Algorithms

Chapter 6
Algorithms Involving Sequences & Sets
Part 1
(pp. 119~127)

Sequences & Sets

- Sequence
  - the order of the given elements is important
- Set
  - order isn't important
  - an element does not appear more than once

Pure Binary Search

- Problem

  Given a sorted sequence of n real numbers

  a real number z

  Find whether z appears in the sequence

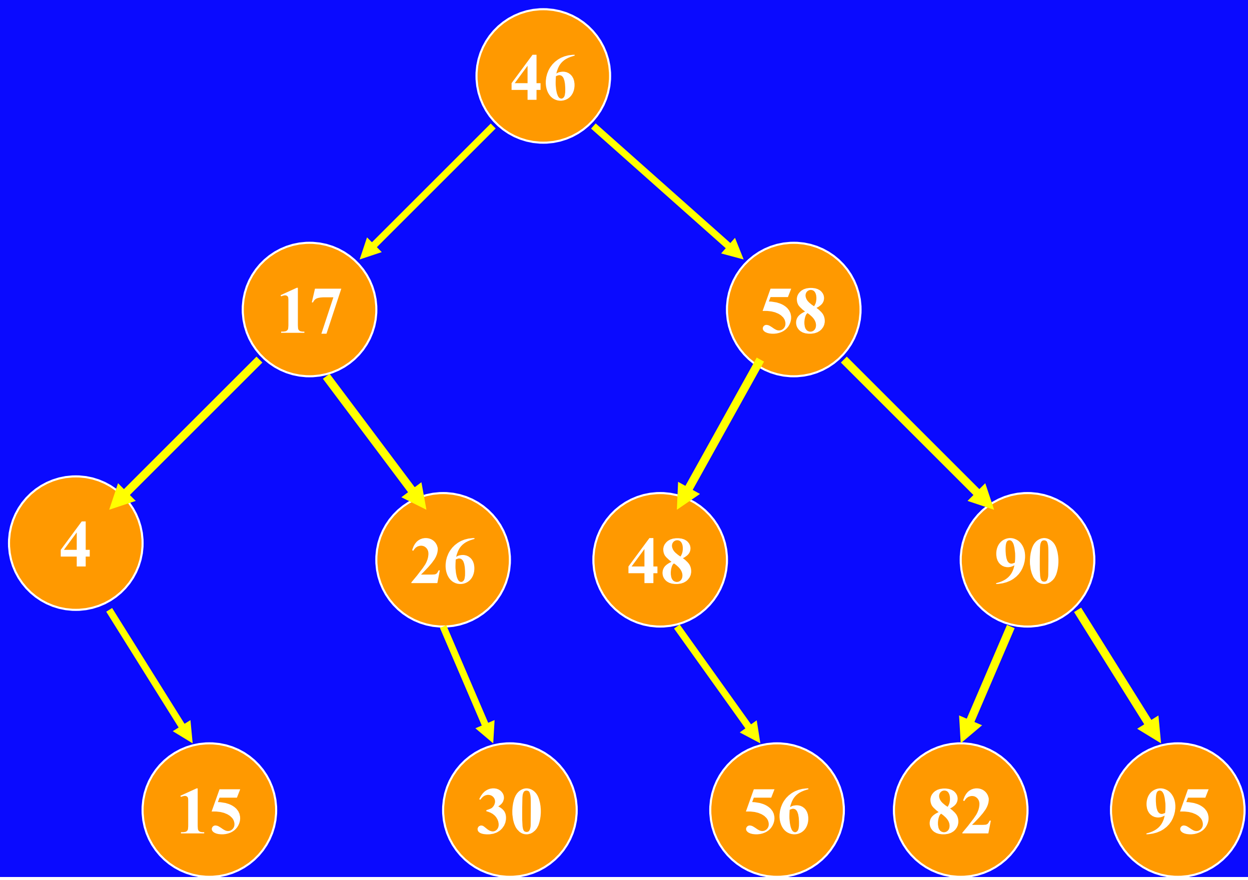
  if it does, find the position of z

- Solution
  - sequential (linear) search: O(n)
  - binary search: O(log$n$)

```
Algorithm Binary_Search(X, n, z);
Input: X (sorted array), z (search key)
Output: Position
begin
    Position:=Find(z, 1, n);
end
Function Find(z, Left, Right):integer;
begin
    if Left = Right then
        if X[Left] = z then Find:=Left
        else Find:=0
    else
        Middle:=⌈ Left+(Right-Left)/2 ⌉;
        if z < X[Middle] then
            Find:=Find(z, Left, Middle-1)
        else
            Find:=Find(z, Middle, Right)
end
```

95

1	2	3	4	5	6	7	8	9	10	11	12
4	15	17	26	30	46	48	56	58	82	90	95



M. K. Shan, CS, NCCU

Binary Search in Cyclic Sequence

Binary Search in Cyclic Sequence

- **Cyclic sorted list**

1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*
58	82	90	95	4	15	17	26	30	46	48	56

