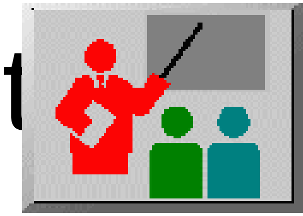
Welcome To . . .

Data Structures

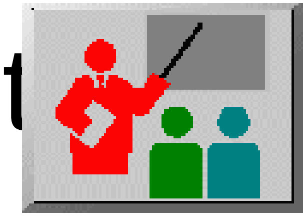
王惠嘉

What The Course Is About

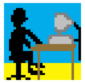


- Data structures is concerned with the representation and manipulation of data.
- All programs manipulate data.
- So, all programs represent data in some way.
- Data manipulation requires an algorithm.

What The Course Is About



- We shall study ways to represent data and algorithms to manipulate these representations.

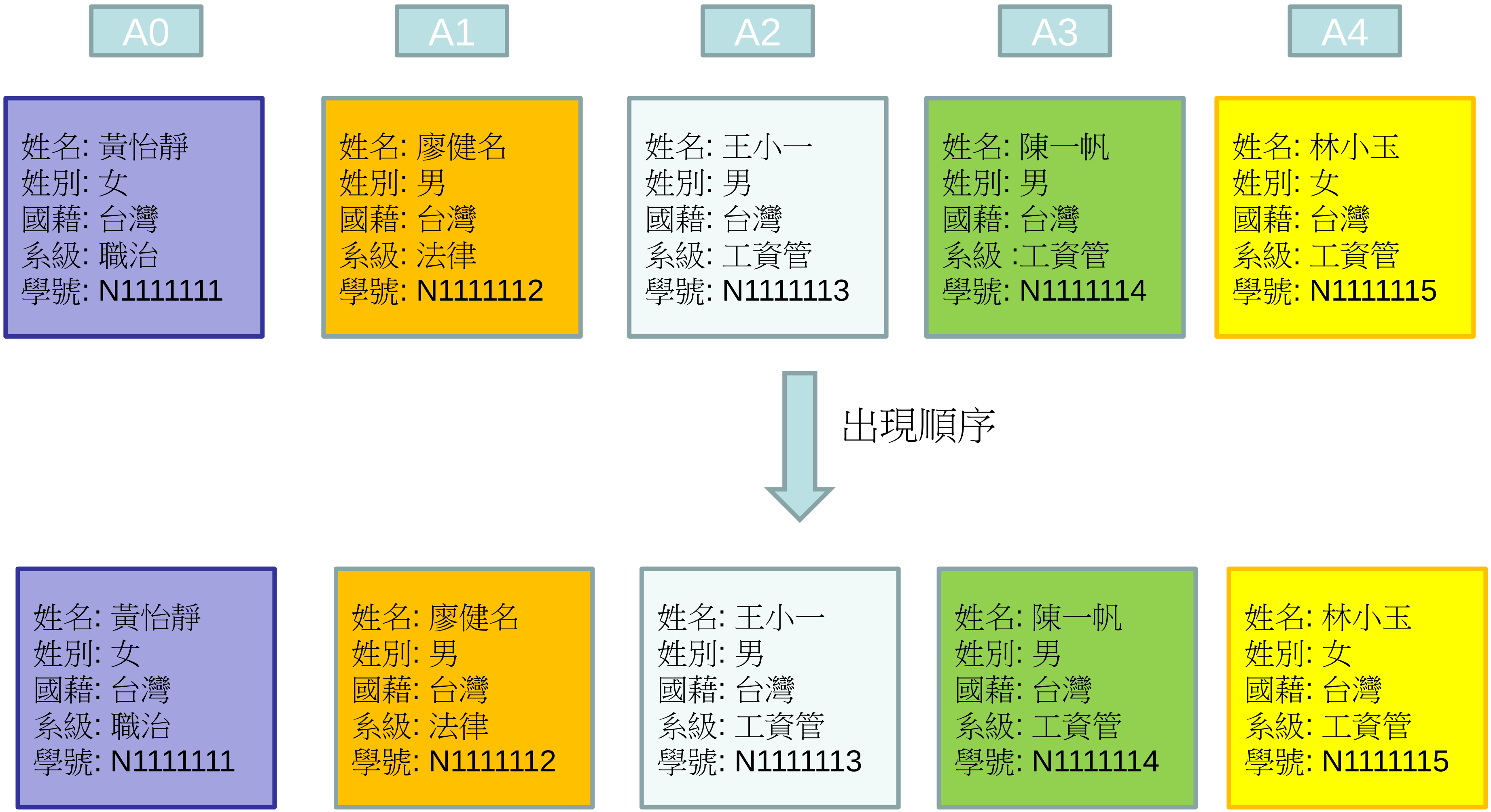


- The study of data structures is fundamental to Management, Science & Engineering.

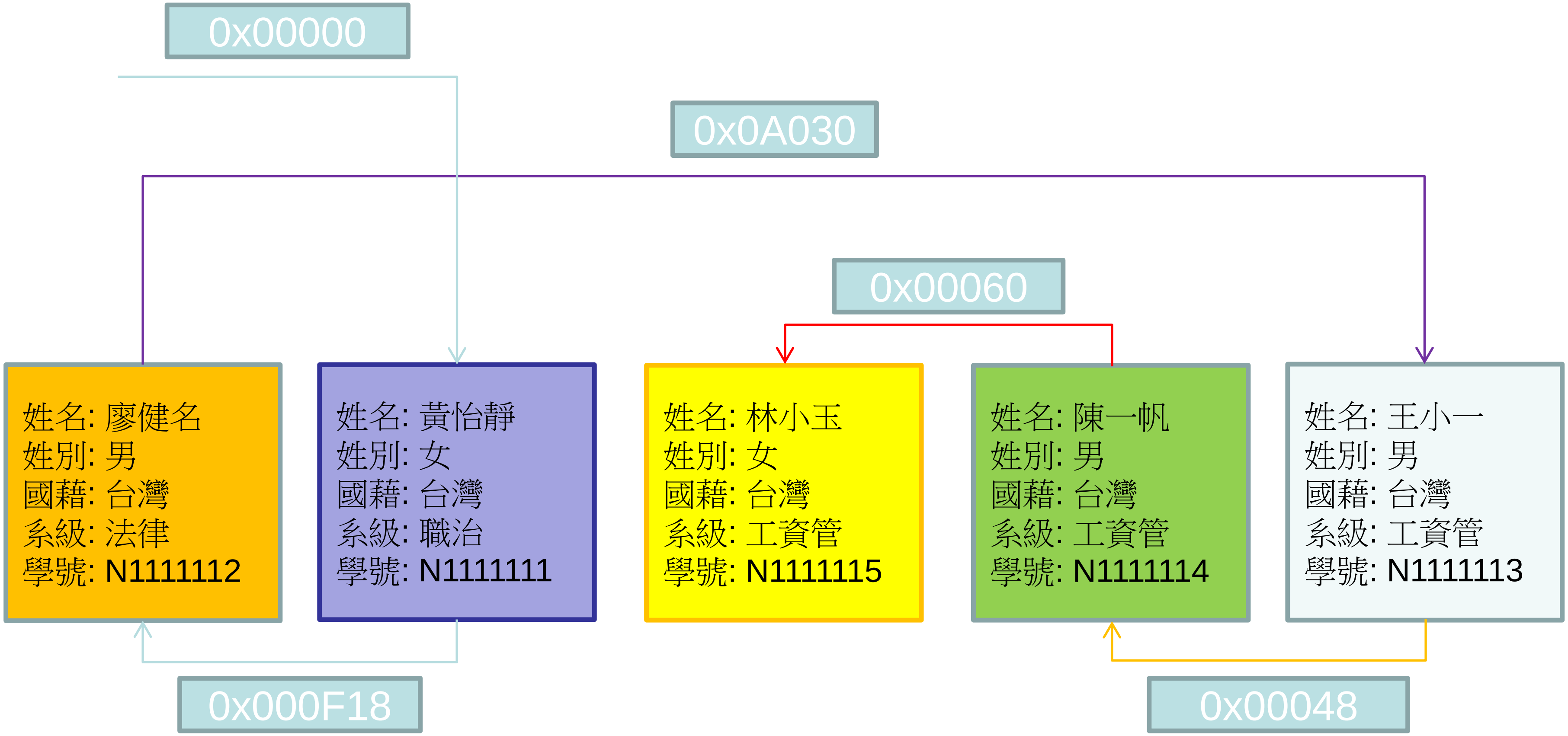
Some examples

- Data Structure: Array or Link
  - Restaurant or Party Games arrangement: Pros & Cons
- Algorithm
  - Sort
  - Insert
  - Insertion Sort
- Complexity

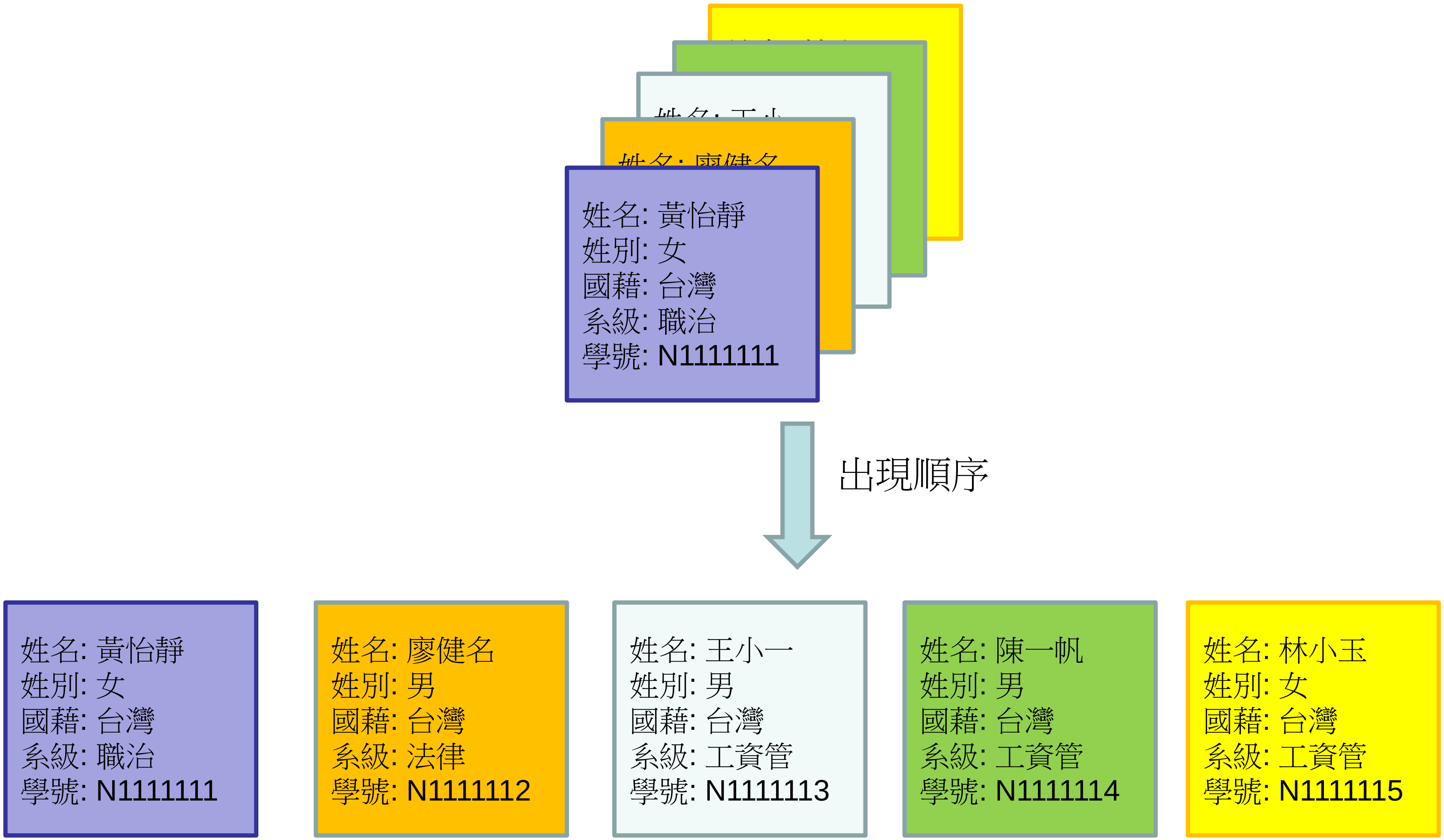
Array


A0
A1
A2
A3
A4
姓名: 黃怡靜
姓別: 女
國藉: 台灣
系級: 職治
學號: N1111111
姓名: 廖健名
姓別: 男
國藉: 台灣
系級: 法律
學號: N1111112
姓名: 王小一
姓別: 男
國藉: 台灣
系級: 工資管
學號: N1111113
姓名: 陳一帆
姓別: 男
國藉: 台灣
系級 :工資管
學號: N1111114
姓名: 林小玉
姓別: 女
國藉: 台灣
系級: 工資管
學號: N1111115
出現順序
姓名: 黃怡靜
姓別: 女
國藉: 台灣
系級: 職治
學號: N1111111
姓名: 廖健名
姓別: 男
國藉: 台灣
系級: 法律
學號: N1111112
姓名: 王小一
姓別: 男
國藉: 台灣
系級: 工資管
學號: N1111113
姓名: 陳一帆
姓別: 男
國藉: 台灣
系級: 工資管
學號: N1111114
姓名: 林小玉
姓別: 女
國藉: 台灣
系級: 工資管
學號: N1111115

Link Lists


0x00000
0x0A030
0x00060
姓名: 廖健名
姓別: 男
國籍: 台灣
系級: 法律
學號: N1111112
姓名: 黃怡靜
姓別: 女
國籍: 台灣
系級: 職治
學號: N1111111
姓名: 林小玉
姓別: 女
國籍: 台灣
系級: 工資管
學號: N1111115
姓名: 陳一帆
姓別: 男
國籍: 台灣
系級: 工資管
學號: N1111114
姓名: 王小一
姓別: 男
國籍: 台灣
系級: 工資管
學號: N1111113
0x000F18
0x00048

Stacks(Last in First out)



Queues(First in First out)

姓名: 黃怡靜
姓別: 女
國藉: 台灣
系級: 職治
學號: N1111111

姓名: 廖健名
姓別: 男
國藉: 台灣
系級: 法律
學號: N1111112

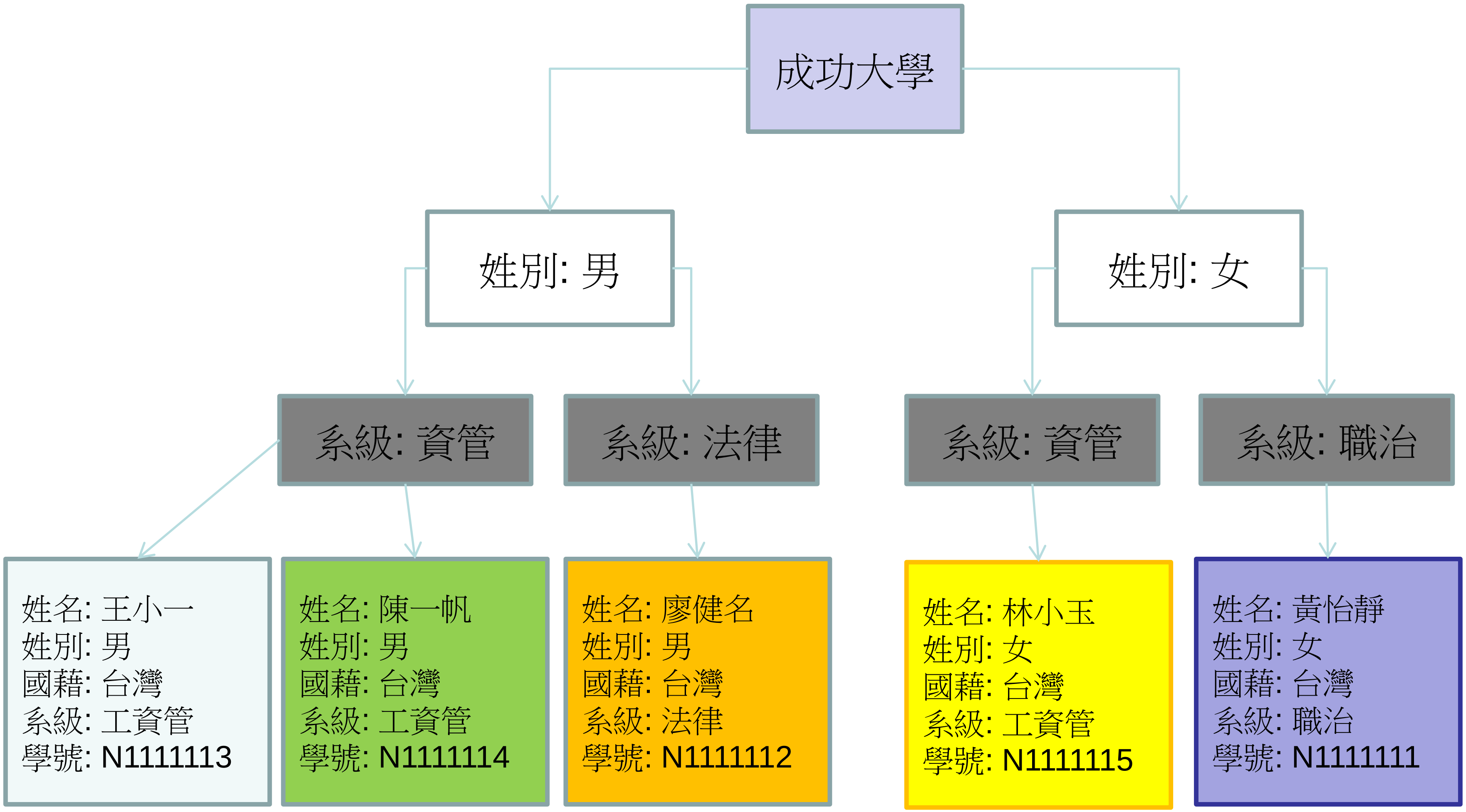
姓名: 王小一
姓別: 男
國藉: 台灣
系級: 工資管
學號: N1111113

姓名: 陳一帆
姓別: 男
國藉: 台灣
系級: 工資管
學號: N1111114

姓名: 林小玉
姓別: 女
國藉: 台灣
系級: 工資管
學號: N1111115

出現順序

Trees



成功大學
姓別: 男
姓別: 女
系級: 資管
系級: 法律
系級: 資管
系級: 職治
姓名: 王小一
姓別: 男
國藉: 台灣
系級: 工資管
學號: N1111113
姓名: 陳一帆
姓別: 男
國藉: 台灣
系級: 工資管
學號: N1111114
姓名: 廖健名
姓別: 男
國藉: 台灣
系級: 法律
學號: N1111112
姓名: 林小玉
姓別: 女
國藉: 台灣
系級: 工資管
學號: N1111115
姓名: 黃怡靜
姓別: 女
國藉: 台灣
系級: 職治
學號: N1111111

Sorting

- Rearrange a[0], a[1], …, a[n-1] into ascending order. When done, a[0] <= a[1] <= … <= a[n-1]
- 8, 6, 9, 4, 3 => 3, 4, 6, 8, 9

Insert An Element

- Given a sorted list/sequence, insert a new element
- Given 3, 6, 9, 14
- Insert 5
- Result 3, 5, 6, 9, 14

Insert an Element

- 3, 6, 9, 14   insert 5
- Compare new element (5) and last one (14)
- Shift 14 right to get 3, 6, 9, , 14
- Shift 9 right to get 3, 6, , 9, 14
- Shift 6 right to get 3, , 6, 9, 14
- Insert 5 to get 3, 5, 6, 9, 14

Insert An Element

// insert t into a[0:i-1] find a place from i-1->0

```
j = i-1
while j >= 0 and temp < a[j]:
    a[j+1] = a[j]
    j += -1
a[j+1] = temp
```

Insertion Sort

- Start with a sequence of size 1
- Repeatedly insert remaining elements

Insertion Sort

- Sort 7, 3, 5, 6, 1
- Start with 7 and insert 3 => 3, 7
- Insert 5 => 3, 5, 7
- Insert 6 => 3, 5, 6, 7
- Insert 1 => 1, 3, 5, 6, 7

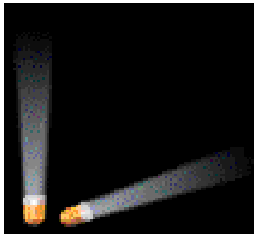
Insertion Sort Algorithm

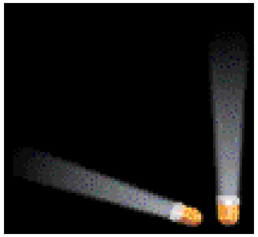
```
for i in range(1, len(a)):
    # insert a[i] into a[0:i-1]
    # code to insert comes here
    ...

