CP 111 Computer Programming (2/2554 Section B01-B02)

Handout 9

<u>หัวข้อ:</u>

Recursion (Chapter 17 textbook)

- Basic concepts

Examples: Running sum, Factorial, Fibonacci, Tower of Hanoi, Binary Search

---

Dynamic memory Allocation (Chapter 19.4 textbook): Dynamically-Sized array, `malloc()`, `free()`

File I/O (Chapter 18.5 textbook)

Miscellaneous C features:

- C preprocessor: Macro substitution (Appendix D.8 textbook),
- Conditional Expression operator: `expA ? expB : expC`
- `typedef`
- Conditional compilation using pre-processing directive

สัปดาห์ถัดไป

<u>Basic concepts of Recursion</u>

1. ปัญหา RunningSum: พิจารณา<u>ฟังก์ชันทางคณิตศาสตร์</u> RunningSum(n) ซึ่งนิยามดังนี้

RunningSum(n) = ค่าผลรวมของ n, n-1, n-2, …, 2, 1

= n + (n-1) + (n-2) + …. + 2 + 1

ตัวอย่าง เช่น กรณี n=10 เราจะได้ว่า RunningSum(10) = 10+9+8+7+6+5+4+3+2+1 = 55

ดังนั้น

RunningSum(n-1) = ……

RunningSum(n-2) =……

RunningSum(1) =……

ดังนั้น

ความสัมพันธ์ระหว่าง RunningSum(n) และ RunningSum(n-1) คือ

……

โดยทั่วไปในเชิงคณิตศาสตร์ RunningSum สามารถนิยามในรูปแบบ Recurrence equation ได้ดังนี้

$$\text{RunningSum}(n) = \begin{cases} \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots & \text{กรณี } n>1 \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots & \text{กรณี } n=1 \end{cases}$$

<u>ตัวอย่าง</u> โดยใช้ Recurrence equation ข้างบน จงหาค่า RunningSum(4)