- **Problem**

  Given a cyclic sorted sequence of *n* real numbers

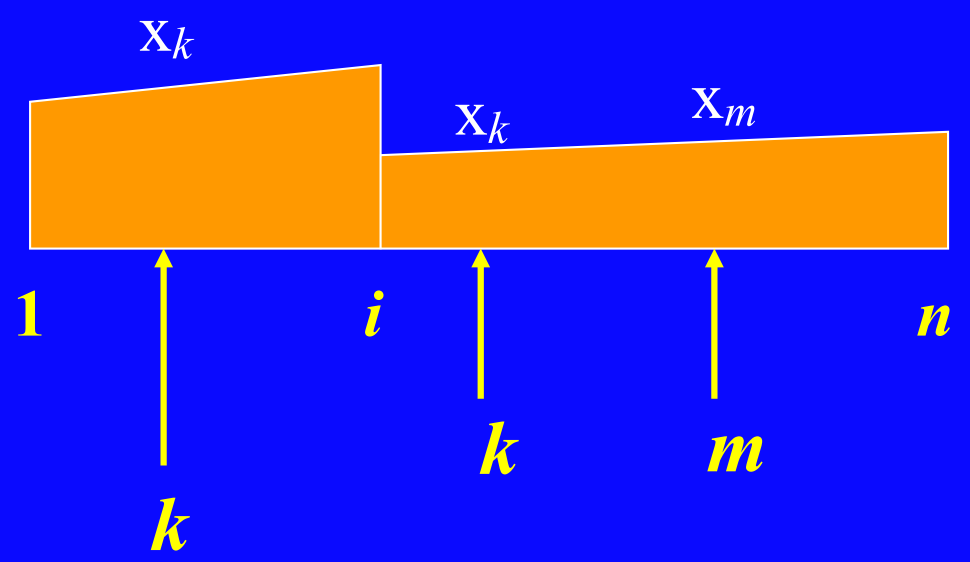
  Find the position of the minimal element in the list

- **Solution**
  - **sequential (linear) search: O(*n*)**
  - **binary search ？**

請設計O(log*n*)演算法由Cyclic Sorted Sequence中找到Minimal Element

1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*
58	82	90	95	4	15	17	26	30	46	48	56

Solution of Binary Search in Cyclic Sequence



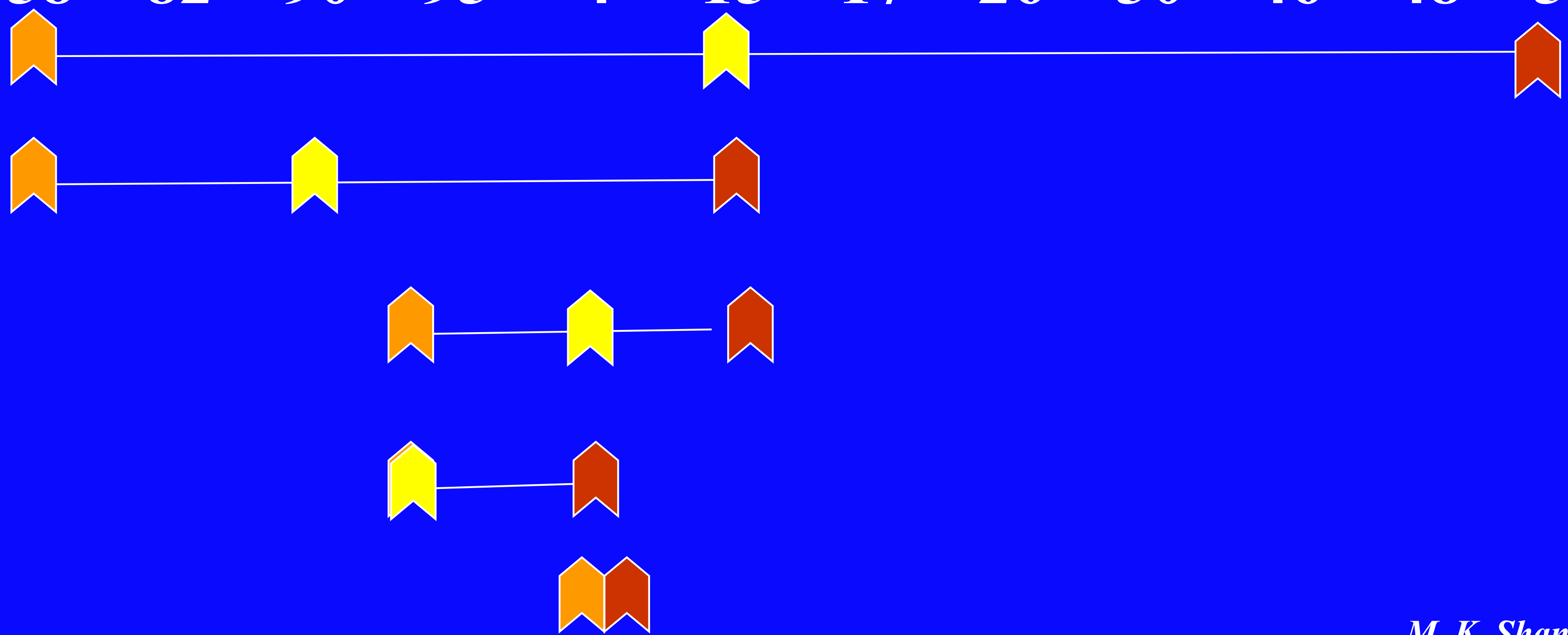
- Take any two numbers x_k & x_m, $k < m$
- If $x_k < x_m$

  then i cannot be in the range [k, m]

  else i must be in the range [k, m]