```

Insertion Sort Algorithm

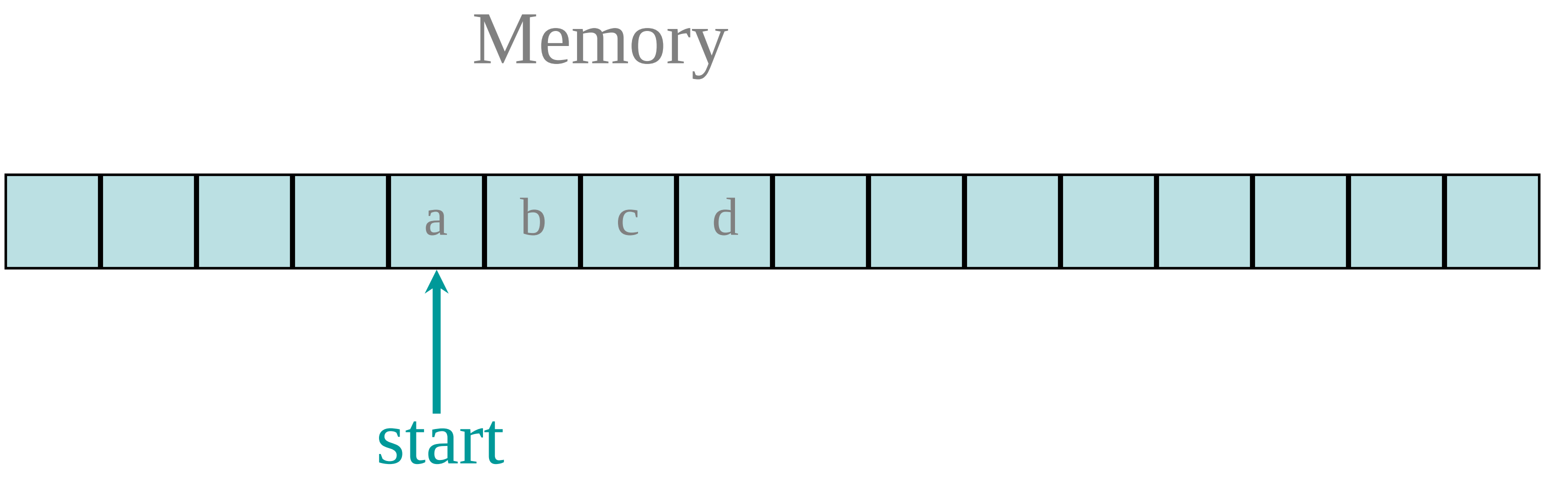
```
for i in range(1, len(a)):
    # insert a[i] into a[0:i-1]
    temp = a[i]
    j = i-1
    while j >= 0 and temp < a[j]:
        a[j+1] = a[j]
        j += -1
    a[j+1] = temp
```

Arrays



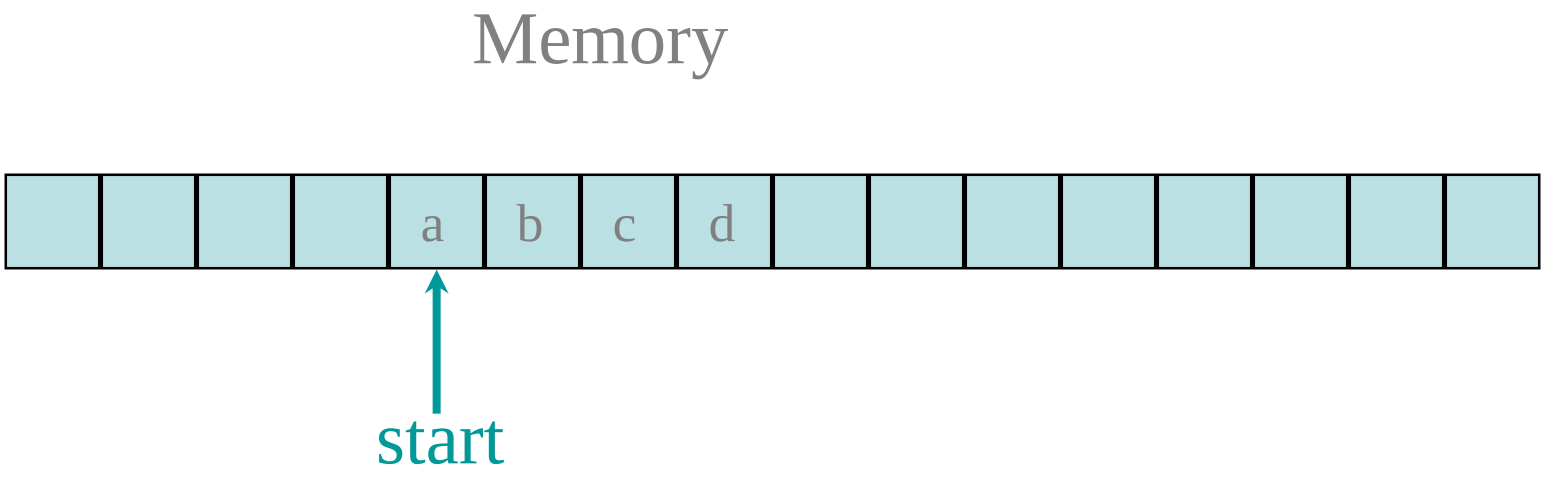


1D Array Representation In C++



- 1-dimensional array x = [a, b, c, d]
- map into contiguous memory locations
- location(x[i]) = start + i

Space Overhead



space overhead = 4 bytes for start

(excludes space needed for the elements of x)

2D Arrays (需check python的2D情況)

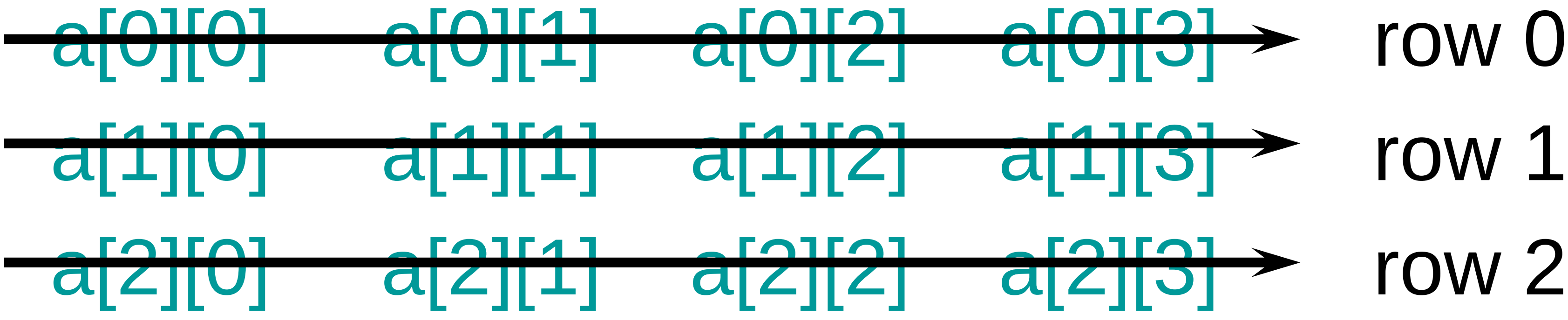
The elements of a 2-dimensional array a declared as:

a = [['00','01','02','03'],['10','11','12','13'],['20','21','22','23']]

may be shown as a table

a[0][0]	a[0][1]	a[0][2]	a[0][3]
a[1][0]	a[1][1]	a[1][2]	a[1][3]
a[2][0]	a[2][1]	a[2][2]	a[2][3]

Rows Of A 2D Array



Columns Of A 2D Array

a[0][0]	a[0][1]	a[0][2]	a[0][3]
a[1][0]	a[1][1]	a[1][2]	a[1][3]
a[2][0]	a[2][1]	a[2][2]	a[2][3]
column 0	column 1	column 2	column 3

2D Array Representation In C++

2-dimensional array x

a, b, c, d

e, f, g, h

i, j, k, l

view 2D array as a 1D array of rows

x = [row0, row1, row 2]

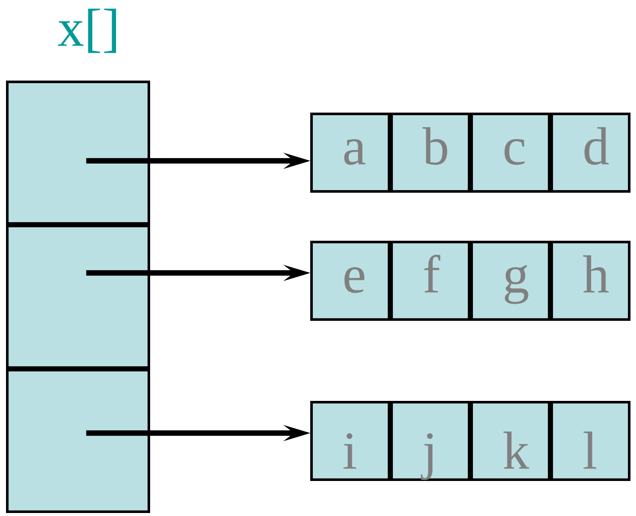
row 0 = [a,b, c, d]

row 1 = [e, f, g, h]

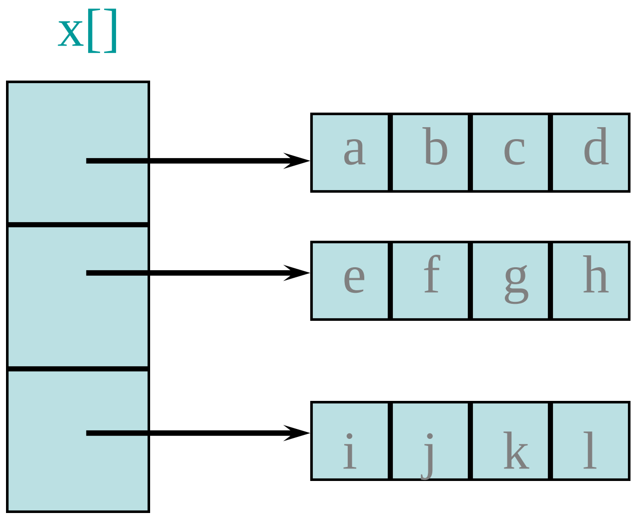
row 2 = [i, j, k, l]

and store as 4 1D arrays

2D Array Representation In C++



Space Overhead



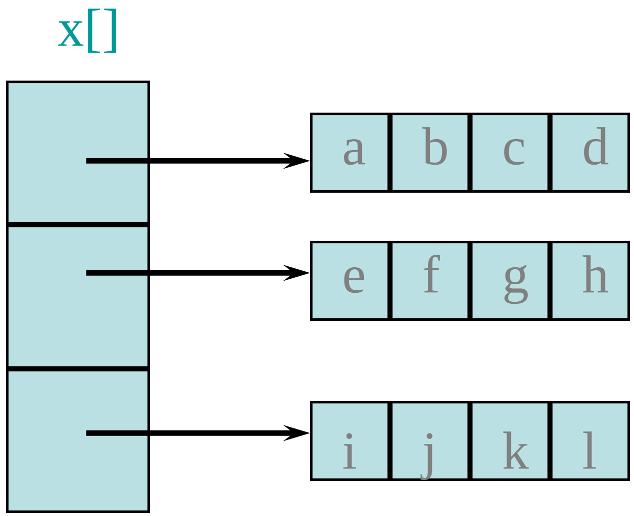
space overhead = overhead for 4 1D arrays

= 4 * 4 bytes

= 16 bytes

= (number of rows + 1) x 4 bytes

Array Representation In C++



- This representation is called the array-of-arrays representation.
- Requires contiguous memory of size 3, 4, 4, and 4 for the 4 1D arrays.
- 1 memory block of size number of rows and number of rows blocks of size number of columns

Row-Major Mapping

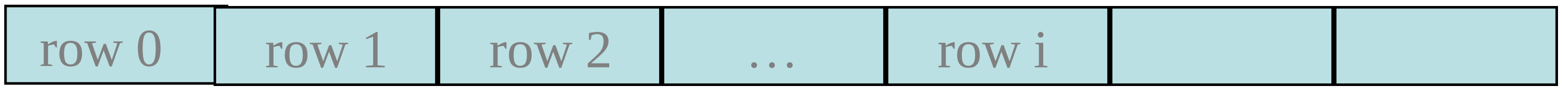
- Example 3 x 4 array:

a b c d
e f g h
i j k l

- Convert into 1D array y by collecting elements by rows.
- Within a row elements are collected from left to right.
- Rows are collected from top to bottom.
- We get {a, b, c, d, e, f, g, h, i, j, k, l}

row 0	row 1	row 2	…	row i		

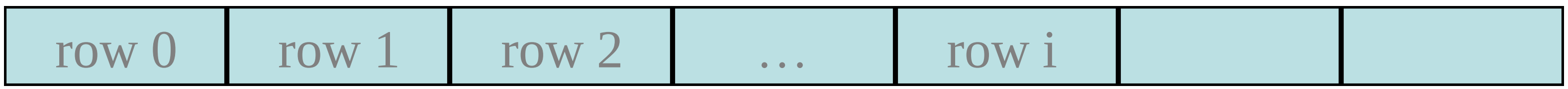
Locating Element x[i][j]



- assume x has r rows and c columns
- each row has c elements
- i rows to the left of row i
- so ic elements to the left of x[i][0]
- so x[i][j] is mapped to position

  ic + j of the 1D array

Space Overhead



4 bytes for start of 1D array +

4 bytes for c (number of columns)

= 8 bytes

Disadvantage

Need contiguous memory of size rc.

Column-Major Mapping

a b c d

e f g h

i j k l

- Convert into 1D array y by collecting elements by columns.
- Within a column elements are collected from top to bottom.
- Columns are collected from left to right.
- We get y = {a, e, i, b, f, j, c, g, k, d, h, l}

Matrix

Table of values. Has rows and columns, but numbering begins at 1 rather than 0.

a b c d	row 1
e f g h	row 2
i j k l	row 3

- Use notation x(i,j) rather than x[i][j].
- May use a 2D array to represent a matrix.

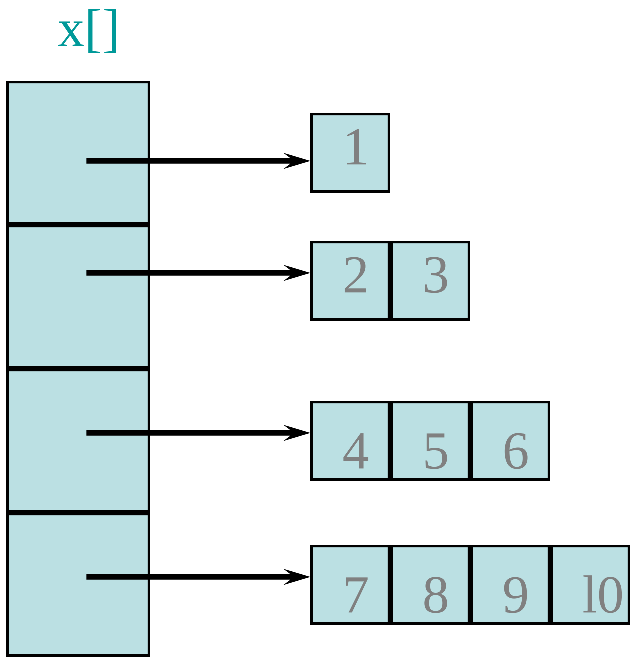
Lower Triangular Matrix

An $n \times n$ matrix in which all nonzero terms are either on or below the diagonal.

1	0	0	0
2	3	0	0
4	5	6	0
7	8	9	10

- $x(i,j)$ is part of lower triangle iff $i >= j$.
- number of elements in lower triangle is $1 + 2 + \ldots + n = n(n+1)/2$.
- store only the lower triangle

Array Of Arrays Representation



Use an irregular 2-D array … length of rows is not required to be the same.

Creating And Using An Irregular Array

```
# declare a two-dimensional array variable
# and allocate the desired number of rows
irregular_array = [0] * number_of_rows

# now allocate space for the elements in each row
for i in range(0, number_of_rows):
    irregular_array[i] = [0] * length[i]

# use the array like any regular array
irregular_array[2][3] = 5
irregular_array[4][6] = irregular_array[2][3] + 2
irregular_array[1][1] += 3
```

Map Lower Triangular Array Into A 1D Array

Use row-major order, but omit terms that are not part of the lower triangle.

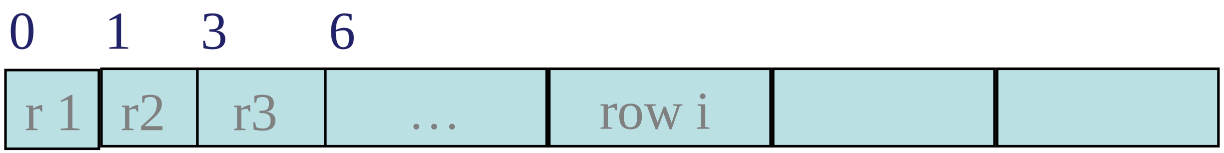
For the matrix

1	0	0	0
2	3	0	0
4	5	6	0
7	8	9	10

we get

1, 2, 3, 4, 5, 6, 7, 8, 9, 10

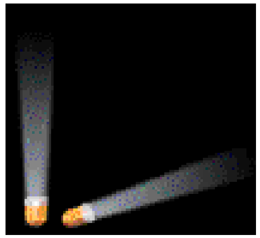
Index Of Element [i][j]

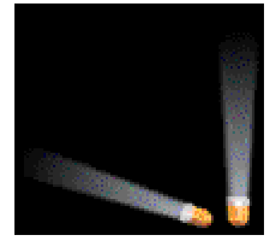


- Order is: row 1, row 2, row 3, …
- Row i is preceded by rows 1, 2, …, i-1
- Size of row i is i.
- Number of elements that precede row i is

  $1 + 2 + 3 + \ldots + i-1 = i(i-1)/2$
- So element (i,j) is at position $i(i-1)/2 + j - 1$ of the 1D array.