RunningSum(4) = ……

= ……

= ……

= ……

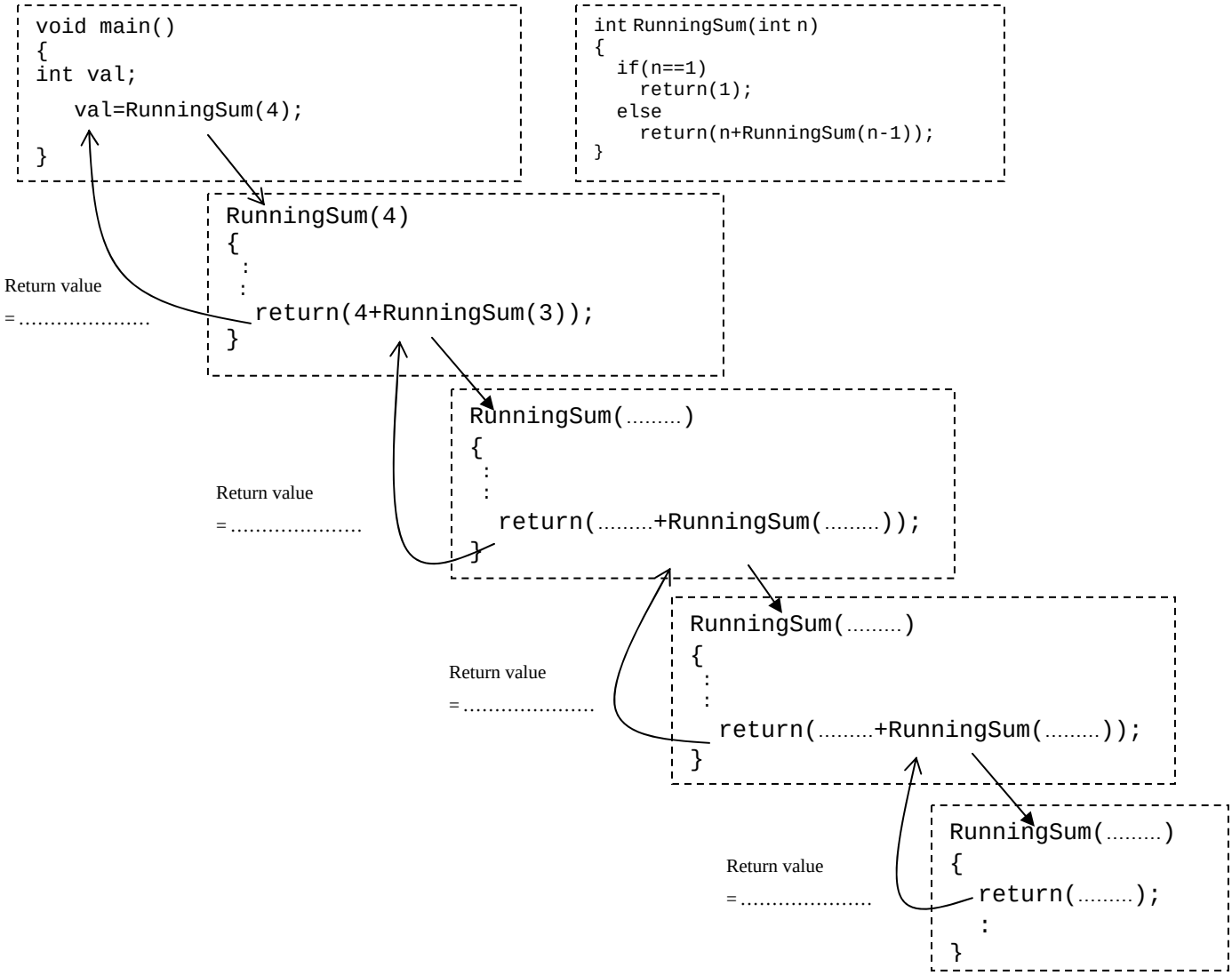
2. Coding ปัญหา RunningSumจากแนวคิดของ Recurrence equation ดังกล่าว เราสามารถเขียนโค้ดภาษา C ในรูปของฟังก์ชันในที่นี้ชื่อ `RunninSum` ซึ่งจะ Call ไปยังฟังก์ชันตัวเอง (`RunningSum`) ได้ดังโปรแกรมข้างล่าง

```
int RunningSum(int n)
{
int sum,p;
      if(n==1)
            ...................................................
      else
      {     // กรณี n > 1
            p= RunningSum(.............);
            sum=.............................................
            return(sum);
      }
}
```

โค้ด 3 บรรทัดนี้สามารถยุบเหลือบรรทัดเดียวได้เป็น

.....

3. รูปข้างล่างแสดง Flow ของการเรียกฟังก์ชันเมื่อ `RunningSum` ถูกเรียกใน `main` โดยส่งค่า 4 มาเป็นอาร์กิวเมนต์



4. Definition: โค้ดของฟังก์ชัน RunningSum ข้างบน เป็นตัวอย่างของการแก้ปัญหา (computer programming problems) โดยใช้เทคนิคที่เรียกว่า Recursion (การเวียนเกิด) ซึ่งสามารถนิยามลักษณะของเทคนิคได้ดังนี้

- Recursion หรือ Recursive function เป็นเทคนิคในการแก้ปัญหาโดยที่ฟังก์ชันจะทำการเรียกตัวมันเอง
- โดยแต่ละครั้งของการเรียกฟังก์ชันนั้นขนาดของข้อมูลอินพุตจะเล็กลง (smaller subtask) จนกระทั่งถึงจุดที่ข้อมูลอินพุตเป็นกรณี base case (เช่น กรณี n=1 ในตัวอย่าง RunningSum) ก็จะใช้แก้ปัญหาแบบ non-recursive

5.คุณลักษณะของการแก้ปัญหาแบบ recursion คือ

(i) ปัญหาหรือ task หนึ่งๆจะต้องนิยามในรูปแบบของตัวมันเอง โดยที่มีขนาดของปัญหามีขนาดเล็กลง

(ii) Task นั้นจะต้องมีนิยามของกรณี Base case ซึ่งจะแก้ปัญหาในรูปแบบ non-recursive (ไม่ได้เรียกตัวมันเอง)

6. ตัวอย่างการแก้ปัญหาแบบ Recursion: Factorial

(i) จงเขียน Recurrence equation เพื่อนิยาม n! (factorial of n)

$$n! = \begin{cases} \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots & \text{กรณี } n>0 \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots & \text{กรณี } n=0 \text{ (Base case)} \end{cases}$$