Binary Search in Cyclic Sequence

1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*
58	**82**	**90**	**95**	**4**	**15**	**17**	**26**	**30**	**46**	**48**	**56**



```
Algorithm Cyclic_Binary_Search(X, n, z);
Input: X (cyclic sorted array)
Output: Position (of the smallest element)
begin
    Position:=Find(1, n);
end
Function Cyclic_Find(Left, Right):integer;
begin
    if Left = Right then Cyclic_find:=Left
   else
        Middle:=⌊1/2(Left+Right)⌋;
        if X[Left] > X[Middle] then
            Cyclic_Find:=Cyclic_Find(Left, Middle)
        else
            Cyclic_Find:=Cyclic_Find(Middle+1, Right)
end
```

Special Binary Search

Special Binary Search

1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*
-1	0	1	2	4	5	6	8	10	12	36

- **Problem**

  Given a sorted sequence of n distinct integers $a_1, a_2, \ldots, a_n$

  Find the element $a_i = i$

- **Solution**
  - **sequential (linear) search: O(n)**
  - **binary search ?**

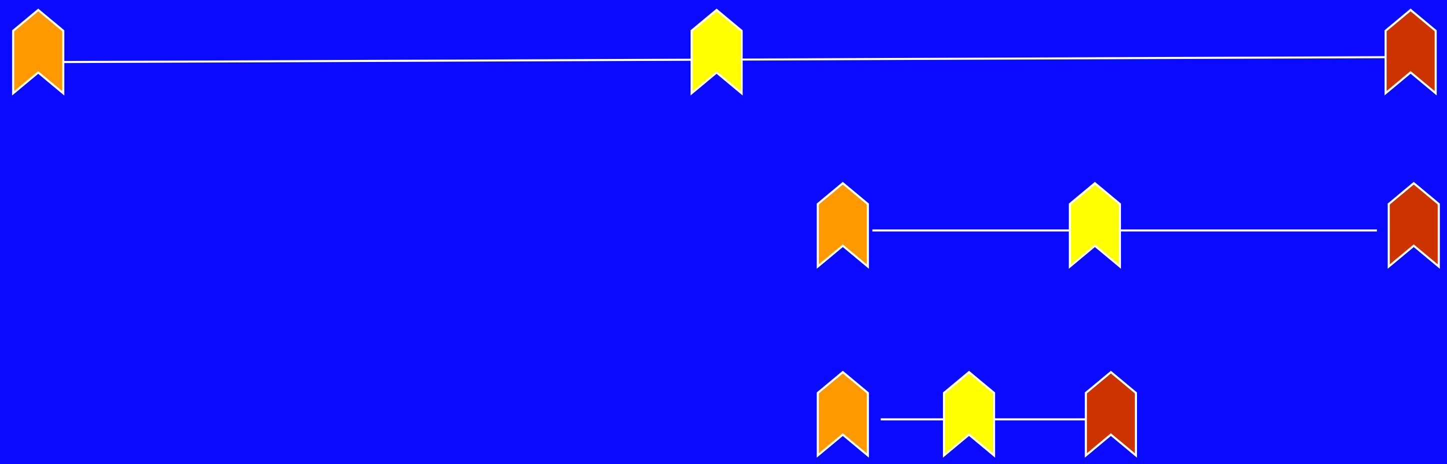
請設計O(log*n*)演算法由Sorted Sequence $a_1, a_2, \ldots, a_n$ 中找到$a_i = i$

Solution of Special Binary Search

- **Consider the value of $a_{n/2}$**
  - **if $a_{n/2}$ = n/2, done**
  - **if $a_{n/2}$ < n/2, no number in the left half satisfy**
  - **if $a_{n/2}$ > n/2, no number in the right half satisfy**

i	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*
a_i	-1	0	1	2	4	5	6	8	10	12	36

1	2	3	4	5	6	7	8	9	10	11
-1	0	1	2	4	5	6	8	10	12	36



```
Algorithm Special_Binary_Search(A, n);
Input: A (sorted array)
Output: Position
begin
    Position:=Special_Find(1, n);
end
Function Special_Find(Left, Right):integer;
begin
    if Left = Right then
        if A[Left] = Left then Special_Find:=Left
        else Special_Find:=0
    else
        Middle:=⌈1/2(Left+Right)⌉;
        if A[Middle] < Middle then
            Special_Find:=Special_Find(Middle+1, Right)
        else
            Special_Find:=Special_Find(Left, Middle)
end
```