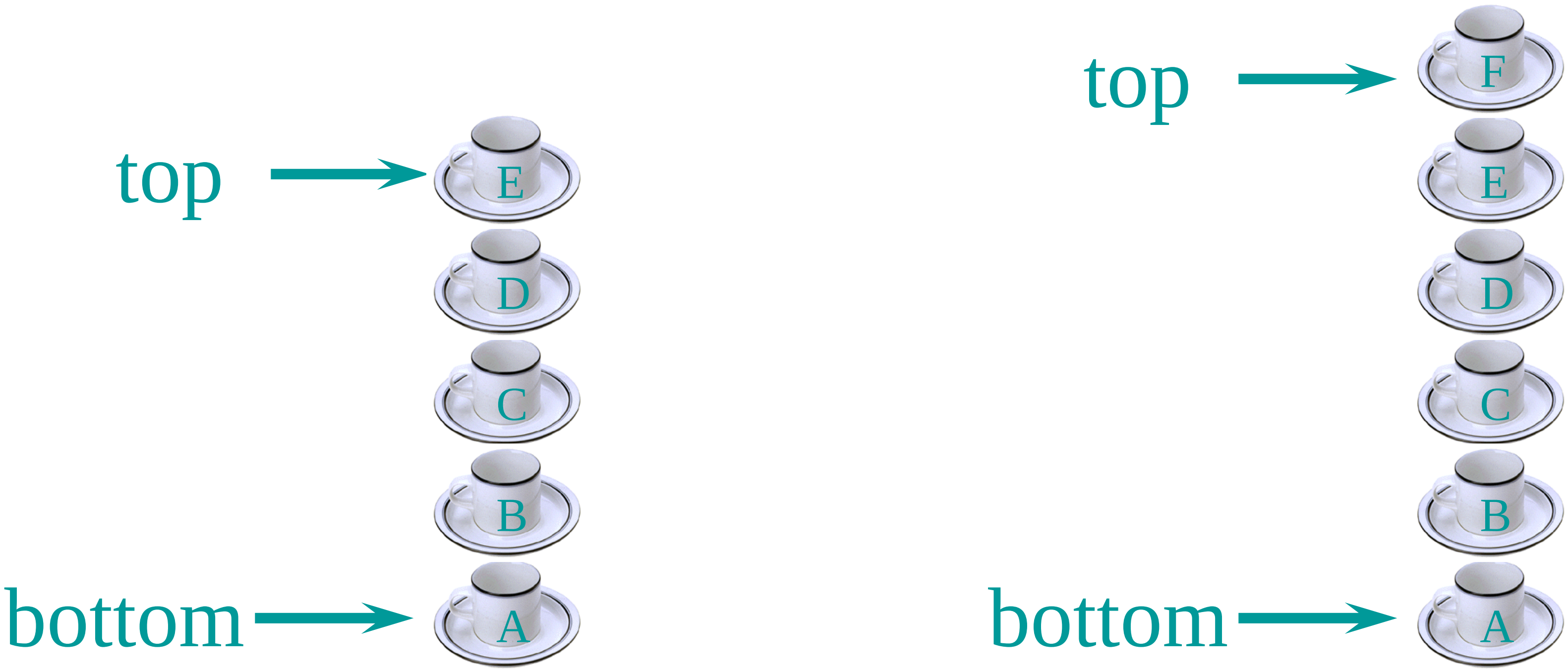
Stacks





- Linear list.
- One end is called top.
- Other end is called bottom.
- Additions to and removals from the top end only.

Stack Of Cups



- Add a cup to the stack.
- Remove a cup from new stack.
- A stack is a LIFO list.

Parentheses Matching

- (((a+b)*c+d-e)/(f+g)-(h+j)*(k-l))/(m-n)
  - Output pairs (u,v) such that the left parenthesis at position u is matched with the right parenthesis at v.
    - (2,6) (1,13) (15,19) (21,25) (27,31) (0,32) (34,38)
- (a+b))*((c+d)
  - (0,4)
  - right parenthesis at 5 has no matching left parenthesis
  - (8,12)

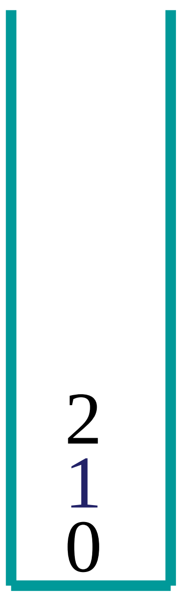
left parenthesis at 7 has no matching right parenthesis

Parentheses Matching

- scan expression from left to right
- when a left parenthesis is encountered, add its position to the stack
- when a right parenthesis is encountered, remove matching position from stack

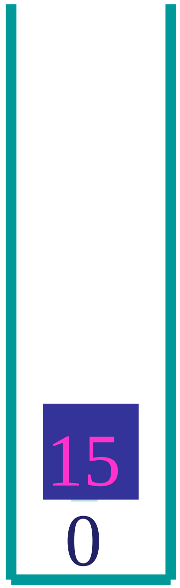
Example

- (((a+b)*c+d-e)/(f+g)-(h+j)*(k-l))/(m-n)



Example

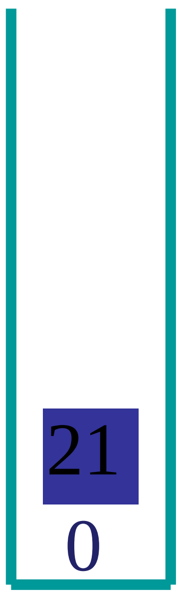
- (((a+b)*c+d-e)/(f+g)-(h+j)*(k-l))/(m-n)



(2,6) (1,13)

Example

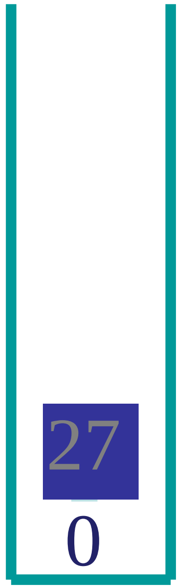
- (((a+b)*c+d-e)/(f+g)-(h+j)*(k-l))/(m-n)



(2,6) (1,13) (15,19)

Example

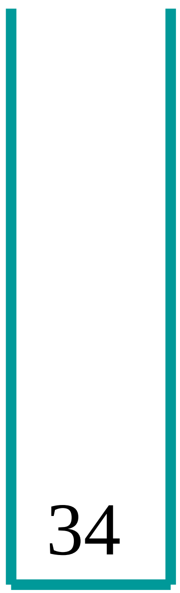
- (((a+b)*c+d-e)/(f+g)-(h+j)*(k-l))/(m-n)



(2,6) (1,13) (15,19) (21,25)

Example

- (((a+b)*c+d-e)/(f+g)-(h+j)*(k-l))/(m-n)



(2,6) (1,13) (15,19) (21,25)(27,31) (0,32)

- and so on

Method Invocation And Return

public void a()

{ …; b(); …}

public void b()

{ …; c(); …}

public void c()

{ …; d(); …}

public void d()

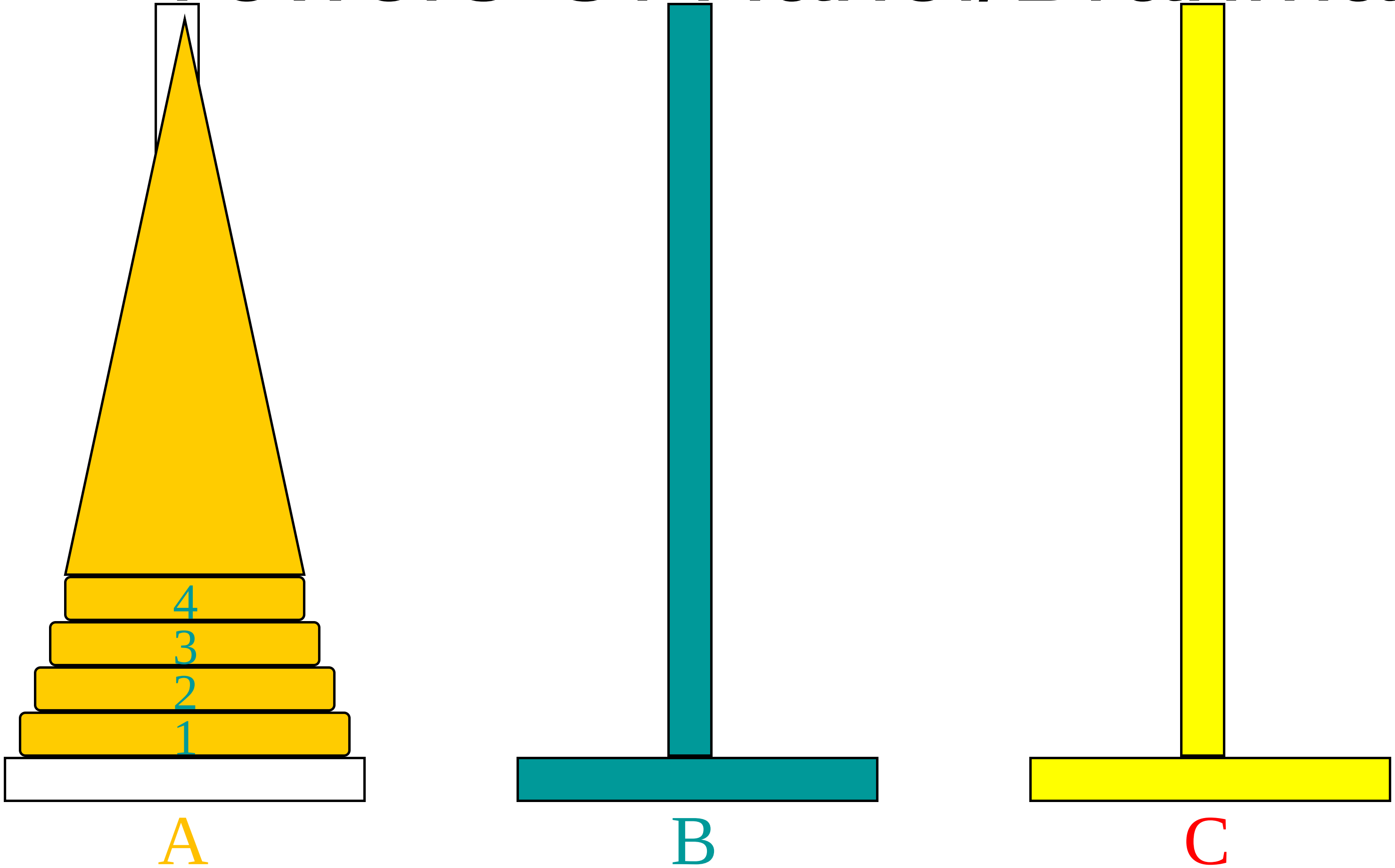
{ …; e(); …}

public void e()

{ …; c(); …}

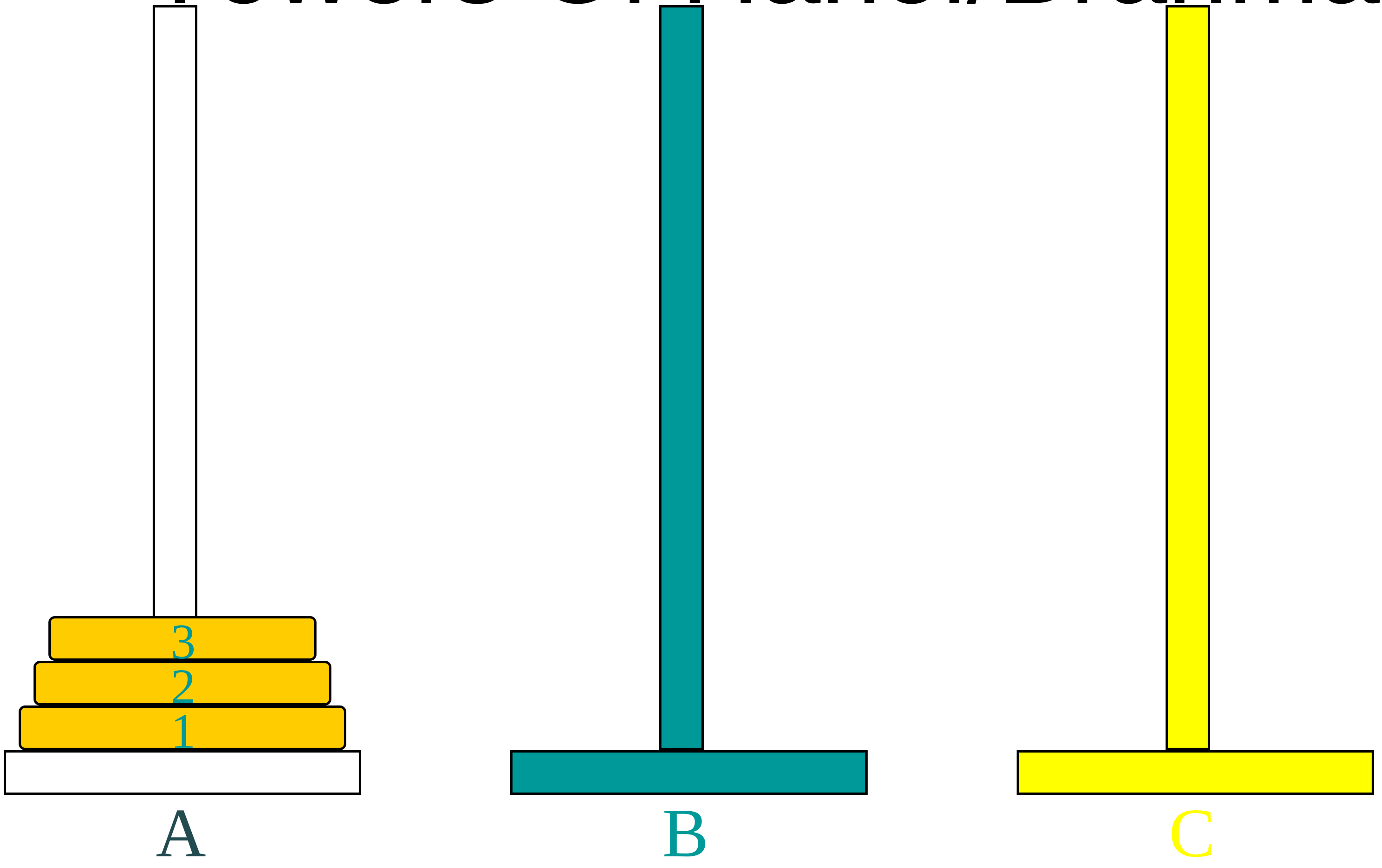
return address in d()
return address in c()
return address in e()
return address in d()
return address in c()
return address in b()
return address in a()

Towers Of Hanoi/Brahma



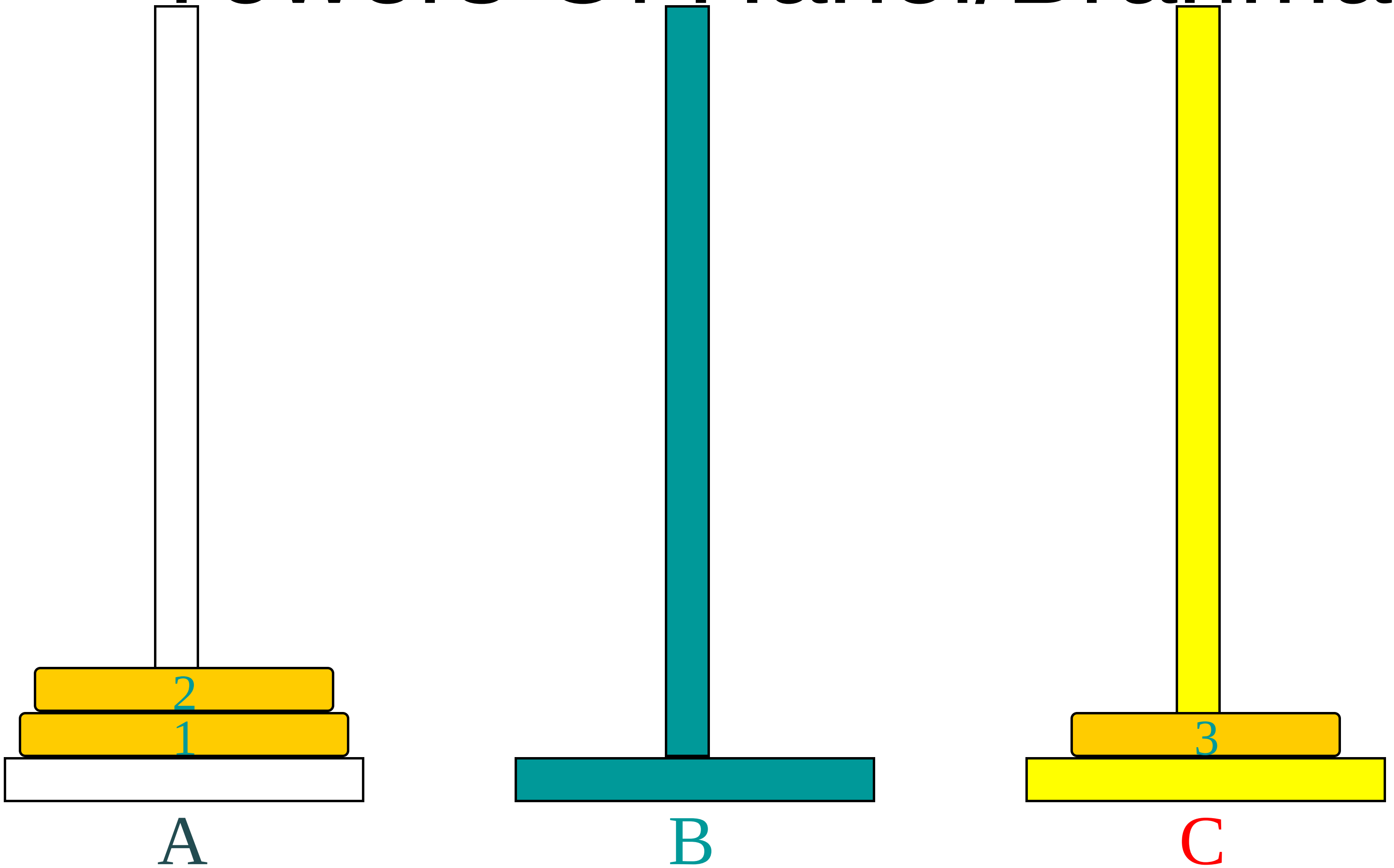
- 64 gold disks to be moved from tower A to tower C
- each tower operates as a stack
- cannot place big disk on top of a smaller one

