Enabling People

Education for Future Generations

©2021 SAMSUNG. All rights reserved.

Samsung Electronics Corporate Citizenship Office holds the copyright of book.

This book is a literary property protected by copyright law so reprint and reproduction without permission are prohibited.

To use this book other than the curriculum of Samsung Innovation Campus or to use the entire or part of this book, you must receive written consent from copyright holder.