Binary Search of Unknown Size

Binary Search of Unknown Size

- **Problem**

  Given a sorted sequence of real numbers

  a real number *z*

  Find whether z appears in the sequence

  if it does, find the position of z

Solution of Binary Search of Unknown Size

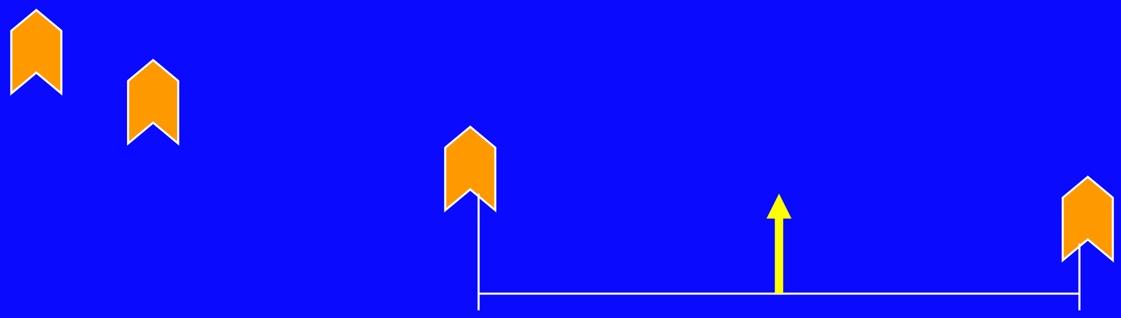
1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*
4	**15**	**17**	**26**	**30**	**46**	**48**	**56**	**58**	**82**	**90**	**95**

Where is 46 ?

請設計O(log$n$)演算法由Unknown-Sized Sorted Sequence $a_1, a_2, \ldots, a_n$ 中找到$a_i = x$

Solution of Binary Search of Unknown Size

1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*
4	15	17	26	30	46	48	56	58	82	90	95



- **Find *j* such that $x_j < z <= x_{2j}$ O(log*j*)**
- **Binary search in the range between x_j and x_{2j} O(log*j*)**
- **Total O(2*log*j*)**

Stutter-Subsequence Problem

Stuttering-Subsequence Problem

- Subsequence

Given two sequences of characters $A=a_1a_2\ldots a_n$, $B= b_1b_2\ldots b_m$

B is a subsequence of A

if there exists indices $i_1<i_2\ldots<i_m$, such that $\forall\ 1\leq j\leq m$, $b_j=a_{i_j}$

- B is a subsequence of A if
  - B can be embedded inside A in the same order
  - but with possible holes
- B=‘euec’ is a subsequence of A=‘sequence’
- Time Complexity of Subsequence Matching ?

Given two sequence of characters

A=$a_1a_2\ldots a_n$, B= $b_1b_2\ldots b_m$,

give the time complexity

to test if B is a subsequence of A

Stuttering-Subsequence Problem (cont.)

- **Stuttering**
  - **B=xyzzx**
  - **B^3=xxxyyyzzzzzzxxx**
- **Stutter-Subsequence Problem**

  Given two sequences A & B

  Find the maximal value of *i* so that

  B^i is a subsequence of A
- **Given A=axbcxdyxyacyxzyxzzyzzzxyzyxzxyx**

  B=xyzzx

  Find the maximal value of *i*

如何利用binary search的精神，
縮小stutter-subsequence problem中
*i*的搜尋範圍 ？

Solution of Stuttering-Subsequence

- **Observation**
  - **$\forall$ *i*, we can construct & test subsequence of B^i easily**
  - **if B^j is a subsequence of A,**
    then B^i is a subsequence of A, $\forall\ 1 \le i \le j$
- **Solution**
  - **Check whether B^i is a subsequence of A, i=(n/m)/2**
  - **if yes, eliminating the lower range**
  - **otherwise, eliminating the upper range**
  - **time complexity: O((n+m)log(n/m))**

Summary of Idea

- Whenever looking for the maximal *i* that satisfy some property
  - it may be sufficient to find an algorithm that determines whether a given *i* satisfy
  - we can do the rest by binary search if we have an upper bound for *i*
  - if do not know the upper bound, use doubling scheme