Towers Of Hanoi/Brahma



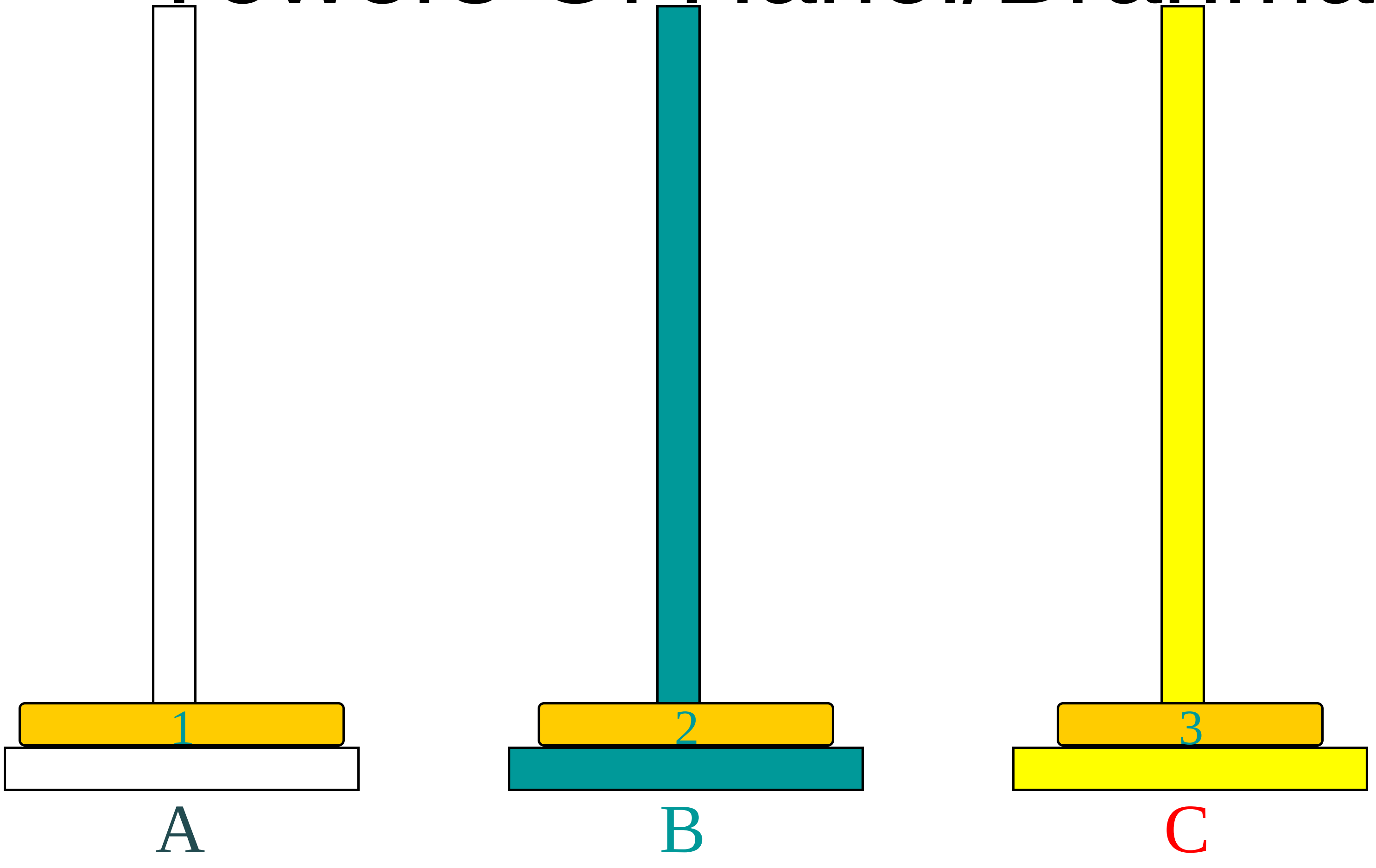
- 3-disk Towers Of Hanoi/Brahma

Towers Of Hanoi/Brahma



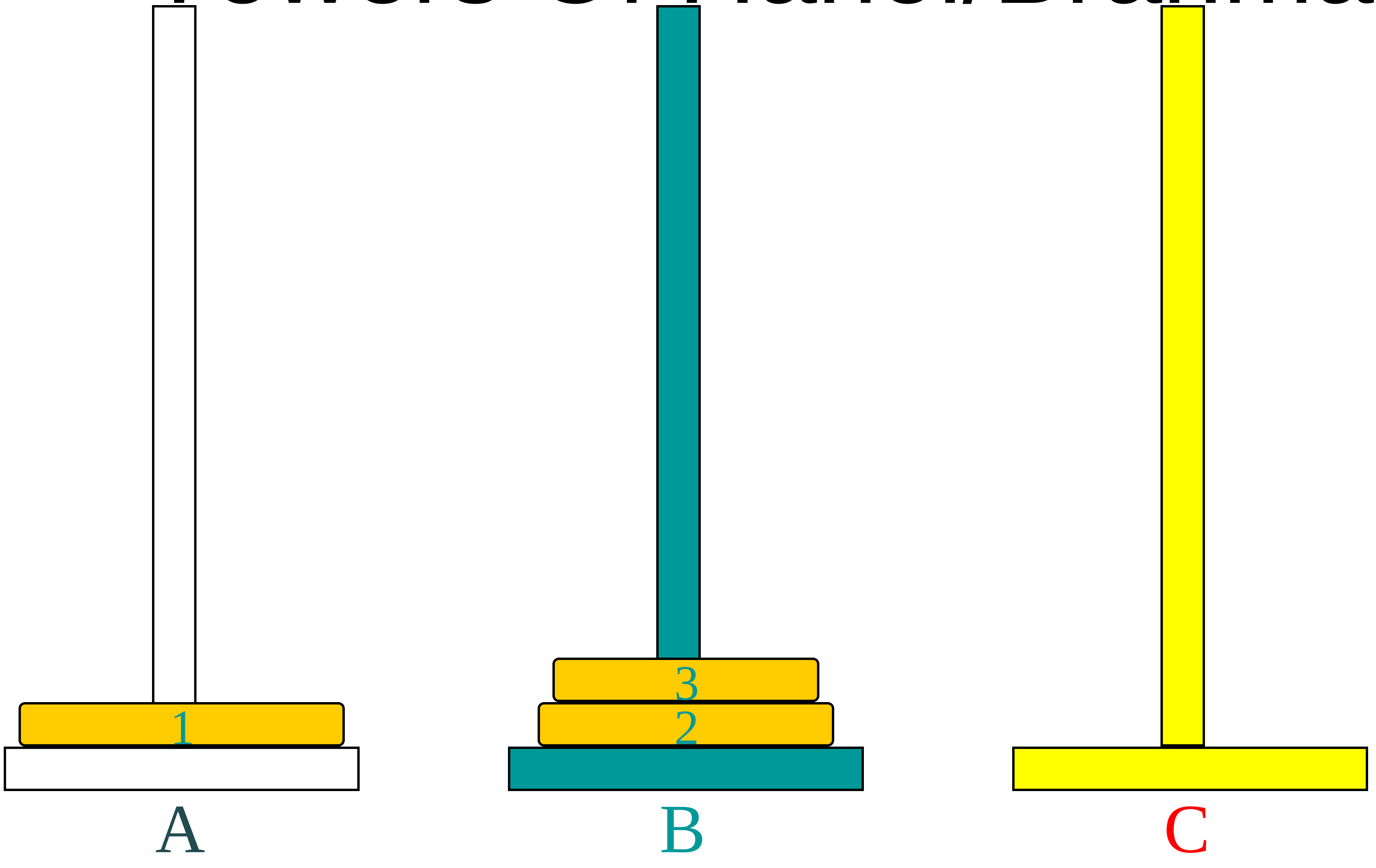
- 3-disk Towers Of Hanoi/Brahma

Towers Of Hanoi/Brahma



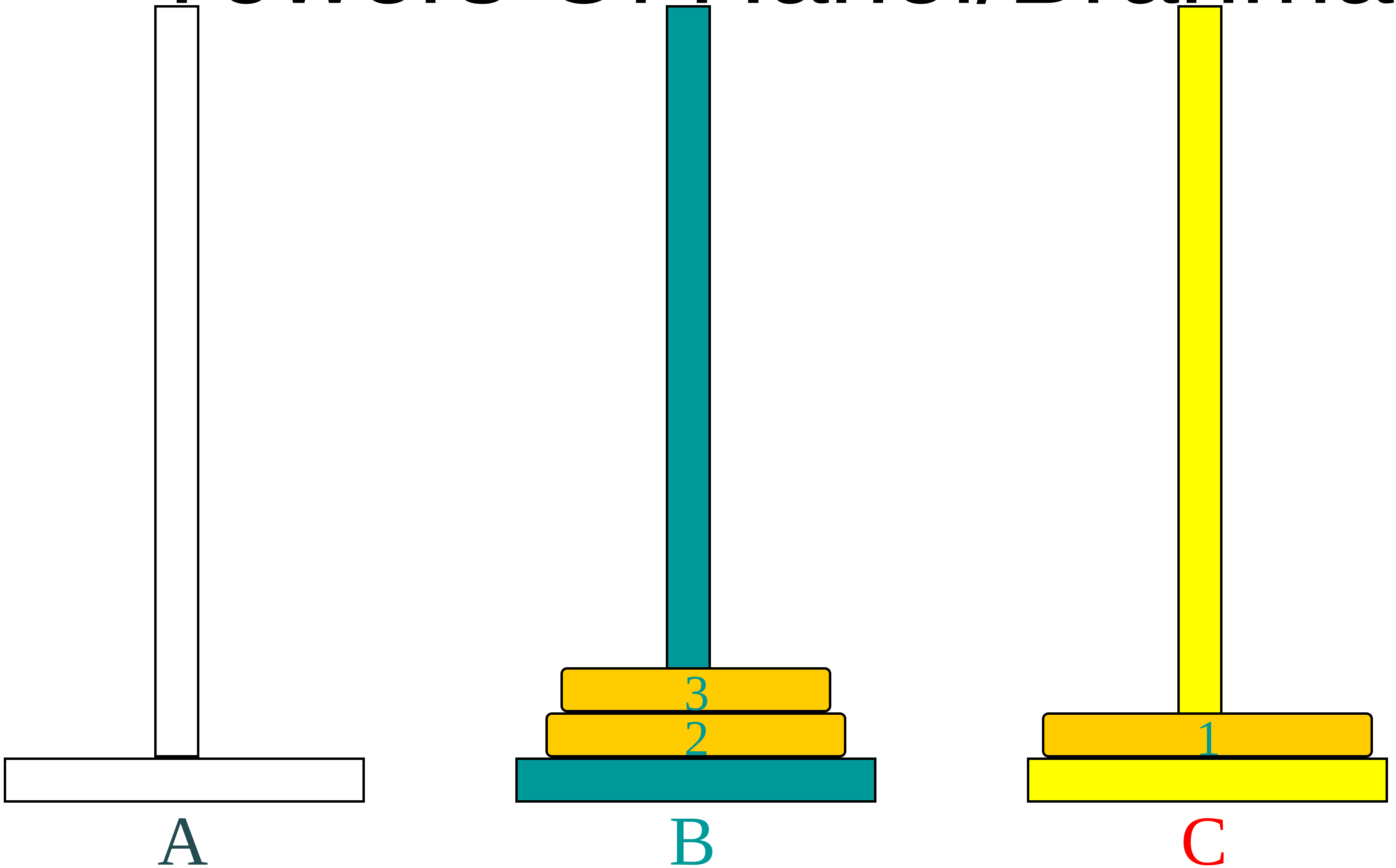
- 3-disk Towers Of Hanoi/Brahma

Towers Of Hanoi/Brahma



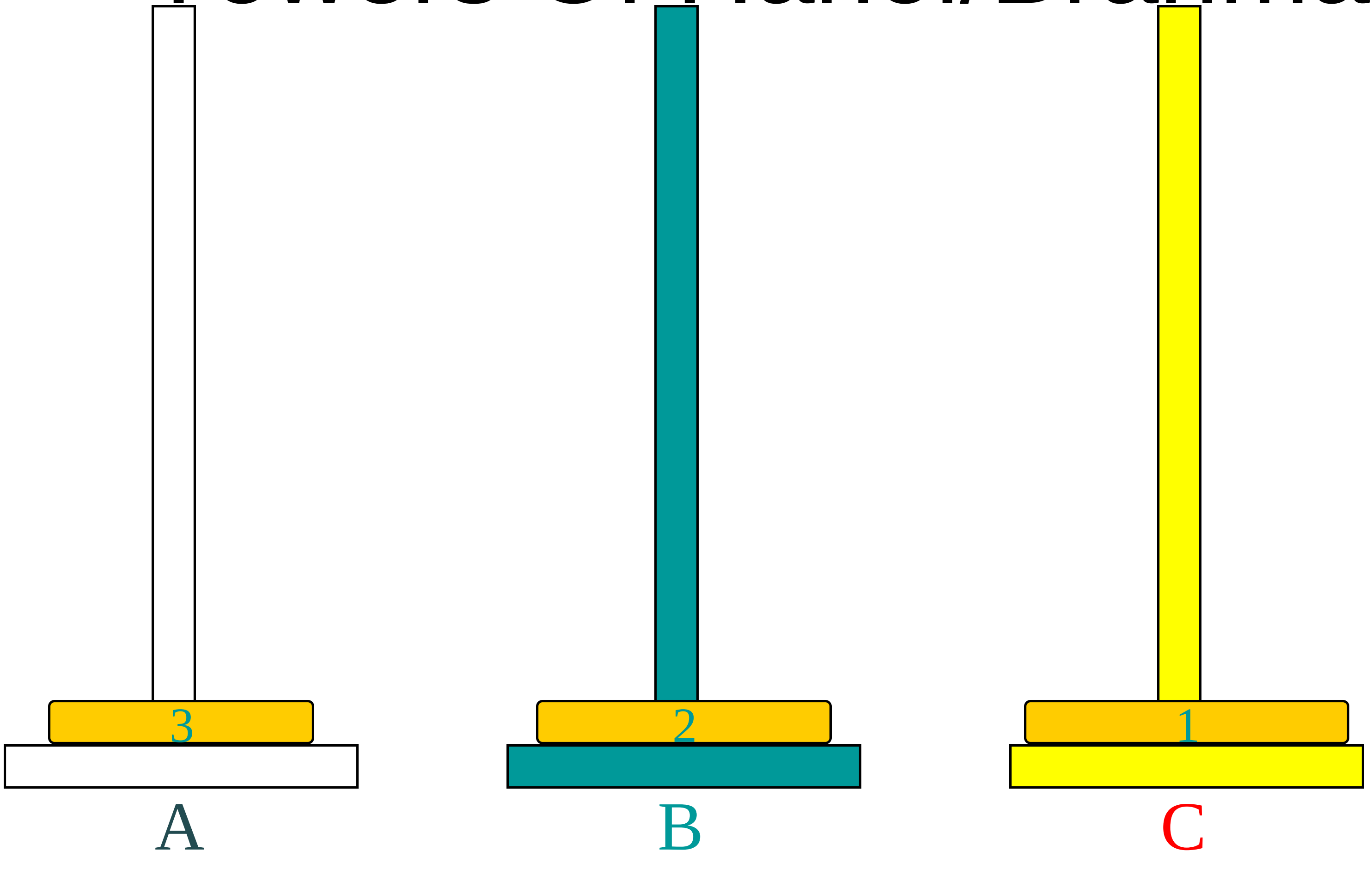
- 3-disk Towers Of Hanoi/Brahma

Towers Of Hanoi/Brahma



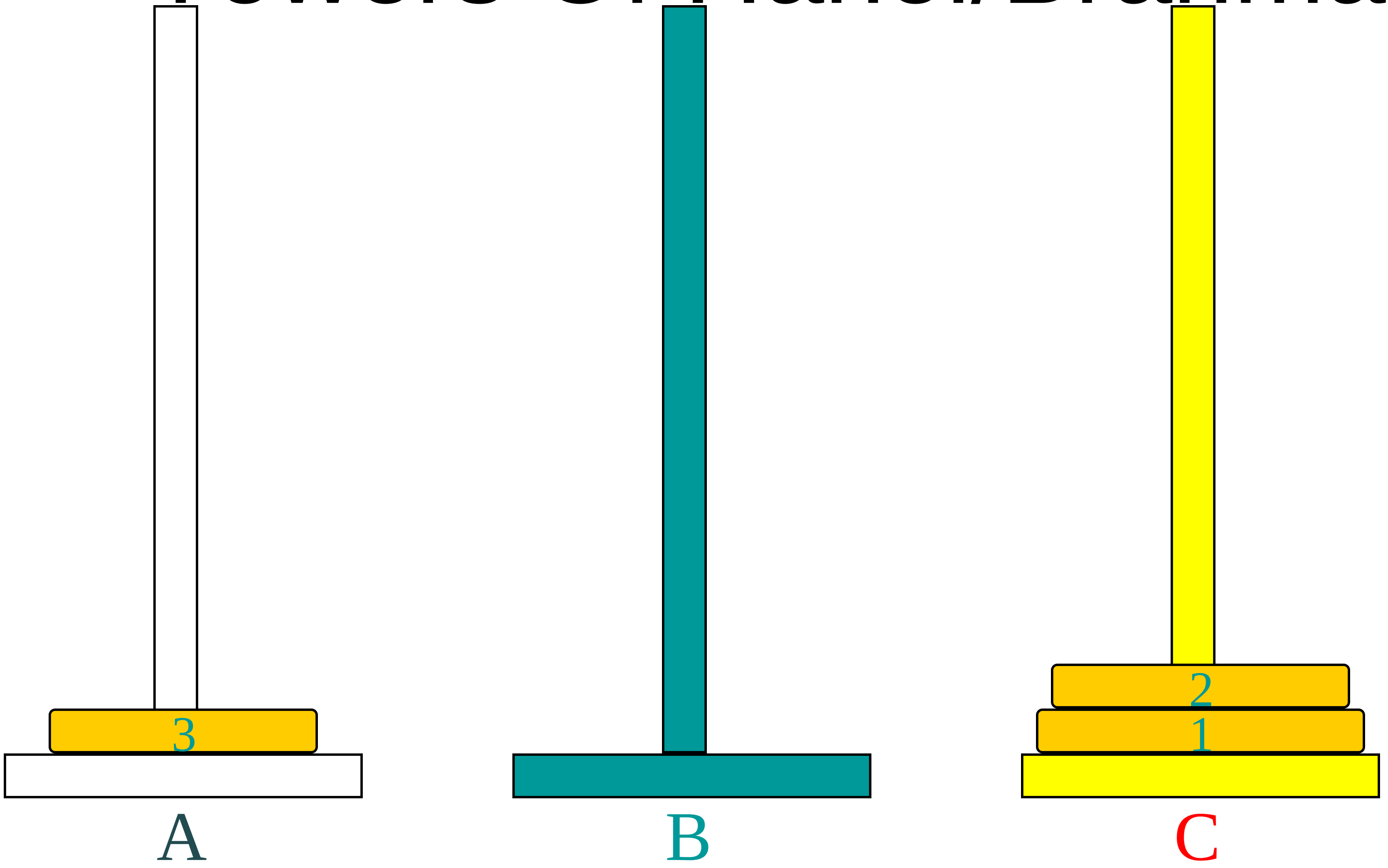
- 3-disk Towers Of Hanoi/Brahma

Towers Of Hanoi/Brahma



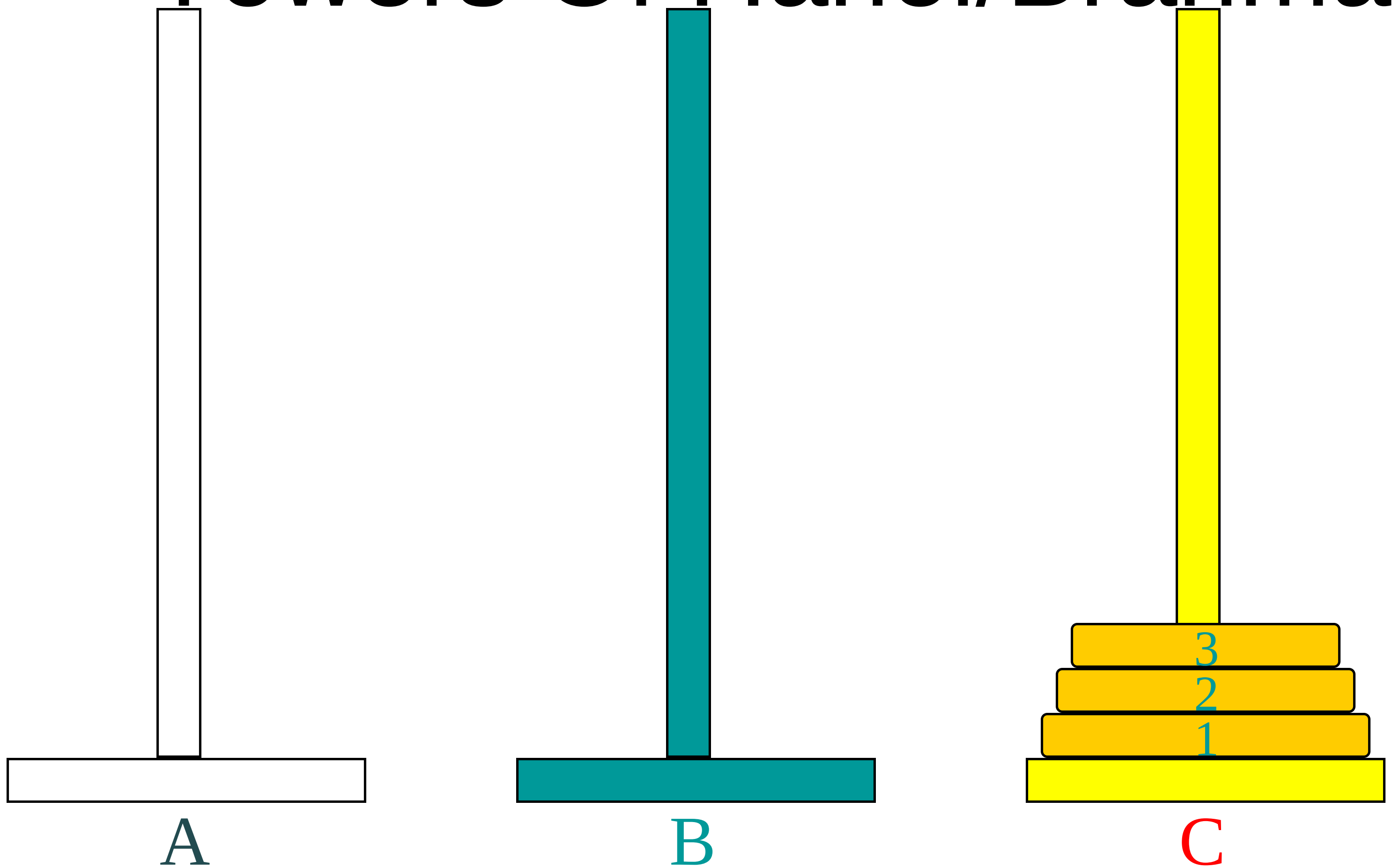
- 3-disk Towers Of Hanoi/Brahma

Towers Of Hanoi/Brahma



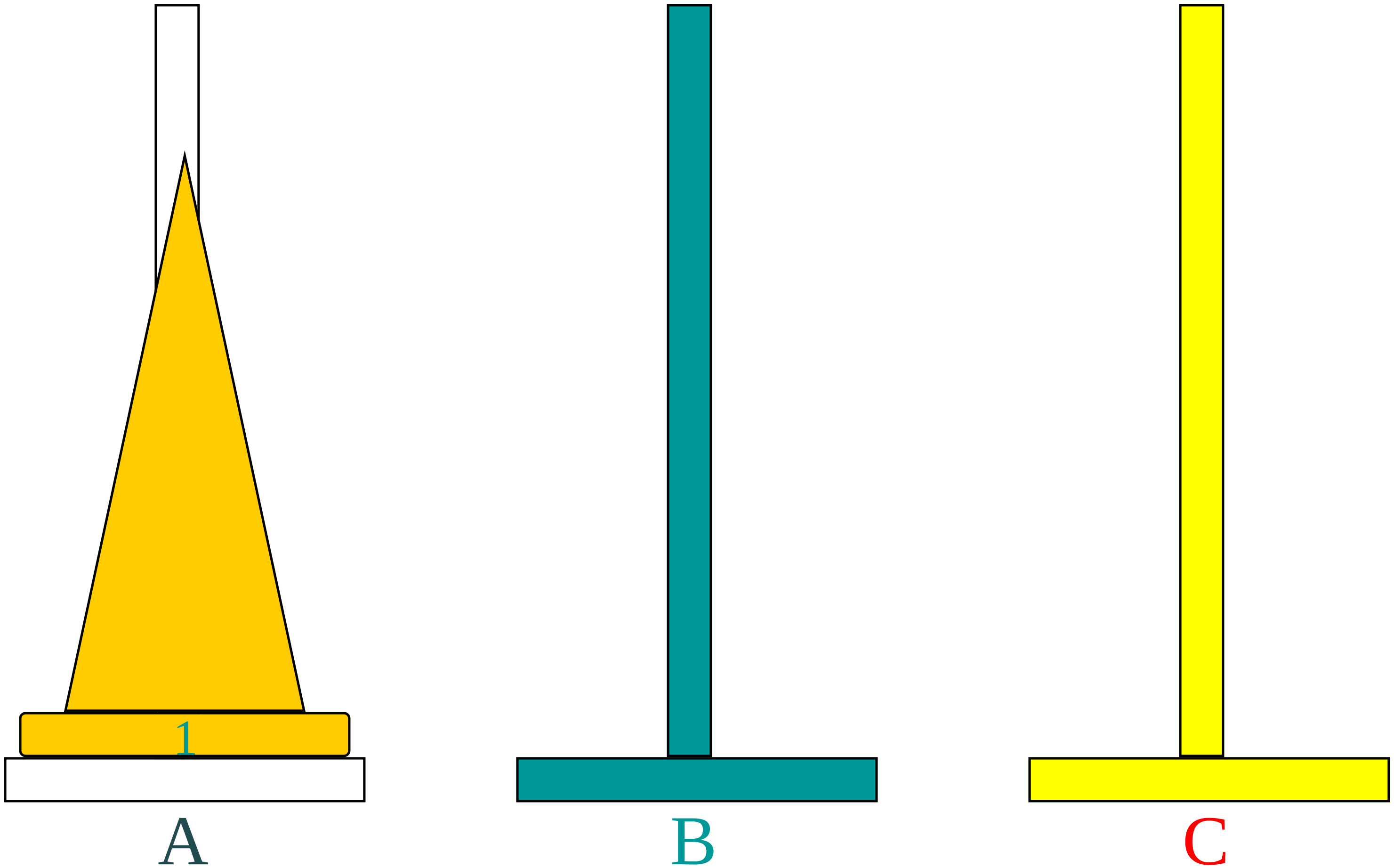
- 3-disk Towers Of Hanoi/Brahma

Towers Of Hanoi/Brahma



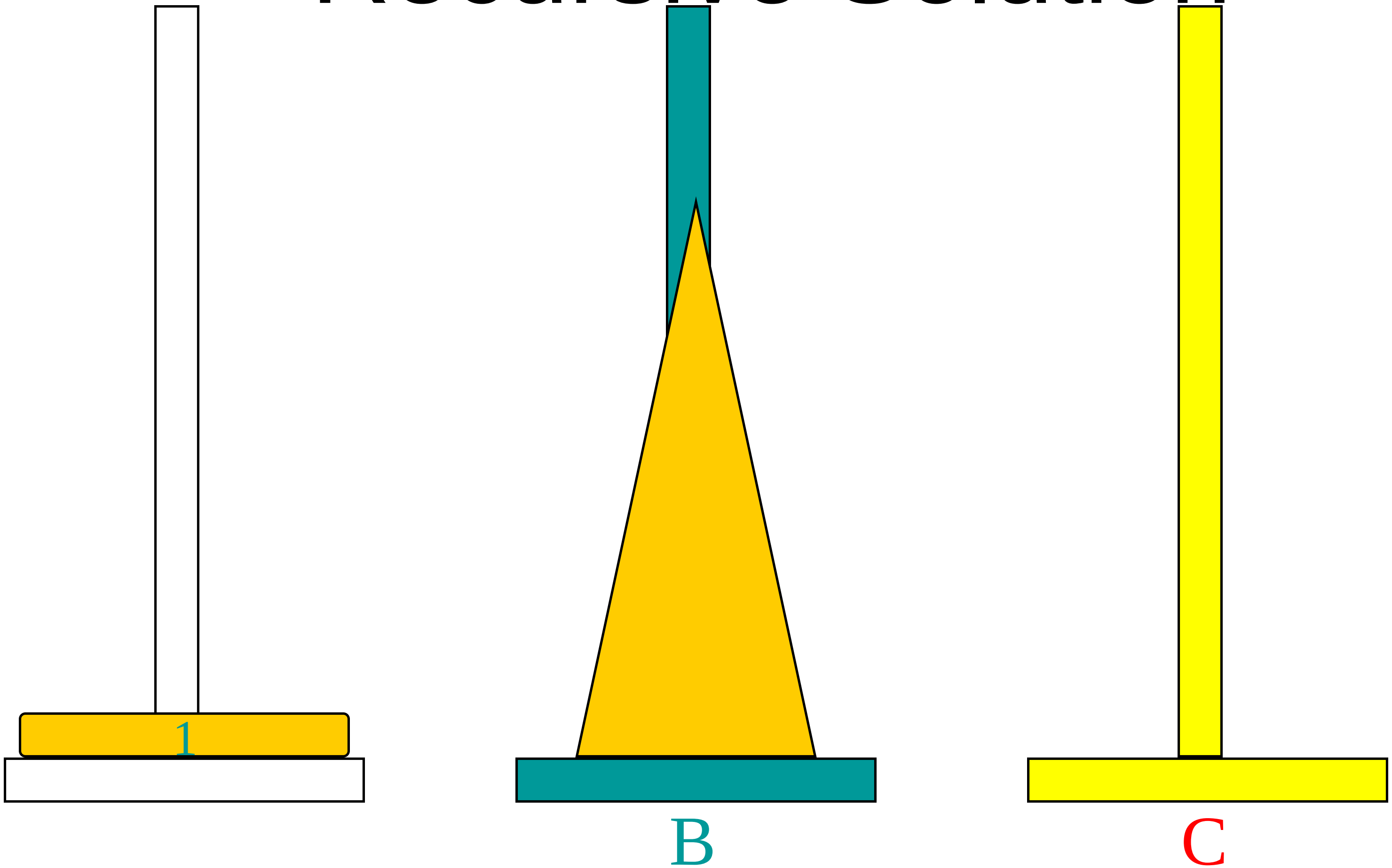
- 3-disk Towers Of Hanoi/Brahma
- 7 disk moves

Recursive Solution



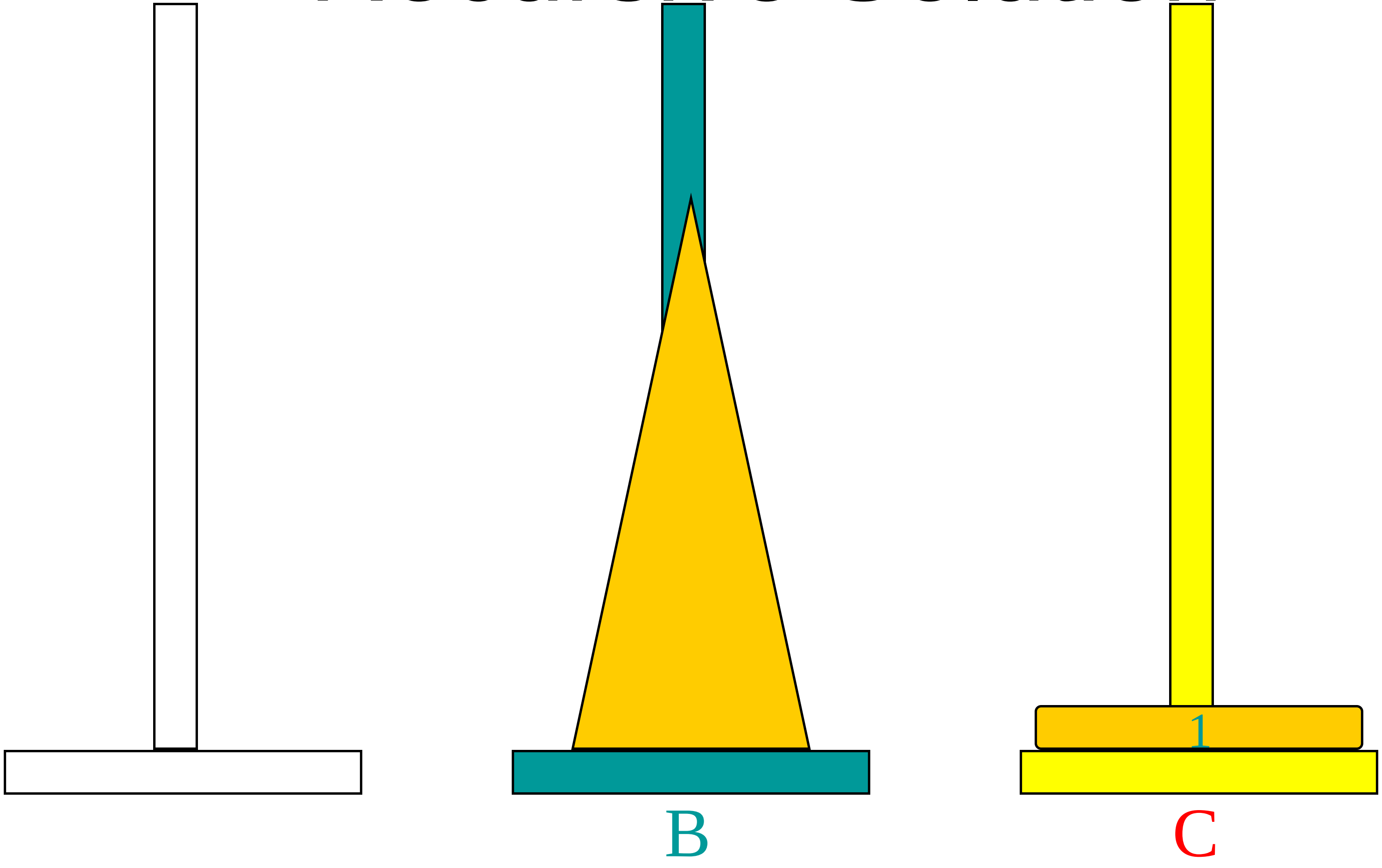
- n > 0 gold disks to be moved from A to C using B
- move top n-1 disks from A to B using C

Recursive Solution



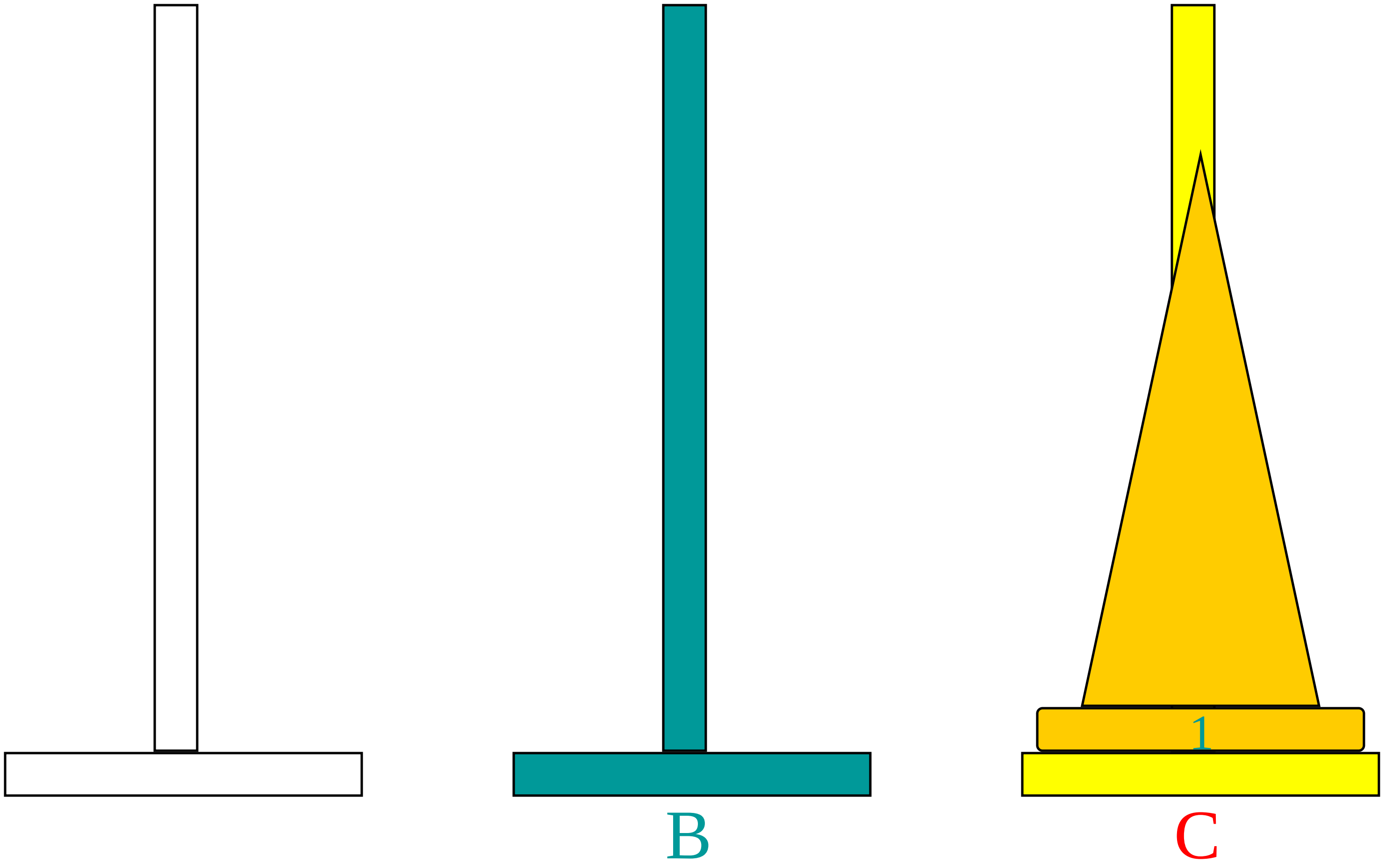
- move top disk from A to C

Recursive Solution



- move top n-1 disks from B to C using A

Recursive Solution



- moves(n) = 0 when n = 0
- moves(n) = 2*moves(n-1) + 1 = 2^n-1 when n > 0

Towers Of Hanoi/Brahma