(ii) โดยใช้ Recurrence equation ข้างบน จงเขียนโค้ดในฟังก์ชัน Factorial ซึ่งจะทำการคำนวณหาค่า n! ของอินพุตอาร์กิวเมนต์ n

```
int Factorial(int n)
{
int val,p;

      if(.....)

            ..............................................
      else
      {
            p==.............................................
            val =...........................................
            ..............................................
      }
}
```

สามารถเขียนโค้ดให้กระชับขึ้นเป็น

```
int Factorial(int n)
{



}
```

7. Why recursion? (Recursion versus Iteration)

คำถาม: จะเห็นได้ว่าทั้งกรณี RunningSum และ Factorial สามารถเขียนโปรแกรมในรูปแบบ iteration (ใช้ลูป while, for, do-while) แล้วทำไมเราต้องศึกษาเทคนิค Recursion ในการแก้ปัญหา (***จริง ๆแล้ว*** ในเกือบทุกปัญหาที่แก้ได้โดย Recursion จะสามารถแก้โดยการเขียนโปรแกรมแบบ iteration)

คำตอบ: ในบางปัญหา การใช้เทคนิค Recursion ในการเขียนโปรแกรมจะทำให้โปรแกรมดูง่ายและมีความสวยงามในเชิงหลักการกว่าการเขียนโปรแกรมโดยใช้เทคนิค Iteration ตัวอย่าง เช่น ปัญหาปริศนาหอคอยฮานอย (Tower of Hanoi) หรือปัญหา Fibonacci numbers

8. ปัญหาปริศนาหอคอยฮานอย (Tower of Hanoi): เป็นหนึ่งในปัญหาที่แก้ด้วยเทคนิค Recursion แล้วดูง่ายกว่าการใช้ iteration

(i) Problem statement:

- มีเสา (post) อยู่ 3 เสา สมมติเรียกว่า 1, 2 และ 3 ตามลำดับ
- มีแผ่นไม้ที่มีรูตรงกลาง (disk) ขนาดต่างๆ วางซ้อนกันบนเสาต้นหนึ่ง โดยแผ่นที่มีขนาดเล็กจะวางซ้อนบนแผ่นขนาดใหญ่
- Goal: ให้ทำการเคลื่อนย้าย Disk ที่อยู่ในเสาต้นปัจจุบันไปอยู่ที่เสาต้นอื่น โดยมีกฎการย้ายอยู่ 3 อย่างคือ
    (1) ทำการย้าย disk ได้ครั้งละ 1 แผ่น
    (2) ห้ามวาง disk ขนาดใหญ่กว่าซ้อนทับบน disk ขนาดเล็กกว่า
    (3) สามารถใช้เสาอีกต้นที่ไม่ใช้เสาเป้าหมายเป็นตัวช่วยได้

ตัวอย่างของปริศนาพร้อมทั้งคำเฉลย กรณีมี 3 disk และต้องการย้ายจากบนเสาต้นที่ 1 ไปอยู่บนเสาต้นที่ 3 โดยใช้เสาต้นที่ 2 เป็นตัวช่วย ดูรูปที่หน้าถัดไป

ในตำนานกล่าวกันว่า นักบวชแห่งวัด Bharma ถูกมอบหมายให้ทำการแก้ปริศนาในการเคลื่อนย้าย disk จำนวน64 disk โดยถ้านักบวชทำสำเร็จ โลกก็จะถึงวันดับสูญ ในวิชานี้เราจะศึกษาการเขียนโปรแกรมโดยใช้เทคนิค Recursion มาสั่งให้ Computer ทำการแก้ปัญหานี้

(ii) Recursive formulation:

เราสามารถกล่าวถึงปัญหาในลักษณะทั่วไปได้ดังนี้

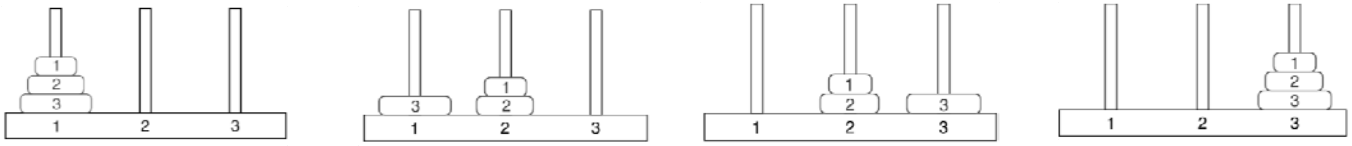
"มีทั้งหมด n disk (เช่น n=64) อยู่ที่เสาต้นที่ startPost (เช่น startPost=1) และเราต้องการย้ายไปเสาต้นที่ endPost (เช่น endPost=3) โดยสามารถใช้เสาต้นที่ tempPost (เช่น tempPost=2) เป็นเสาตัวช่วย และกำหนดให้ disk มีหมายเลข 1, 2,…n โดย disk ใหญ่สุดคือหมายเลข n และ disk เล็กสุดคือหมายเลข n"

เราสามารถอธิบายขั้นตอนการย้าย disk ในรูปแบบ Recursion ได้ดังนี้

Step 1: ทำการย้าย disk จำนวน ……..disk ที่อยู่บนสุด จากเสา……ไปยังเสา …… โดยใช้เสา…….. เป็นตัวช่วย

Step 2: ทำการย้าย disk ใหญ่สุด (disk หมายเลข n) จากเสา……ไปยังเสา …… โดยใช้เสา…….. เป็นตัวช่วย

Step 3: ทำการย้าย disk จำนวน ……..disk ที่อยู่บนสุด จากเสา……ไปยังเสา …… โดยใช้เสา…….. เป็นตัวช่วย



เริ่มต้น	Step 1: Move n-1 disks from post 1 to post 2	Step 2: Move disk 3 from post 1 to post 3	Step 3: Move n-1 disks from post 2 to post 3
original task of size n (n=3)	***Recursion with subtask of size n-1 (2)***	***non-recursive subtask***	***Recursion with subtask of size n-1 (2)***

<u>หมายเหตุ</u>

- Step 1 และ 3 นิยามเป็น Subtask ที่อยู่ในรูปของ Recursive ที่มีขนาดลดลง (จาก n เป็น n-1)
- Step 2 เป็น task ที่แก้แบบ non-recursive (ไม่ได้นิยามในรูปของฟังก์ชันเรียกตัวเอง)

(iii) Coding: จากแนวคิดข้างบนสามารถนำมาเขียนเป็นโปรแกรมภาษา C ในรูปแบบของ recursive function ได้ดังนี้

นิยาม: ฟังก์ชัน `void Move(int n, int startPost, int endPost, int tempPos)`

หน้าที่: ทำการย้าย n disks (disk หมายเลข 1, 2, .. n) จากเสาหมายเลข startPost ไปยังเสาหมายเลข endPost โดยใช้เสาหมายเลข tempos เป็นตัวช่วย

ตัวอย่าง เช่น ถ้าเราต้องการย้าย disk 1, 2 และ 3 จากเสาต้นที่ 1 ไปยังเสาต้นที่ 3 โดยใช้เสาต้นที่ 2 เป็นตัวช่วย สามารถทำได้โดยเรียกฟังก์ชัน Move โดยส่งอาร์กิวเมนต์ดังต่อไปนี้

```
n=.....
Move(n, ........, ........, ................);
```

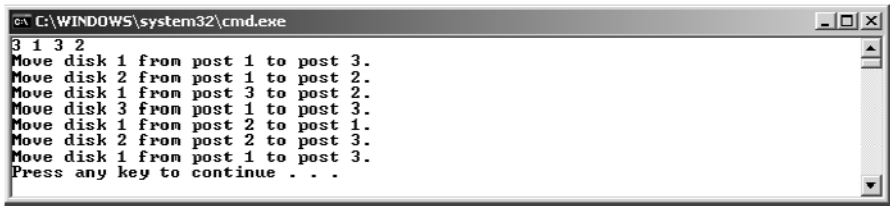
และถ้าเราต้องการย้าย disk 1 และ 2 จากเสาต้นที่ 1 ไปยังเสาต้นที่ 2 โดยใช้เสาต้นที่ 3 เป็นตัวช่วย สามารถทำได้โดยเรียกฟังก์ชัน Move โดยส่งอาร์กิวเมนต์ดังต่อไปนี้

```
n=.....
Move(n, ........, ........, ................);
```