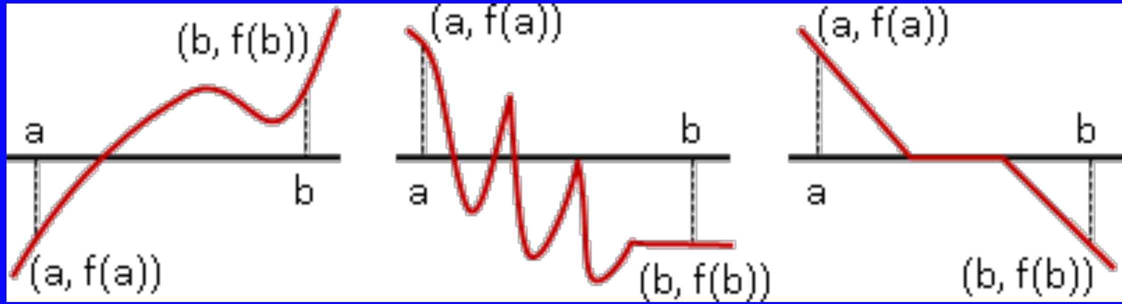
Solving Equations

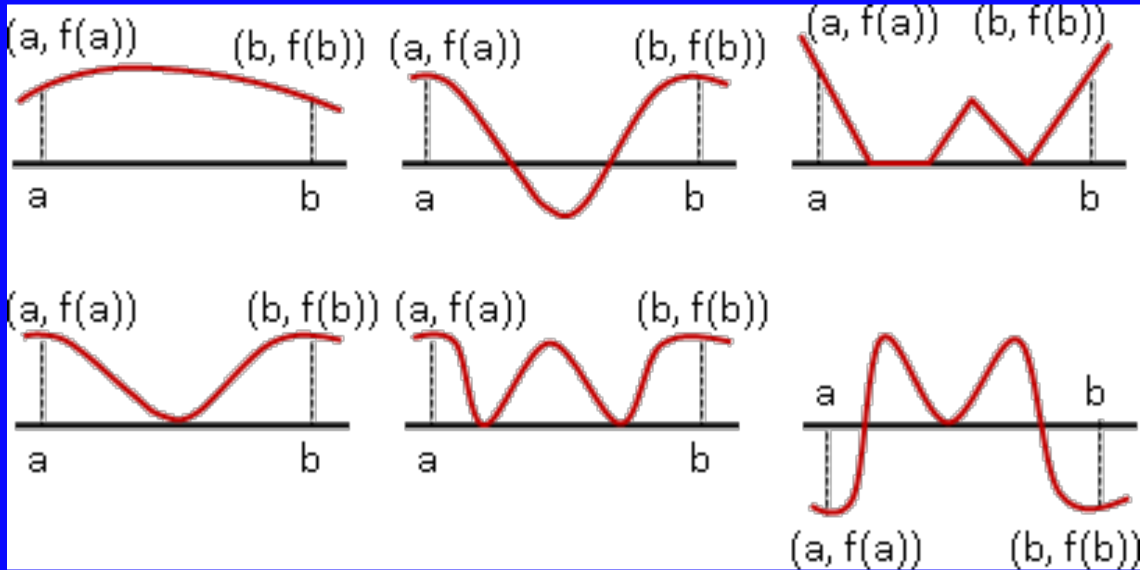
Solving Equations

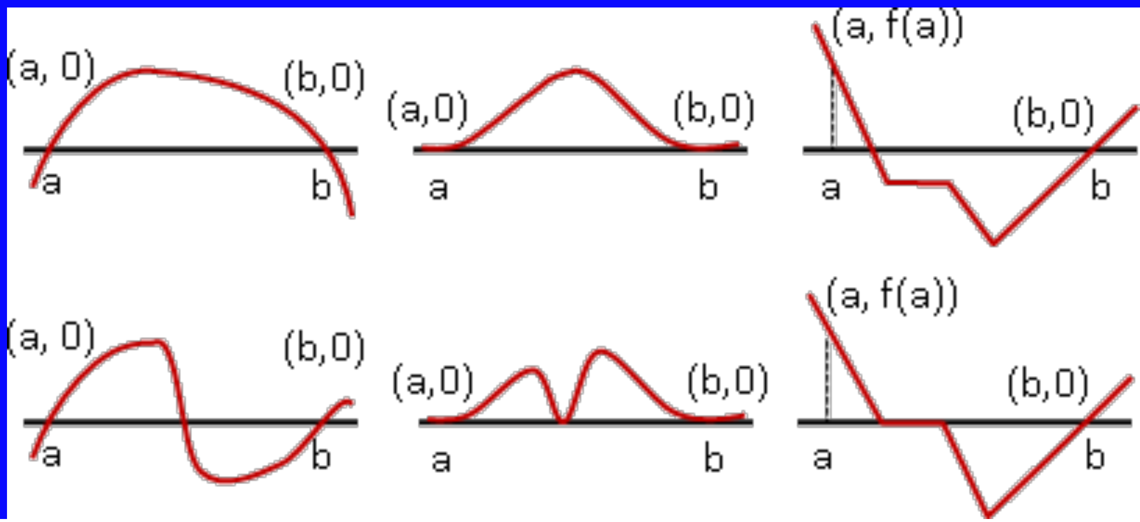
- To find the root x, such that $f(x)=0$
- To find the root of $f(x)= x^3-3x^2+3x-1$
  - $f(x)= x^3-3x^2+3x-1 = (x-1)^3$
  - x = root of $f(x)= 1$

- To find the root of $f(x)= x^3-x-2$
  - Exhaustive approach: try all possible x
  - Better approach to reduce search space ?