- moves(64) = $1.8 * 10^{19}$ (approximately)
- Performing 10^9 moves/second, a computer would take about 570 years to complete.
- At 1 disk move/min, the monks will take about $3.4 * 10^{13}$ years.

Algorithm

```
Hanoi(N, Src, Aux, Dst)
  if N is 0
    exit
  else
    Hanoi(N - 1, Src, Dst, Aux)
    Move from Src to Dst
    Hanoi(N - 1, Aux, Src, Dst)
```

- S(3, A,B,C)
  - S(2, A,C,B)
    - S(1, A,B,C)
      - S(0, A,C,B); A->C; S(0, B,A,C)
    - A->B
    - S(1, C,A,B)
      - S(0, C,B,A); C->B; S(0, A,C,B)
  - A-> C
  - S(2, B,A,C)
    - S(1, B,C,A)
      - S(0, B,A,C); B->A; S(0, C,B,A)
    - B->C
    - S(1, A,B,C)
      - S(0, A,C,B); A->C; S(0, B,A,C)

Method Invocation And Return

public void a()

{ ..; b();.. return }

public void b()

{ ..; c();.. return}

public void c()

{ ..; d(); ..return}

public void d()

{ …; return}

return Hanoi(0,A,C,B)

return Hanoi(1,A,B,C)

return Hanoi(2,A,C,B)

return Hanoi(3,A,B,C)

Stacks

- Standard operations:
  - IsEmpty … return true iff stack is empty
  - Top … return top element of stack
  - Push … add an element to the top of the stack
  - Pop … delete the top element of the stack

Stacks

- Use a 1D array to represent a stack.
- Stack elements are stored in stack[0] through stack[top].