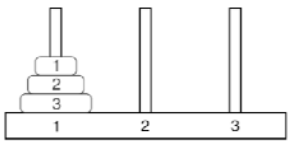
โดยใช้นิยามของฟังก์ชัน Move และแนวคิดของการแก้ปัญหาแบบ Recursion ในข้อ (ii) เราสามารถเขียนโค้ดในฟังก์ชัน Move ได้ดังนี้

```
#include <stdio.h>
void Move(int n,int startPost,int endPost,int tempPost);
int main()
{
int n,startP,endP,tempP;

       scanf("%d %d %d %d",&n,&startP,&endP,&tempP);

       Move(n, startP, endP, tempP);

       return 0;
}
void Move(int n,int startPost,int endPost,int tempPost)
{
   if(...........................)
   {
       Move(...........................,...........................,...........................,...........................);
       printf("Move disk %d from post %d to post %d.\n",n, ....................,................);
       Move(...........................,...........................,...........................,...........................);
   }
   else
    {  // n=1 (Base case)
       printf("Move disk 1 from post %d to post %d.\n",....................,................);
    }
}
```

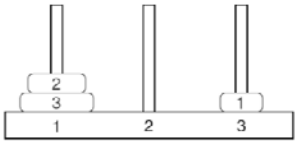
ผลลัพธ์จากการรันโปรแกรมข้างบนเมื่อป้อนข้อมูลนำเข้าเป็น 3 1 3 2 (n=3, startP=1, endP=3, tempP=2) แสดงดังข้างล่าง



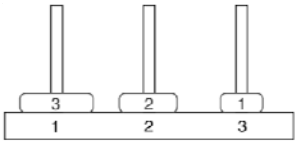
ผลเฉลย (ขั้นตอนในการเคลื่อนย้าย disk) ที่ได้จากโปรแกรมข้างบนสามารถแสดงดังรูปการเคลื่อนย้ายข้างล่าง



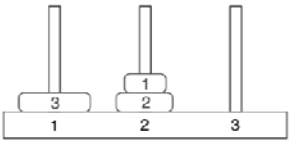
เริ่มต้น



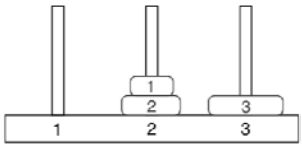
Move disk 1 from post 1 to post 3



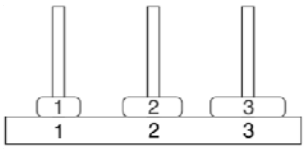
Move disk 2 from post 1 to post 2



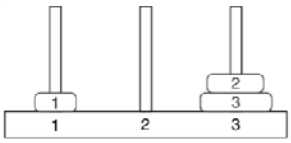
Move disk 1 from post 3 to post 2



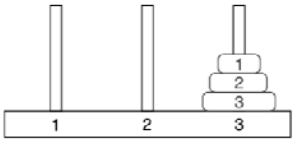
Move disk 3 from post 1 to post 3



Move disk 1 from post 2 to post 1



Move disk 2 from post 2 to post 3



Move disk 1 from post 1 to post 3

9. Fibonacci numbers

จำนวนฟีโบนักซี (Fibonacci numbers) คือจำนวนที่อยู่ในลำดับที่สร้างโดยนิยาม Recurrence equation ข้างล่างด้านซ้าย จงทำโค้ดในฟังก์ชัน Fibonacci ข้างล่างด้านขวาให้สมบูรณ์ เพื่อทำให้ฟังก์ชันเรียกตัวเอง และทำการคำนวณหา f(n) ตามสูตรข้างบน เมื่อ n คืออาร์กิวเมนต์ของฟังก์ชัน

$$f(n) = f(n-1) + f(n-2) \;\; ; \;\; \text{if } n \geq 2$$

$$\left.\begin{array}{l} f(1) = 1 \\ f(0) = 0 \end{array}\right\} \text{Best case}$$

```
int Fibonacci(int n)
{
int sum;
   if(...............................)
      ..................................................
   else if(..........................)
      ...................................
   else
   {
       sum=..............................
      return(sum);
   }
}
```