如何利用binary search的精神，縮小equation root finding的搜尋範圍？

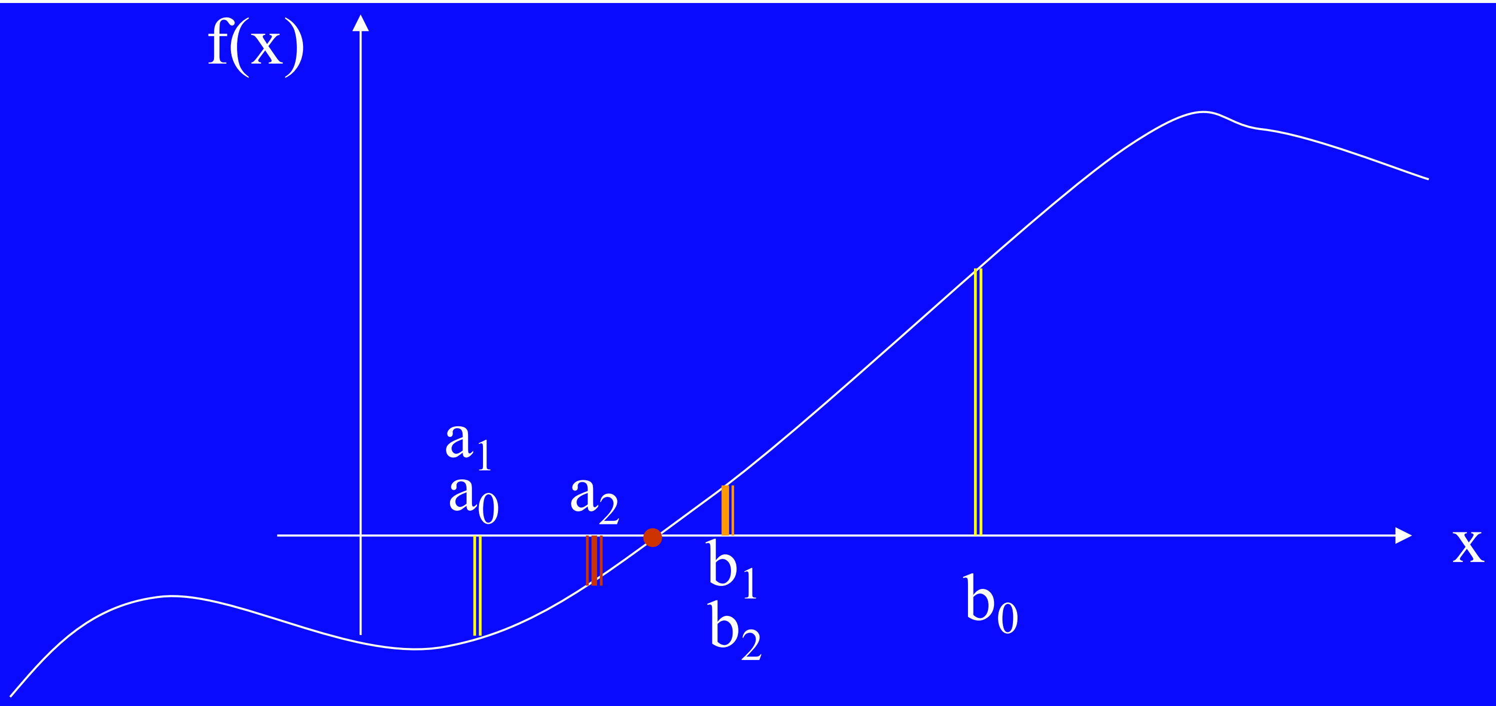

(b, f(b))
a
b
(a, f(a))
(a, f(a))
b
a
(b, f(b))
(a, f(a))
b
a
(b, f(b))


(a, f(a))
(b, f(b))
a
b
(a, f(a))
(b, f(b))
a
b
(a, f(a))
(b, f(b))
a
b
(a, f(a))
(b, f(b))
a
b
(a, f(a))
(b, f(b))
a
b
a
b
(a, f(a))
(b, f(b))


(a, 0)
(b,0)
a
b
(a,0)
(b,0)
a
b
(a, f(a))
(b,0)
a
b
(a, 0)
(b,0)
a
b
(a,0)
(b,0)
a
b
(a, f(a))
(b,0)
a
b

```
INPUT: Function f, endpoint values a, b, tolerance TOL, maximum iterations NMAX
CONDITIONS: a < b, either f(a) < 0 and f(b) > 0 or f(a) > 0 and f(b) < 0
OUTPUT: value which differs from a root of f(x)=0 by less than TOL

N ← 1
While N ≤ NMAX # limit iterations to prevent infinite loop
  c ← (a + b)/2 # new midpoint
  If f(c) = 0 or (b – a)/2 < TOL then # solution found
    Output(c)
    Stop
  EndIf
  N ← N + 1 # increment step counter
  If sign(f(c)) = sign(f(a)) then a ← c else b ← c # new interval
EndWhile
Output("Method failed.") # max number of steps exceeded
```



$f(x)=x^3-x-2$

Iteration	a_n	b_n	c_n	$f(c_n)$
1	1	2	1.5	−0.125
2	1.5	2	1.75	1.6093750
3	1.5	1.75	1.625	0.6660156
4	1.5	1.625	1.5625	0.2521973
5	1.5	1.5625	1.5312500	0.0591125
6	1.5	1.5312500	1.5156250	−0.0340538
7	1.5156250	1.5312500	1.5234375	0.0122504
8	1.5156250	1.5234375	1.5195313	−0.0109712
9	1.5195313	1.5234375	1.5214844	0.0006222
10	1.5195313	1.5214844	1.5205078	−0.0051789
11	1.5205078	1.5214844	1.5209961	−0.0022794
12	1.5209961	1.5214844	1.5212402	−0.0008289
13	1.5212402	1.5214844	1.5213623	−0.0001034
14	1.5213623	1.5214844	1.5214233	0.0002594
15	1.5213623	1.5214233	1.5213928	0.0000780