The Class Stack

```
class Stack:
    def __init__(self, capacity=10): ...
    def is_empty(self): ...
    def top(self): ...
    def push(self, x): ...
    def pop(self): ...
```

Constructor

```
def __init__(self, capacity=10):
    if capacity < 1:
        raise Exception('Stack capacity must be > 0')

    # position of top element
    self.__capacity = capacity

    # list for stack elements
    self.__stack = []

    # capacity of stack list
    self.__top = -1
```

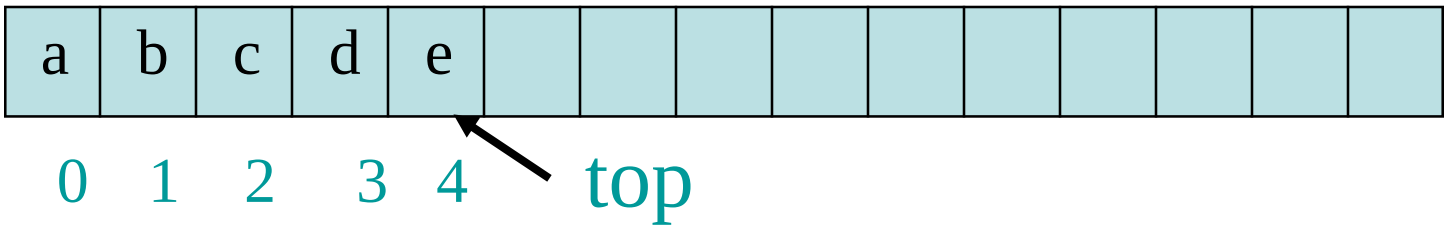
IsEmpty

```
def is_empty(self):
    return self.__top == -1
```

Top

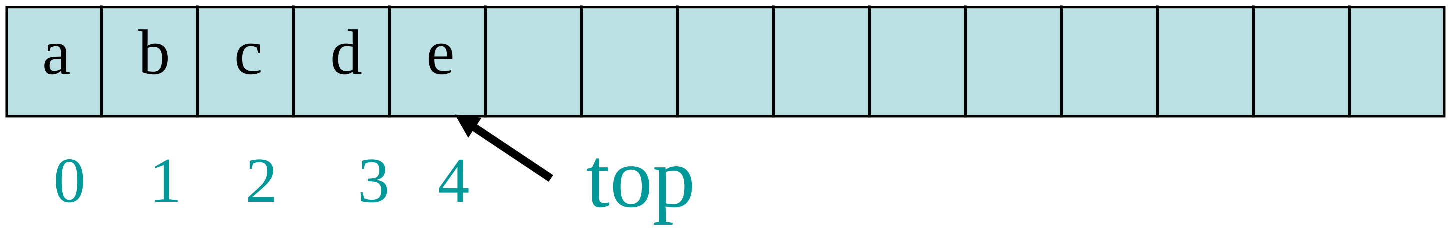
```python
def top(self):
    if self.is_empty():
        raise Exception('Stack is empty')
    return self.__stack[self.__top]
```

Push




```python
# Add x to the stack.
def push(self, x):
    if self.__top == self.__capacity-1:
        self.__capacity *= 2

    # add at stack top
    self.__stack.append(x)
    self.__top += 1
```

Pop



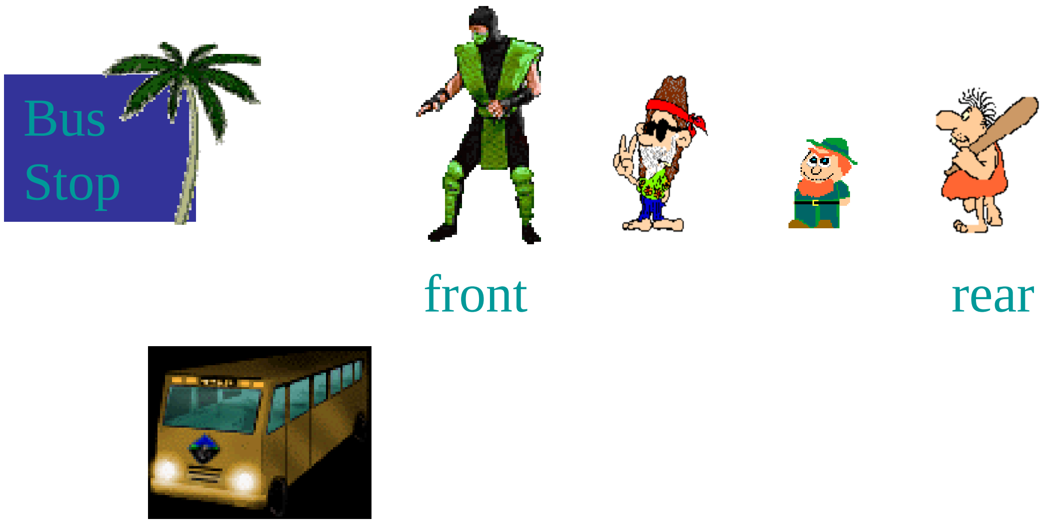

```python
def pop(self):
    if self.is_empty():
        raise Exception('Stack is empty. Cannot delete.')

    self.__top += -1
    return self.__stack.pop()
```

Queues

- Linear list.
- One end is called front.
- Other end is called rear.
- Additions are done at the rear only.
- Removals are made from the front only.

Bus Stop Queue

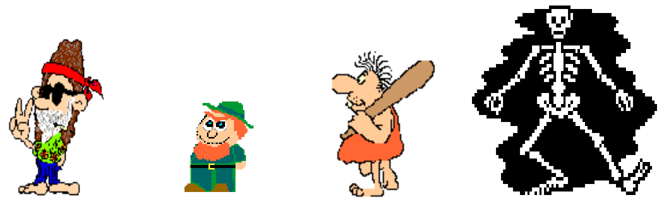
Bus Stop Queue



Bus Stop Queue

front rear

Bus Stop Queue

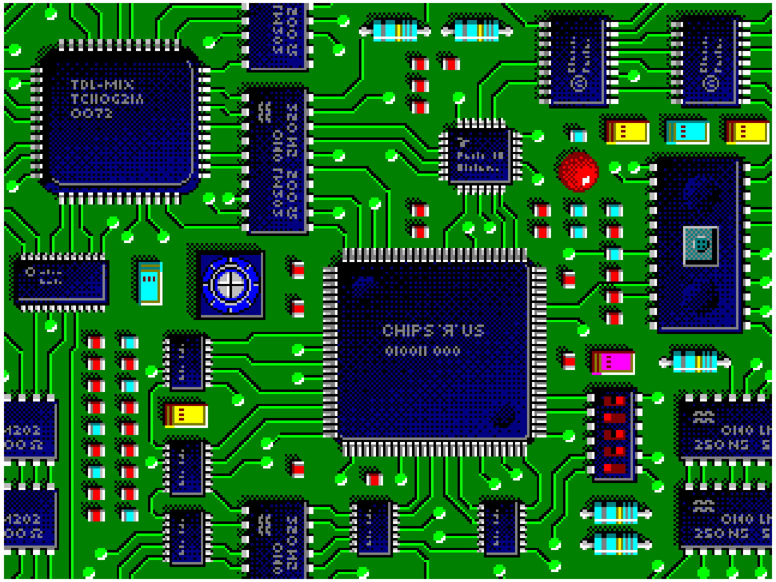


front rear

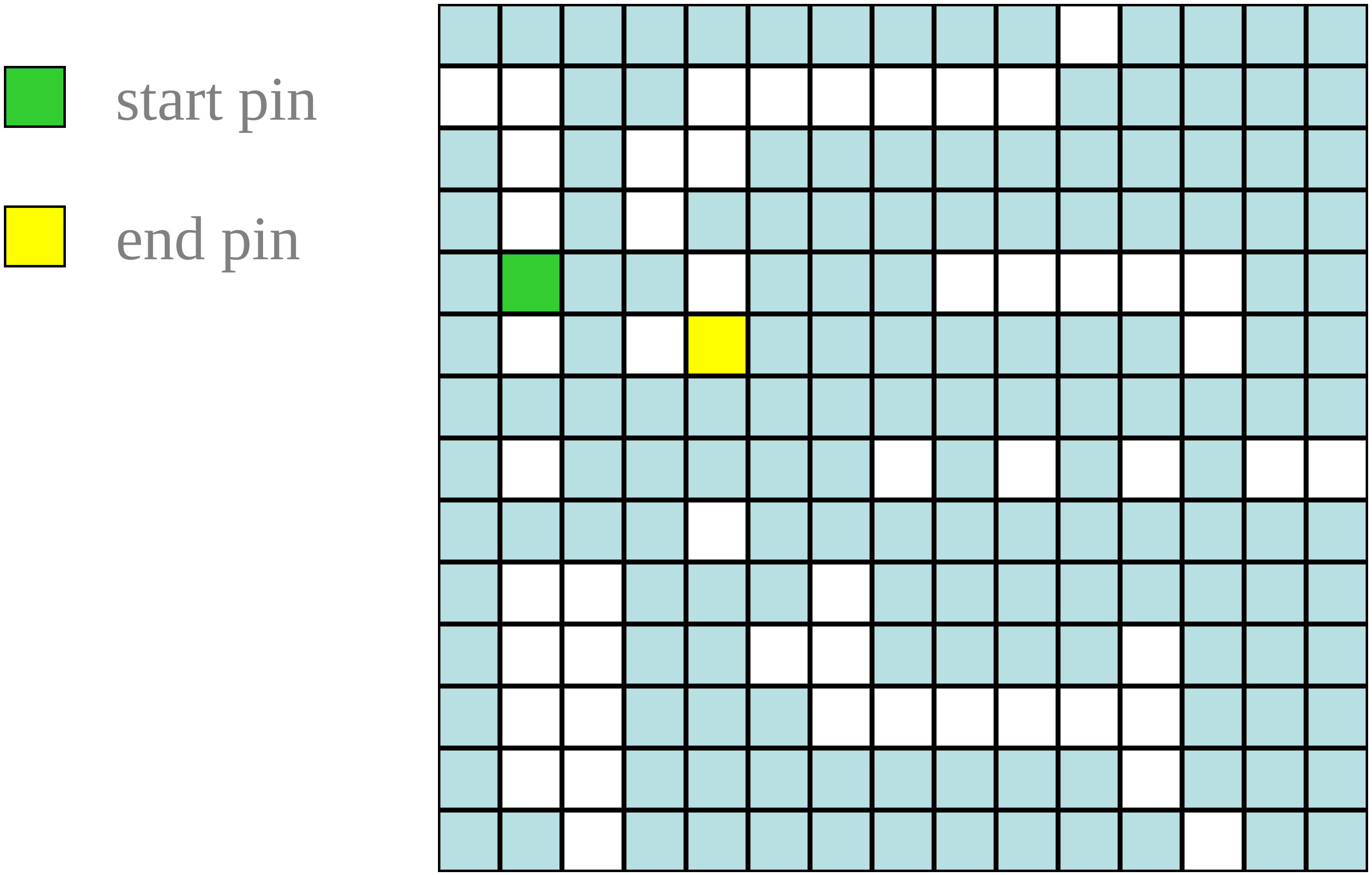
Revisit Of Stack Applications

- Applications in which the stack cannot be replaced with a queue.
  - – Parentheses matching.
  - – Towers of Hanoi.
  - – Method invocation and return.
- Application in which the stack may be replaced with a queue.
  - – Rat in a maze.
    - Results in finding shortest path to exit.

Wire Routing

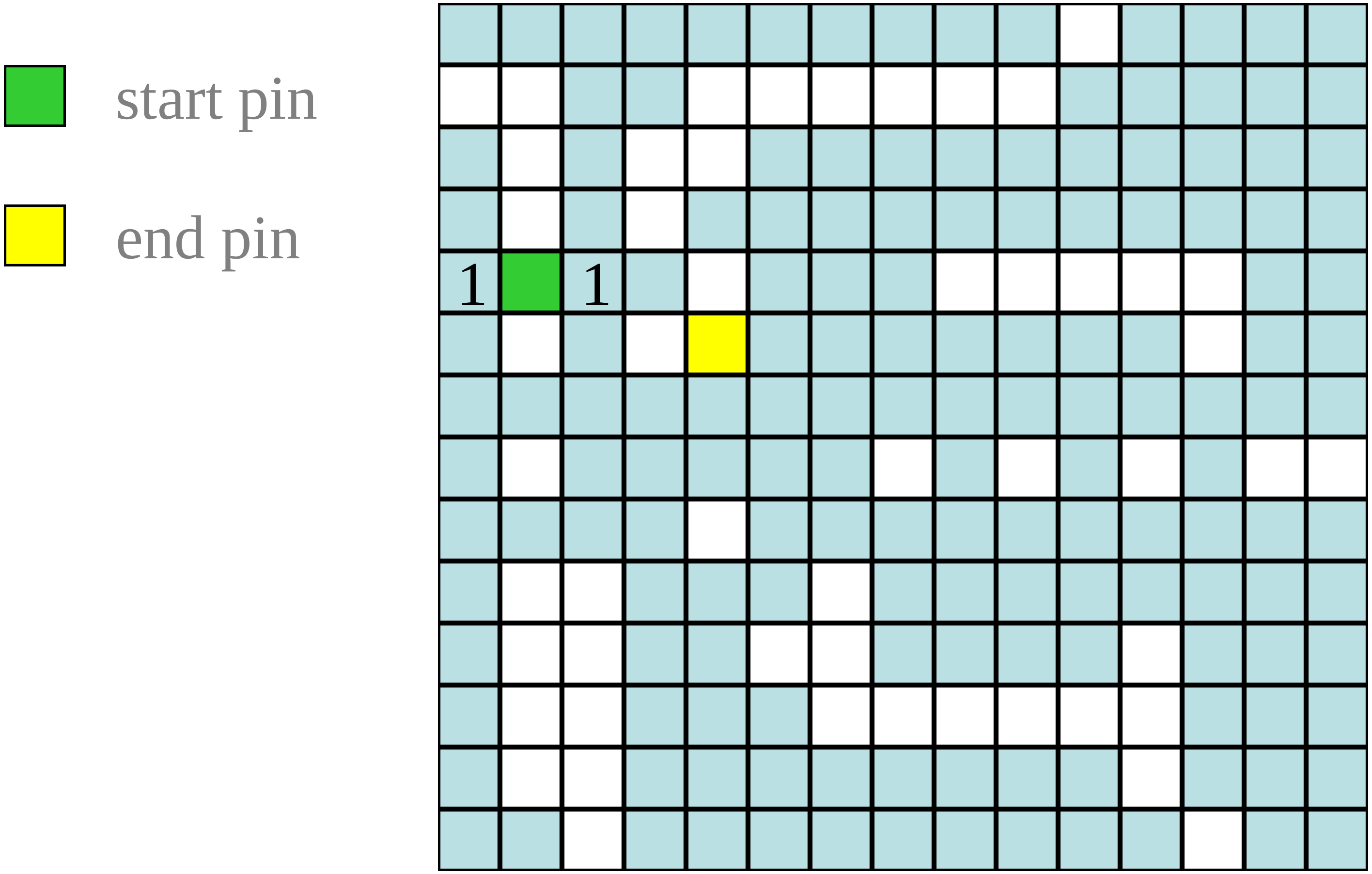


Lee's Wire Router



Label all reachable squares 1 unit from start.

Lee’s Wire Router

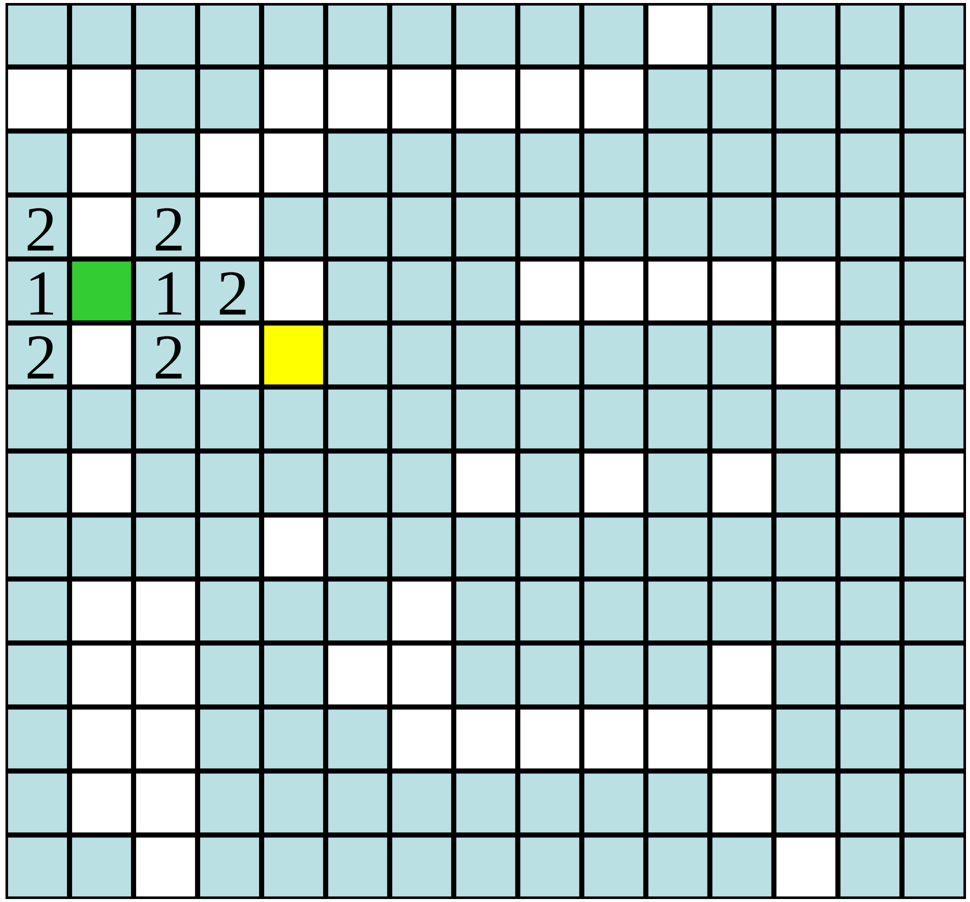


Label all reachable unlabeled squares 2 units from start.