10. ปัญหา Binary search

(i) Problem statement:

กำหนดให้ข้อมูลนำเข้าคืออาร์เรย์ A แบบ int ขนาด n elements ที่เรียงจากค่าน้อยไปมากแล้ว (sorted array) นั้น คือ $A[0] \leq A[1] \leq A[2] \leq A[3] \ldots \leq A[n-2] \leq A[n-1]$ และกำหนดให้ข้อมูล b จงค้นหาว่าข้อมูล b อยู่ในอาร์เรย์ A นี้หรือไม่ (b มีค่าเท่ากับค่าใน element หนึ่งๆ ของ A หรือไม่)

ถ้าพบว่าอยู่ให้คืนค่า index ของ element ใน A ที่มีค่าเท่ากับ b แต่ถ้าไม่พบให้คืนค่า -1

ตัวอย่างเช่นอาร์เรย์ A ขนาด 11 element ดังรูปข้างล่าง ถ้าให้ b คือ 109 BinarySearch จะคืนค่า 4 แต่ถ้า b คือ 112 จะคืนค่า -1

	A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]	A[9]	A[10]
A	12	32	37	49	109	110	153	387	392	777	926

ปัญหาดังกล่าวสามารถแก้โดยไล่เปรียบเทียบตั้งแต่ ตัวแรก (A[0]) ไปเรื่อยจนถึงตัวสุดท้าย (A[10]) แล้วดูว่ามีตัวที่มีค่าเท่ากับ b หรือไม่ แต่ถ้าสมมติว่าอาร์เรย์ขนาดใหญ่มาก (10,000,000 elements) การใช้วิธีนี้จะไม่มีประสิทธิภาพ (ช้า)

โดยใช้ประโยชน์จากข้อกำหนดที่ว่าข้อมูลในอาร์เรย์ได้ถูกเรียงจากน้อยไปมาแล้ว เราจะศึกษาเทคนิคการค้นข้อมูลในอาร์เรย์ที่เรียกว่า Binary search ซึ่งใช้เทคนิค Recursion ในการแก้ปัญหา

(ii) Recursive formulation:

Input: Sorted array A และ ค่า sub-array index: start และ end และ ข้อมูลที่จะค้น b

โดยที่ start คือ index ของ element แรกของ subarray

end คือ index ของ element สุดท้ายของ subarray

ตัวอย่าง เช่น ถ้า start=0 และ end=5 จะหมายถึง subarray ในรูปข้างล่าง

Note: ถ้า start=0 และ end=n-1 แล้ว subarray นี้คืออาร์เรย์ทั้งหมดที่เป็นข้อมูลนำเข้า

<u>Step</u>

- หาจุด midpoint ของ sub-array A[start]….A[end] สมมติ index มีค่าเท่ากับ middle=(end+start)/2 ให้ทำการเปรียบเทียบ A[mid] กับ b

Case 1: ถ้า b เท่ากับ A[middle] เราเจอข้อมูลในอาร์เรย์ ดังนั้นให้ทำการคืนค่า mid (Base case)

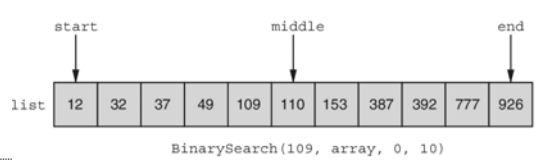
Case 2: ถ้า b น้อยกว่า A[middle]: ค้นหาใน subarray A[……..] ถึง A[..……..] (Recursive)

Case 3: ถ้า b มากกว่า A[middle]: ค้นหาใน subarray A[……..] ถึง A[……..] (Recursive)

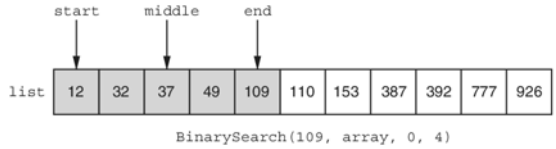
Case 4: ถ้าไม่มีข้อมูลในอาร์เรย์นั้นคือ subarray of size of 0 (n-1< 0) แสดงว่าไม่เจอ b ให้คืนค่า ……(Base case)