$f(a_n)$	$f(b_n)$
-2	4
-0.125	4
-0.125	1.6093750

Interpolation Search

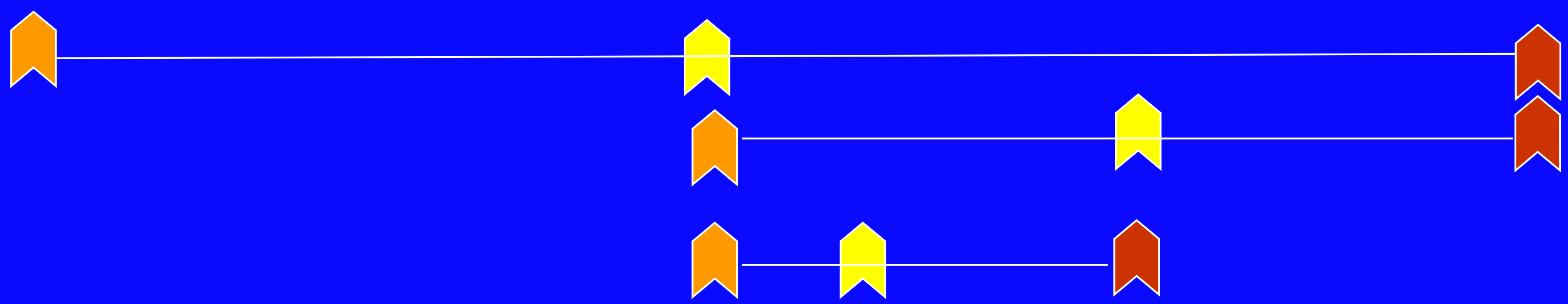
Interpolation Search

- **Observation of binary search**

if during the search we find a value close to the search number z, it seems more reasonable to continue search in that neighbor, instead of going to the next half point

Search 48

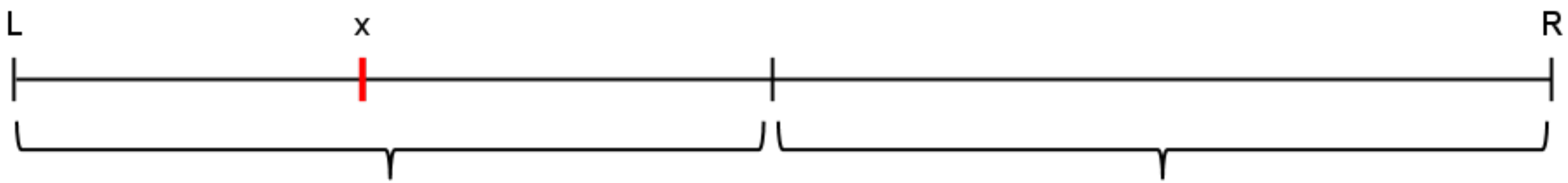
0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*
4	15	17	26	30	46	48	56	58	82	90	95

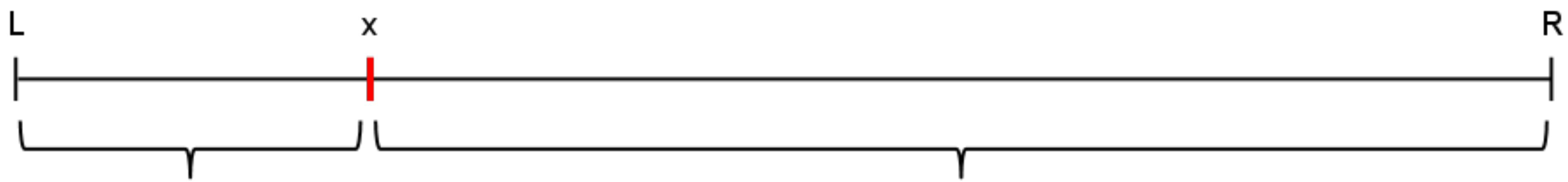


有可能搜尋範圍縮小比例不只1/2嗎？

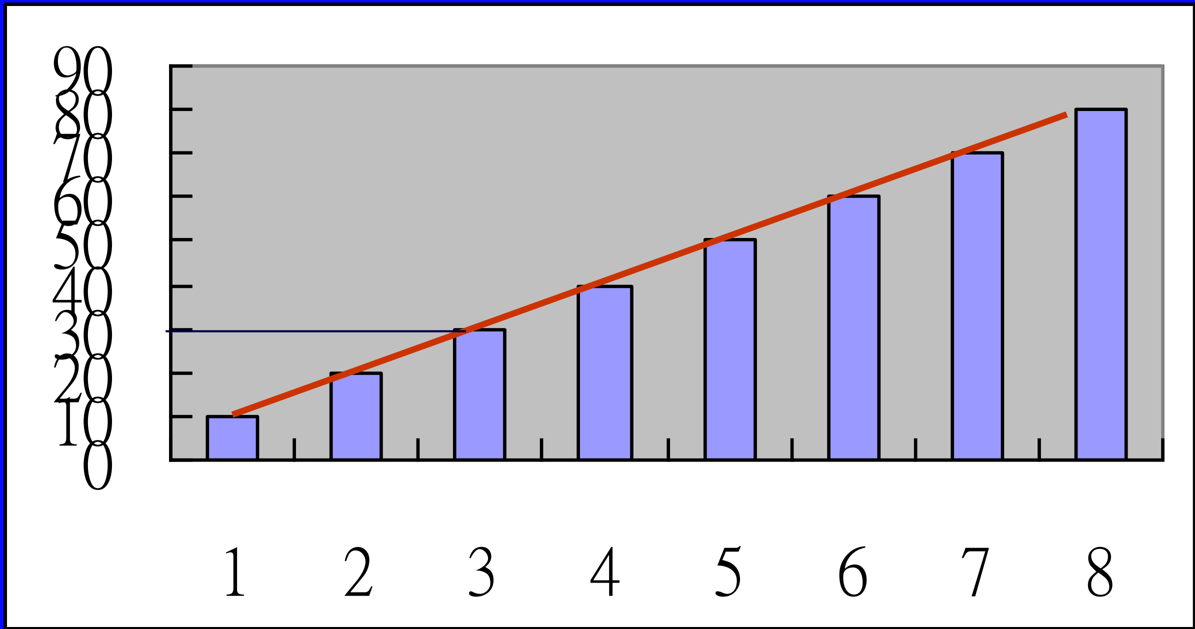
Interpolation Search (cont.)