Lee’s Wire Router

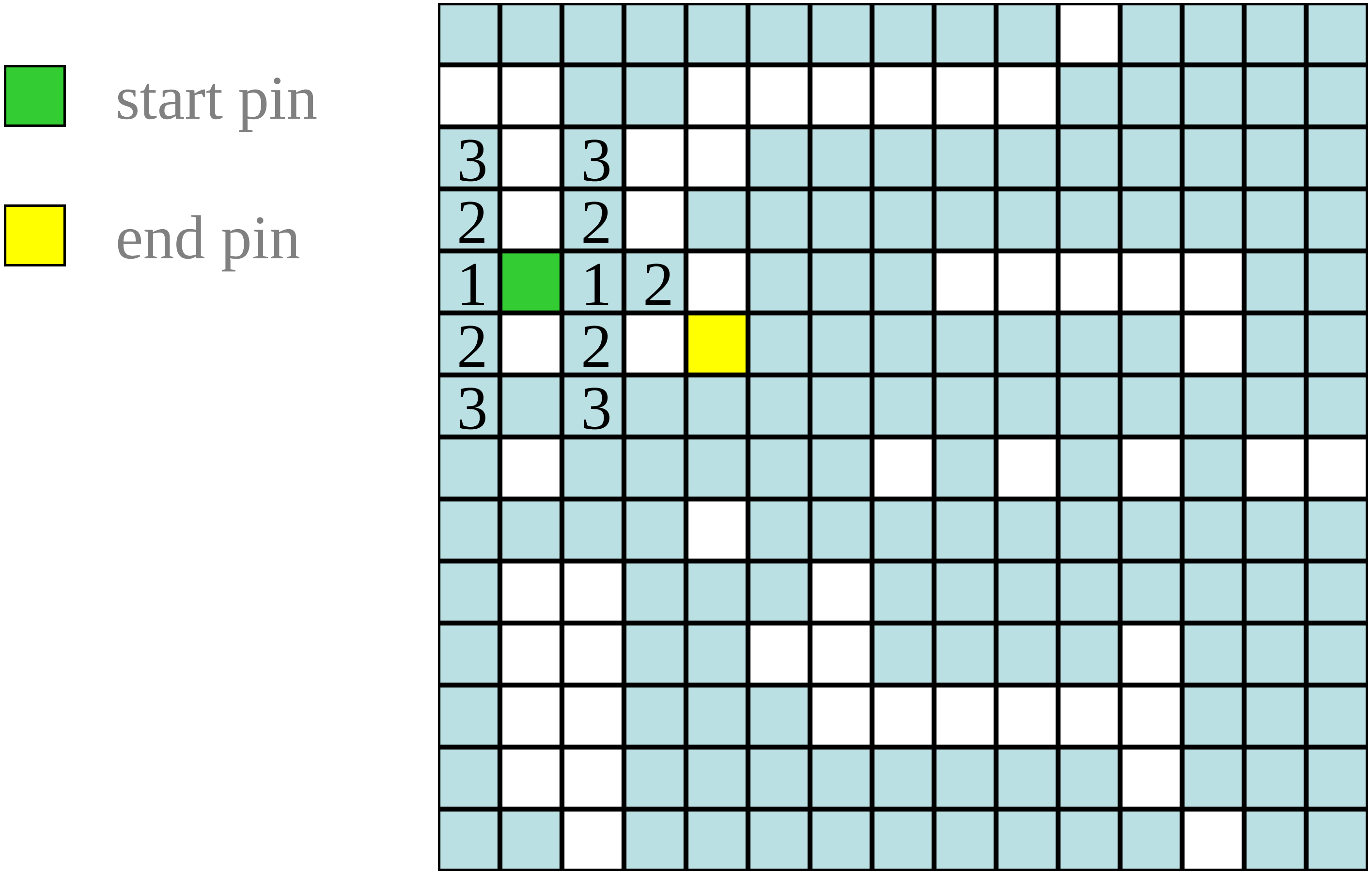
start pin

end pin



Label all reachable unlabeled squares 3 units from start.

Lee’s Wire Router

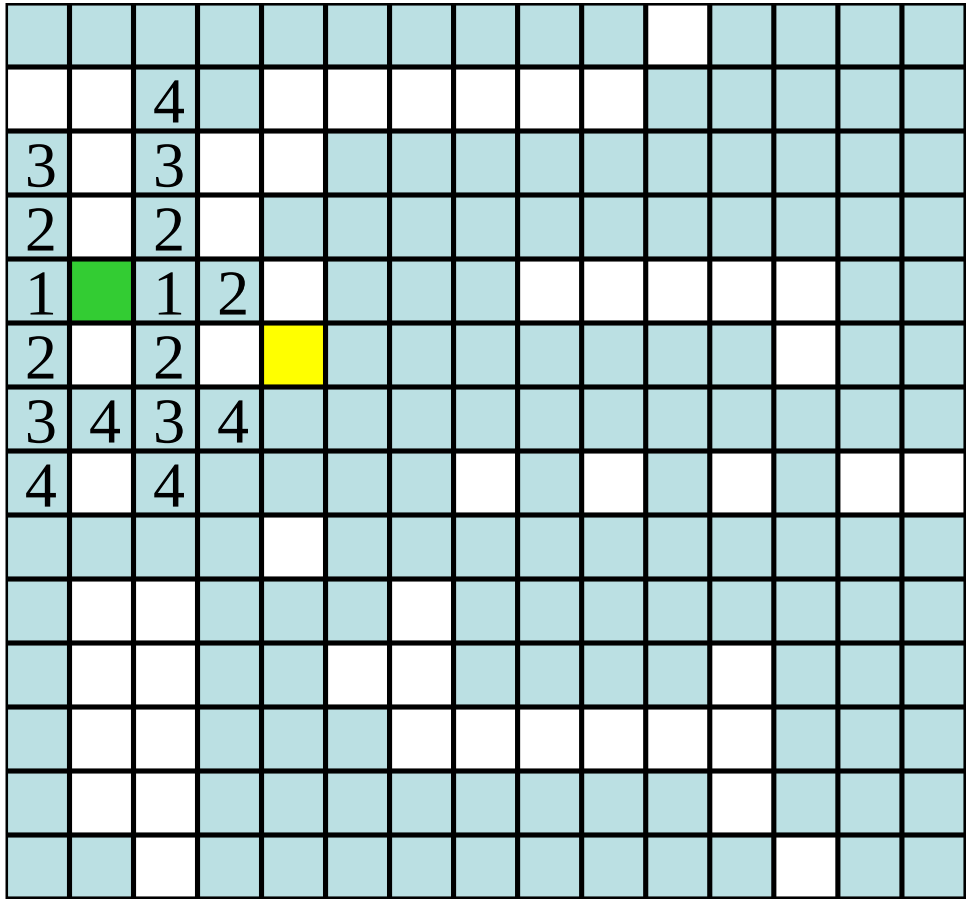


Label all reachable unlabeled squares 4 units from start.

Lee's Wire Router

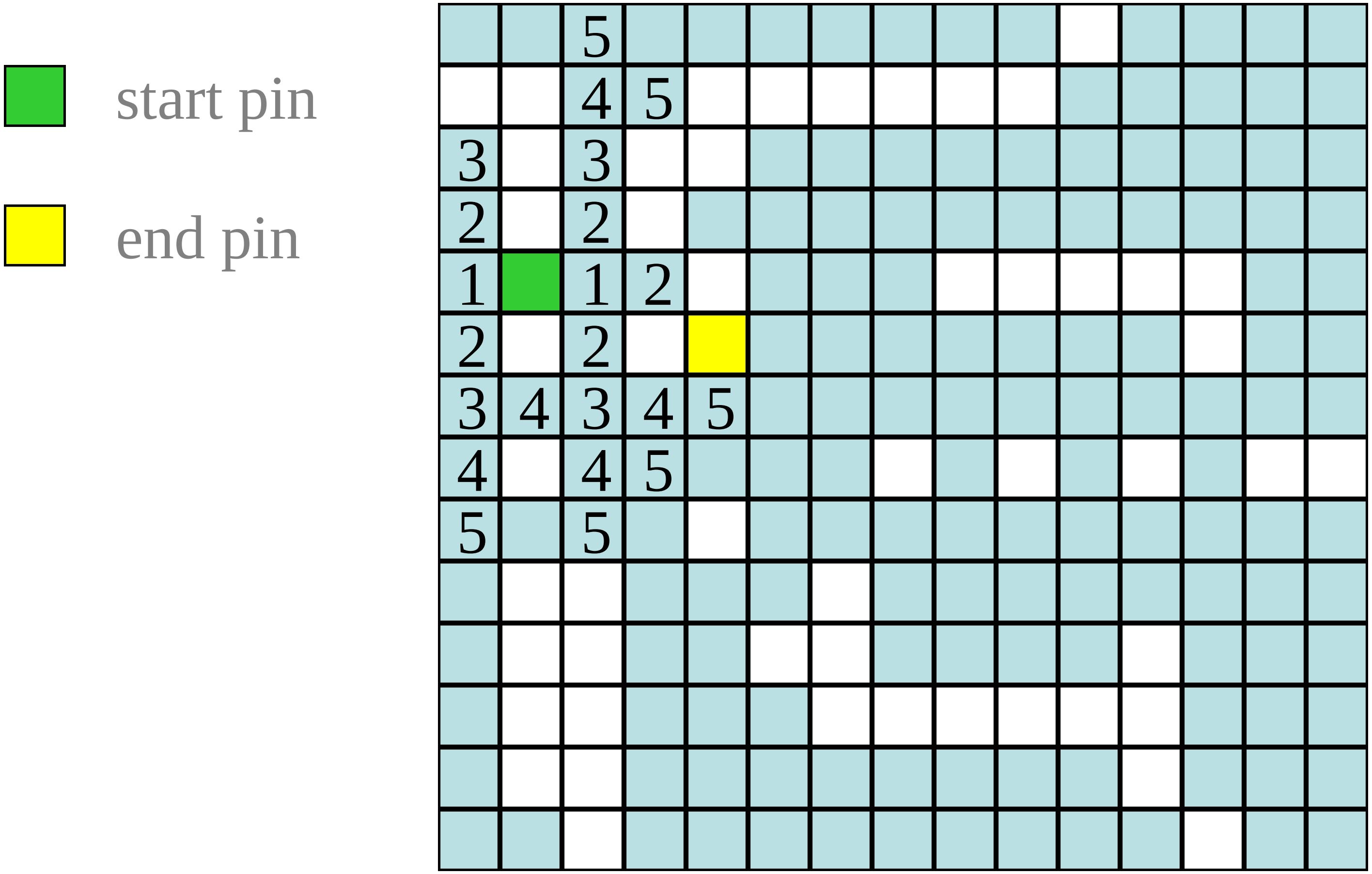
start pin

end pin



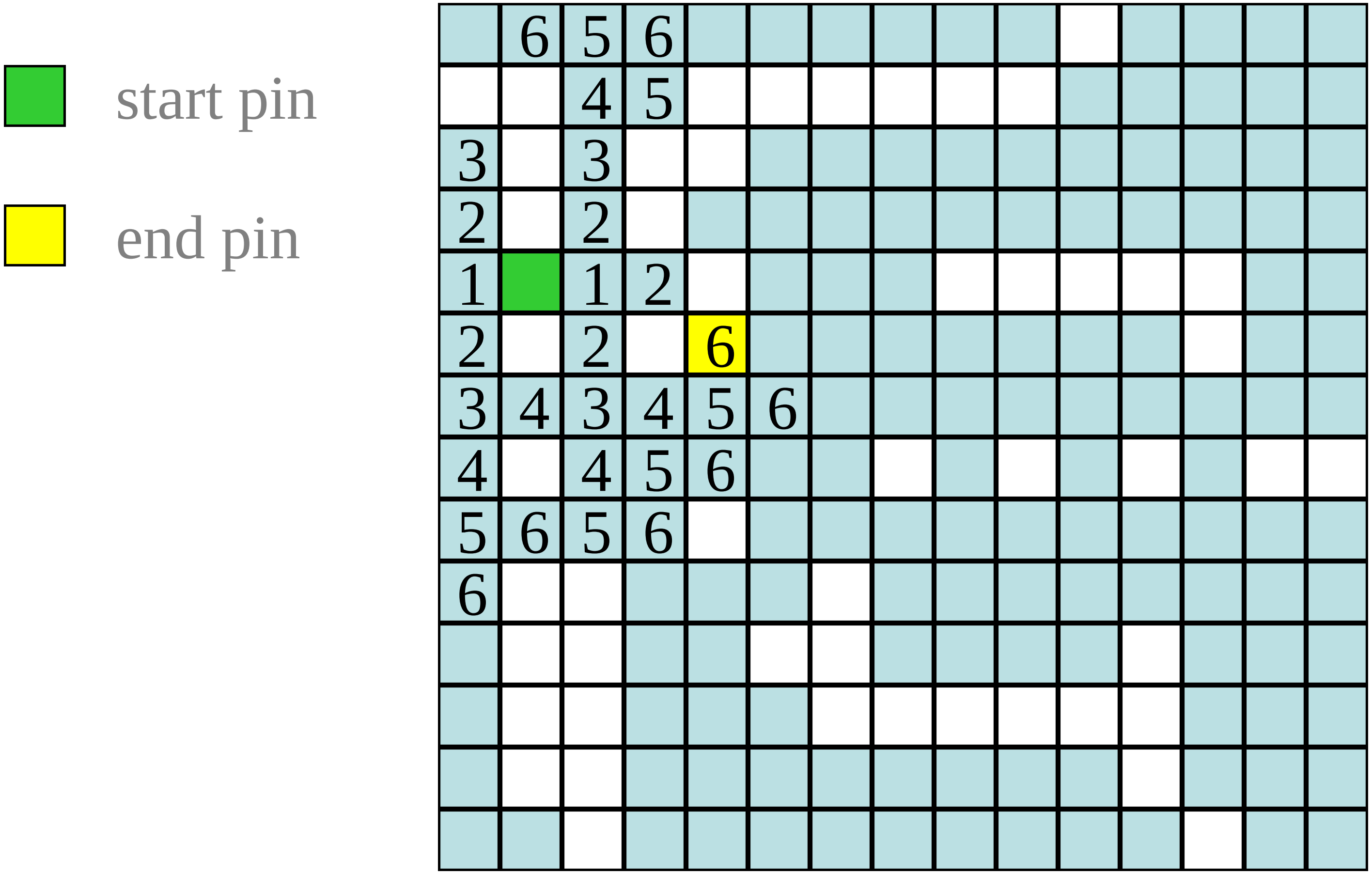
Label all reachable unlabeled squares 5 units from start.

Lee’s Wire Router



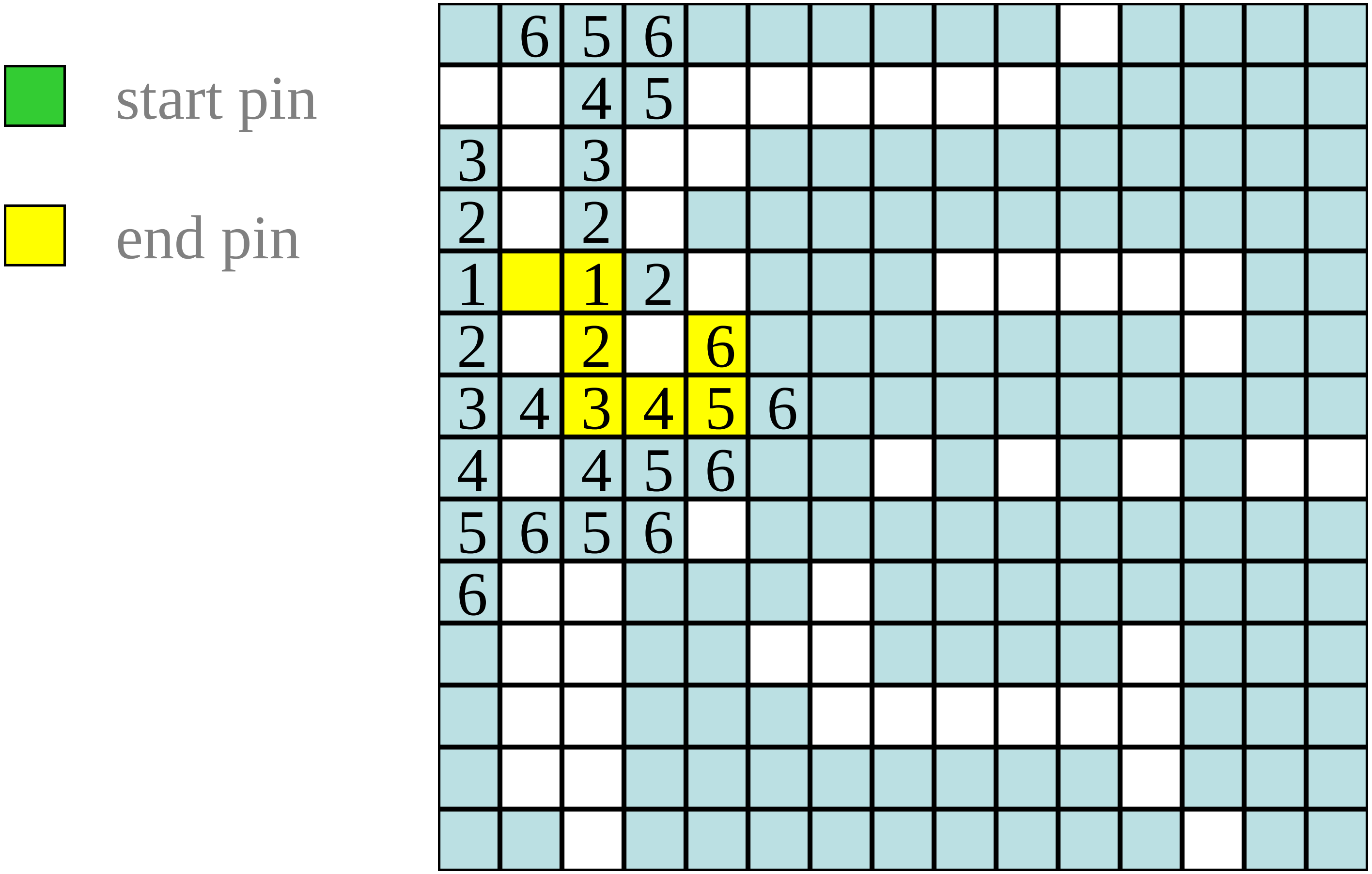
Label all reachable unlabeled squares 6 units from start.

Lee’s Wire Router



End pin reached. Traceback.

Lee’s Wire Router



End pin reached. Traceback.

Queue Operations

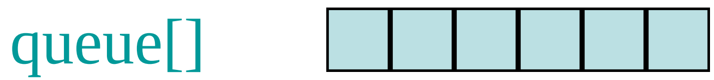
- – IsEmpty … return true iff queue is empty
- – Front … return front element of queue
- – Rear … return rear element of queue
- – Push … add an element at the rear of the queue
- – Pop … delete the front element of the queue

Queue in an Array

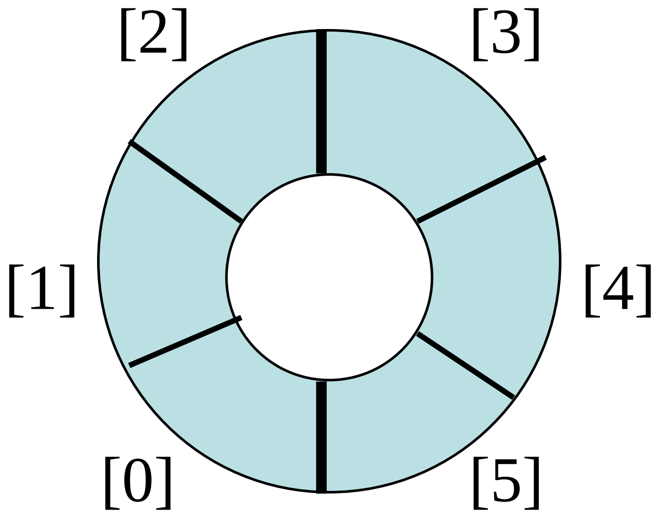
- Use a 1D array to represent a queue.
- Suppose queue elements are stored with the front element in queue[0], the next in queue[1], and so on.

Custom Array Queue

- Use a 1D array queue.