<u>ตัวอย่างของการค้นหา</u>เมื่อเริ่มต้นด้วย start=0 และ end=10 และ b=109

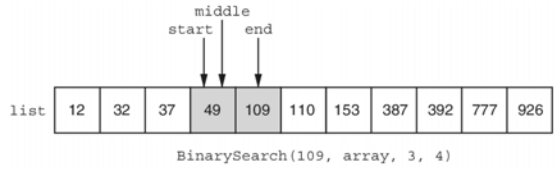
Step 1: หา midpoint ได้ middle=(0+10)/2=5 เปรียบเทียบ A[5] (110) กับ b (109) และเราพบว่า b ….. A[middle] ดังนั้นค้นหาใน subarray A[start] ถึง A[middle-1] นั้นคือ A[……] ถึง A[……..]



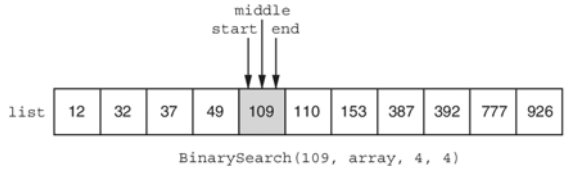
Step 2: หา midpoint ได้ middle=(0+4)/2=2 เปรียบเทียบ A[2] (37) กับ b (109) และเราพบว่า b ….. A[middle] ดังนั้นค้นหาใน subarray A[middle+1] ถึง A[end] นั้นคือ A[……] ถึง A[……..]



Step 3: หา midpoint ได้ middle =(3+4)/2=3 เปรียบเทียบ A[3] (49) กับ b (109) และเราพบว่า b ….. A[middle] ดังนั้นค้นหาใน subarray A[middle +1] ถึง A[end] นั้นคือ A[……] ถึง A[……..]



Step 4: หา midpoint ได้ middle =(4+4)/2=4 เปรียบเทียบ A[4] (109) กับ b (109) และเราพบว่า b ….. A[middle] (เจอแล้ว) ดังนั้นคืนค่า middle =2



(iii) Coding:

- กำหนดให้ฟังก์ชัน `int BinarySearch(int b, int *A, int start, int end)` จะทำการค้นหา b ใน subarray ของอาร์เรย์ A ที่ index เริ่มต้นที่ `start` และสิ้นสุดที่ end
- โดยใช้แนวคิดที่อธิบายข้างต้นสามารถเขียนโค้ดของฟังก์ชัน `BinarySearch` ได้ดังนี้

```
int BinarySearch(int b, int *A, int start, int end)
{
int middle;
    middle=.....
    if(.....) // กรณี subarray of size 0
        .....
  . else if(.....) // กรณี A[middle] เท่ากับ b
        .....
  . else if(.....) // กรณี A[middle] ..... b
        .....
  . else // กรณี A[middle] ..... b
        .....
}
```

11. โจทย์ตัวอย่าง (ข้อสอบเก่า Final ปี 2552): จงใช้เทคนิค Recursion ทำการเขียนโค้ดในฟังก์ชัน `Power` เพื่อทำการคำนวณและคืนค่า x^p โดยอาร์กิวเมนต์ x (ชนิดข้อมูลแบบ `double`) ส่วนอาร์กิวเมนต์ p (ชนิดข้อมูลแบบ `int`) และค่า p ที่จะส่งเข้ามาจะมีค่ามากกว่าหรือเท่ากับ 0 เท่านั้น

(i) จงเขียน Recurrence equation เพื่อแสดงความสัมพันธ์ x^p ในรูปแบบ Recursion	(ii) เขียนโค้ดในฟังก์ชัน `Power`

	`double Power(double x,int p)`
	`{`
	`  if(.....)`
	`  .....`
	`  else`
	`  .....`
	`}`

12. โจทย์ตัวอย่าง (ข้อสอบเก่า Final ปี 2552): จงระบุผลลัพธ์ของโปรแกรมข้างล่าง

```
#include <stdio.h>
void Magic(int n);
void main()
{
      Magic(10);
      printf("\n\n");

}
```

```
void Magic(int n)
{
      if(n<=0)
            return;
      if(n%2 == 0){
            Magic(n-1);
            printf("%d ",n);
      }
      else{
            printf("%d ",n);
            Magic(n/2);
      }
}
```