- **Basic idea of interpolation search**
  - **Instead of cutting the search space by a fixed half, cut it by an amount that seems the most likely to succeed.**
  - **Amount is determined by interpolation**





Interpolation Search (cont.)

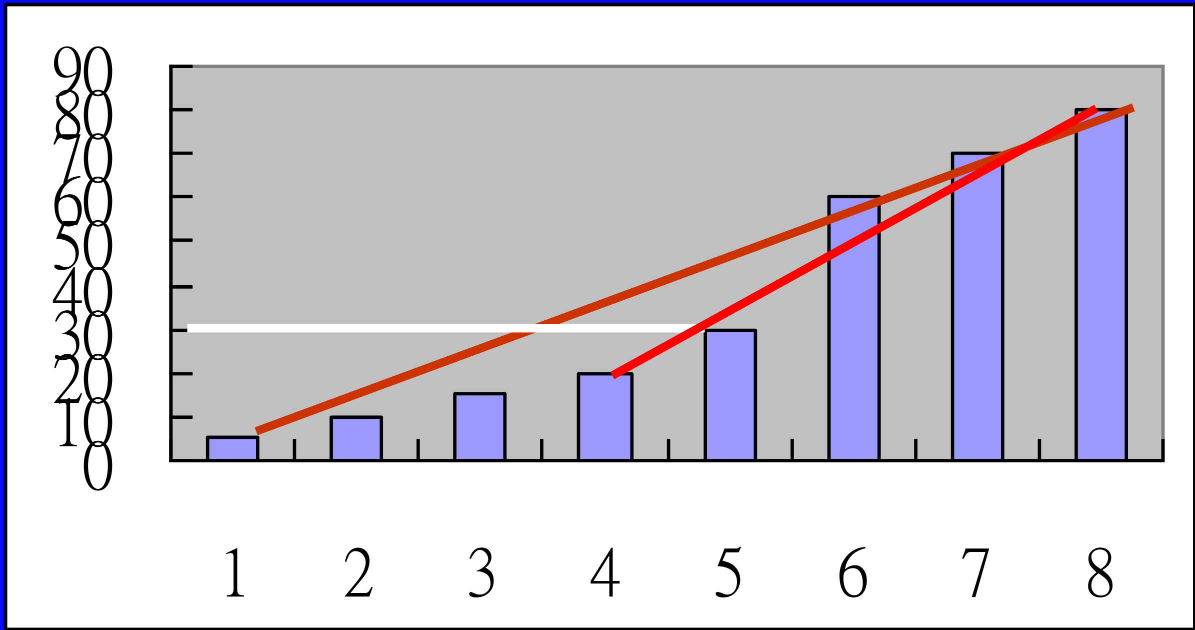


10 20 30 40 50 60 70 80

Search 30

1+[(30-10)/(80-10)]*(8-1)=1+(20/70)*7

Interpolation Search (cont.)



5 10 15 20 30 60 70 80

Search 30

1+[(30-5)/(80-5)]*(8-1)=1+(25/75)*7=10/3 ~ 4

4+[(30-20)/(80-20)]*(8-4)=4+(10/60)*4=28/6 ~5

M. K. Shan, CS, NCCU

```
Algorithm Interpolation_Search(X, n, z);
Input: X (cyclic sorted array), z
Output: Position (of the smallest element)
begin
   if z < X[1] or Z > X[n] then Position:=0
   else Position := Int_Find(z, 1, n)
end
Function Int_Find(z, Left, Right):integer;
begin
   if X[Left] = z then Int_Find:=Left
   else if Left = Right or X[Left] = X[Right] then
         Int_Find:=0
   else  Next_Guess:=
```

$$\left\lceil \text{Left} + \frac{(z - X[\text{Left}])}{X[\text{Right}] - X[\text{Left}]}(\text{Right} - \text{Left}) \right\rceil$$

```
          if X[Middle] < X[Right] then
             Int_Find:=Int_Find(z, Left, Next_Guess-1)
          else
             Int_Find:=Int_Find(z, Next-Guess, Right)
end
```