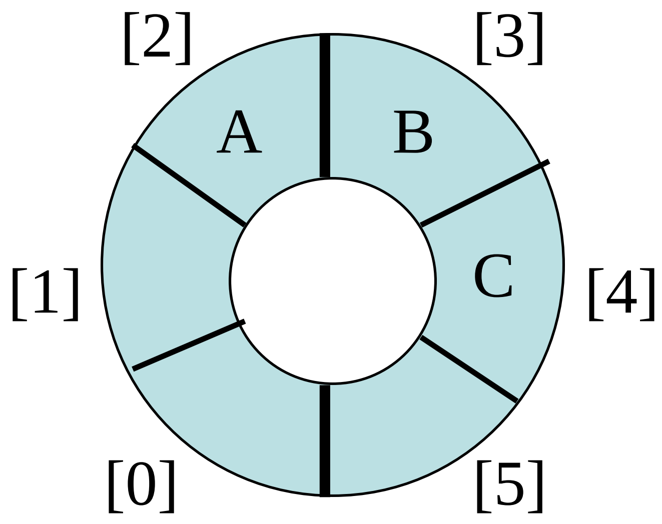


- Circular view of array.



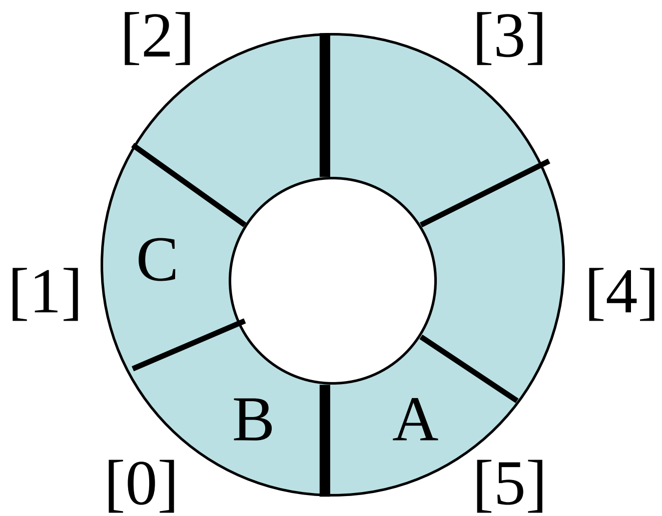
Custom Array Queue

- Possible configuration with 3 elements.



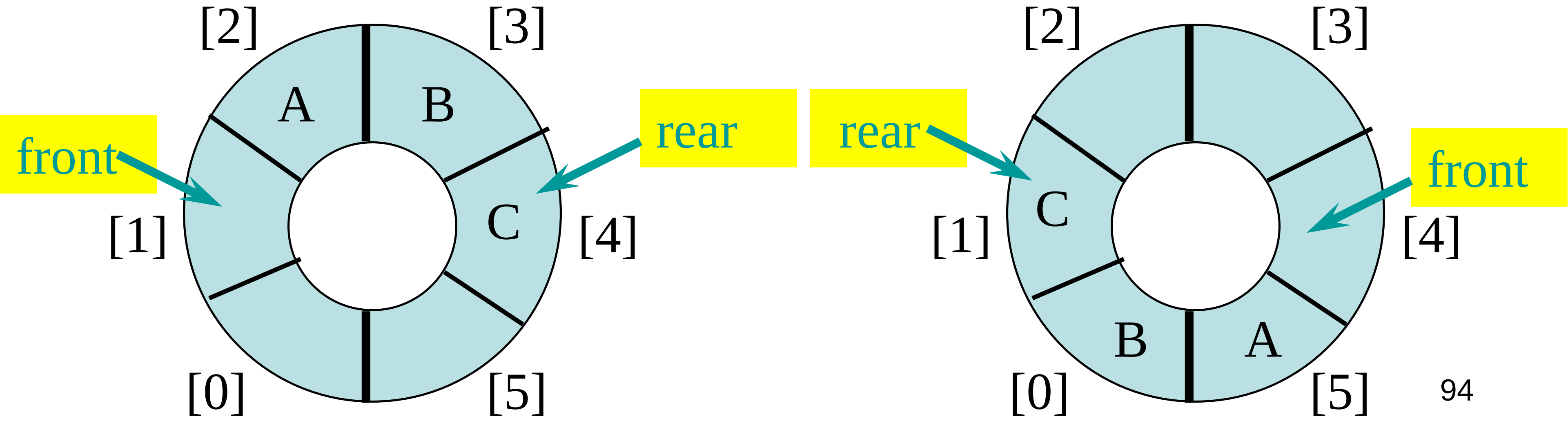
Custom Array Queue

- Another possible configuration with 3 elements.



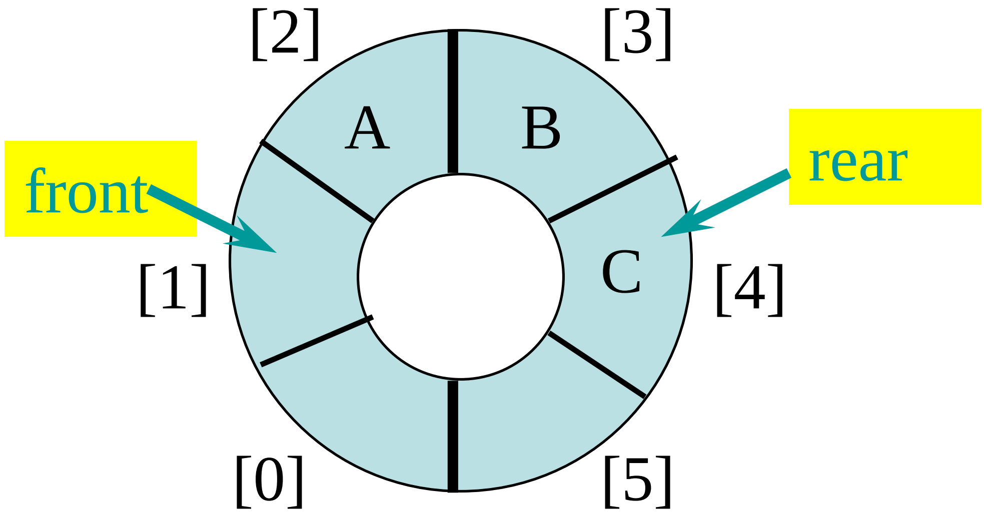
Custom Array Queue

- Use integer variables front and rear.
  - front is one position counterclockwise from first element
  - rear gives position of last element



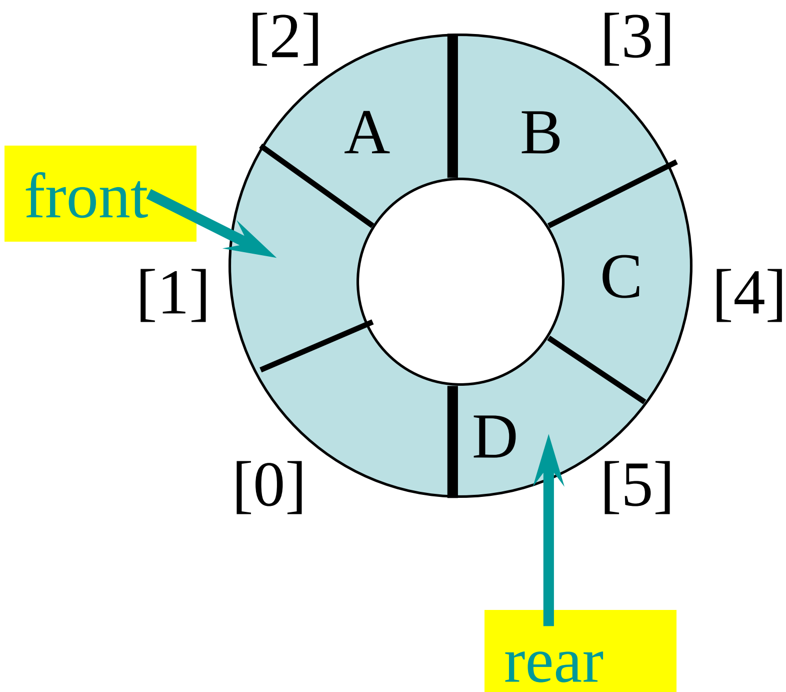
Push An Element

- Move rear one clockwise.



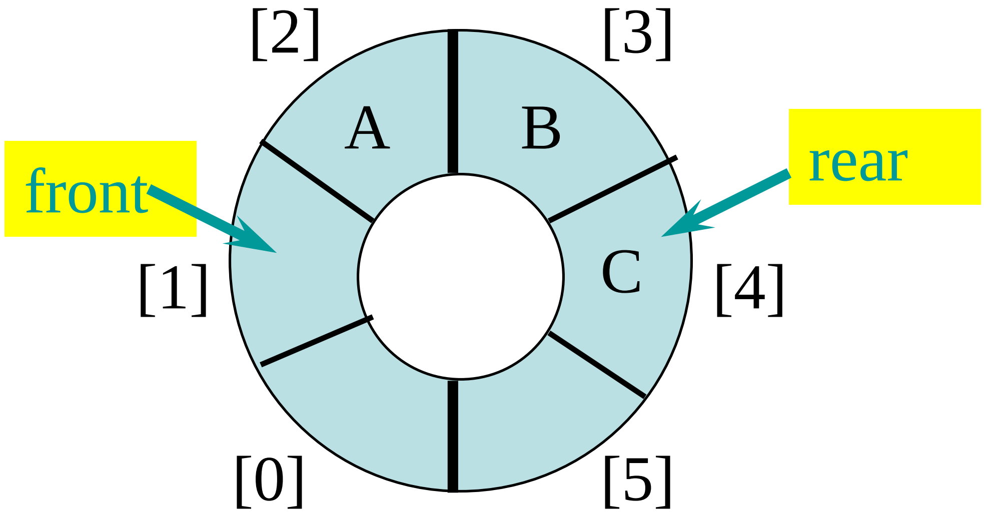
Push An Element

- Move rear one clockwise.
- Then put into queue[rear].



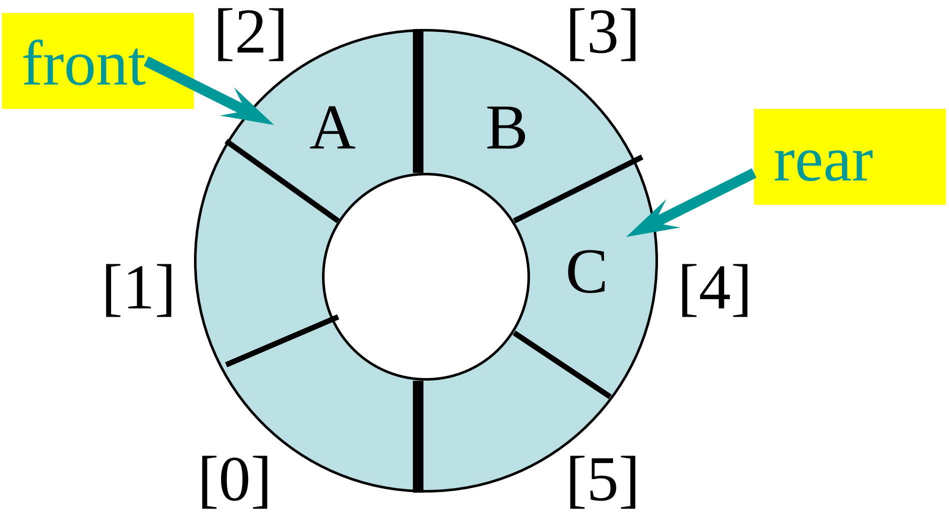
Pop An Element

- Move front one clockwise.



Pop An Element

- Move front one clockwise.
- Then extract from queue[front].

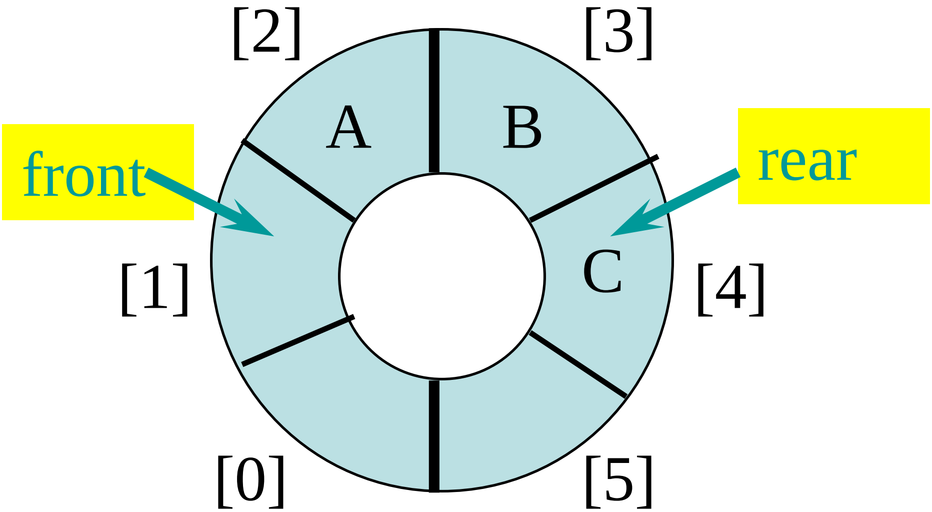


Moving rear Clockwise

- rear += 1

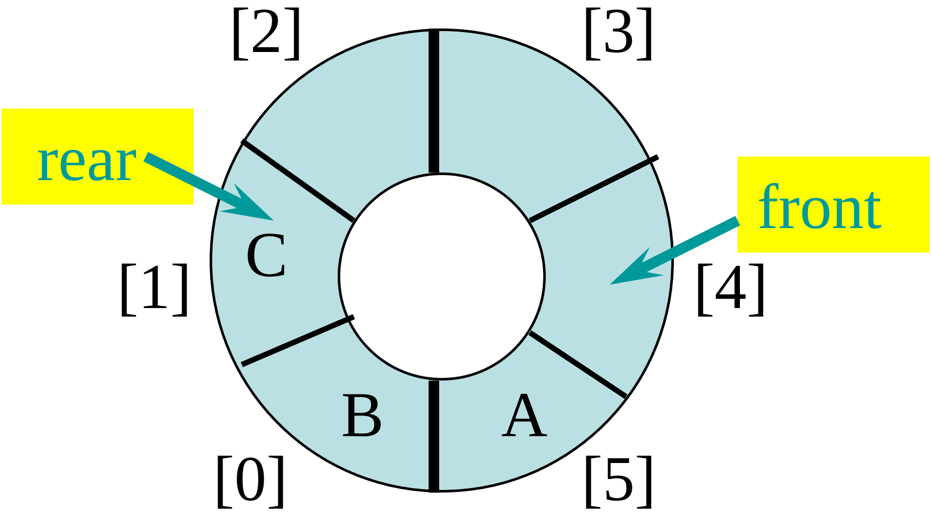
  if rear = = capacity:

      rear = 0

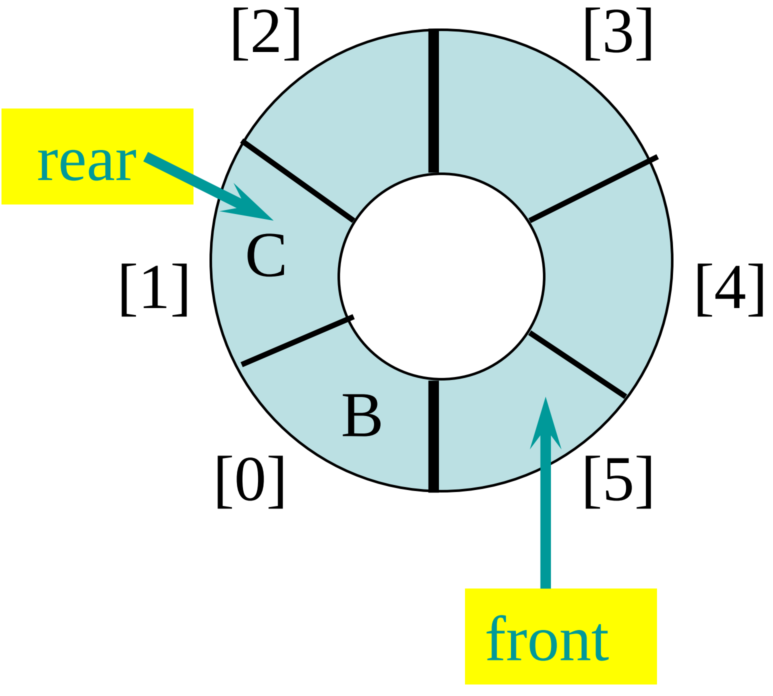


- rear = (rear + 1) % capacity

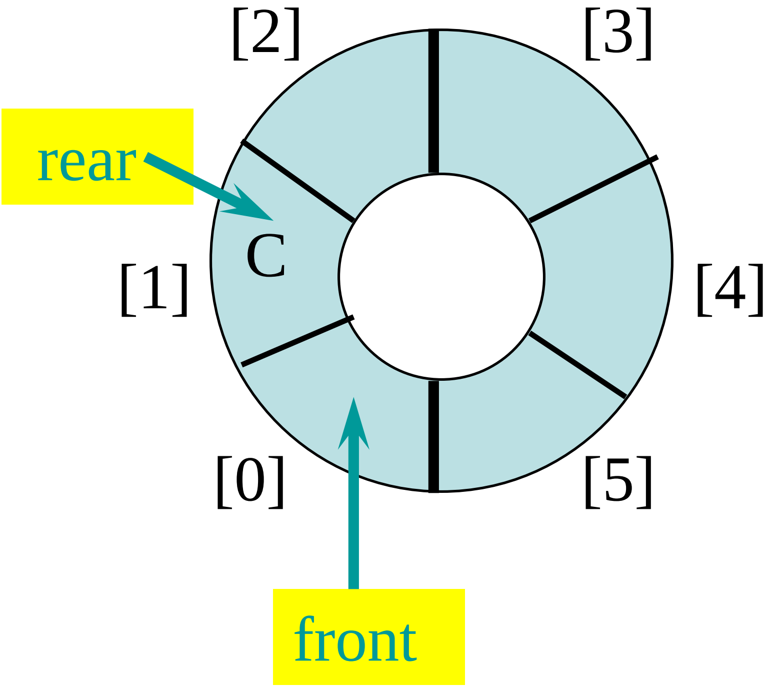
Empty That Queue



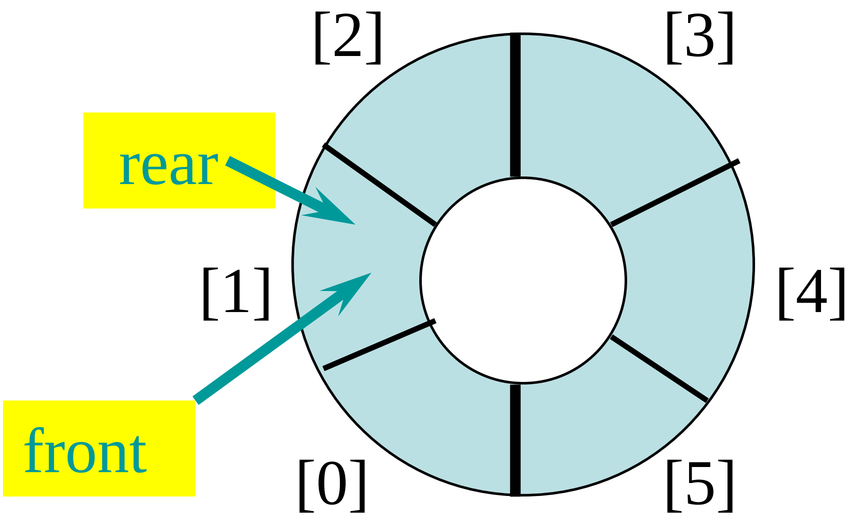
Empty That Queue



Empty That Queue

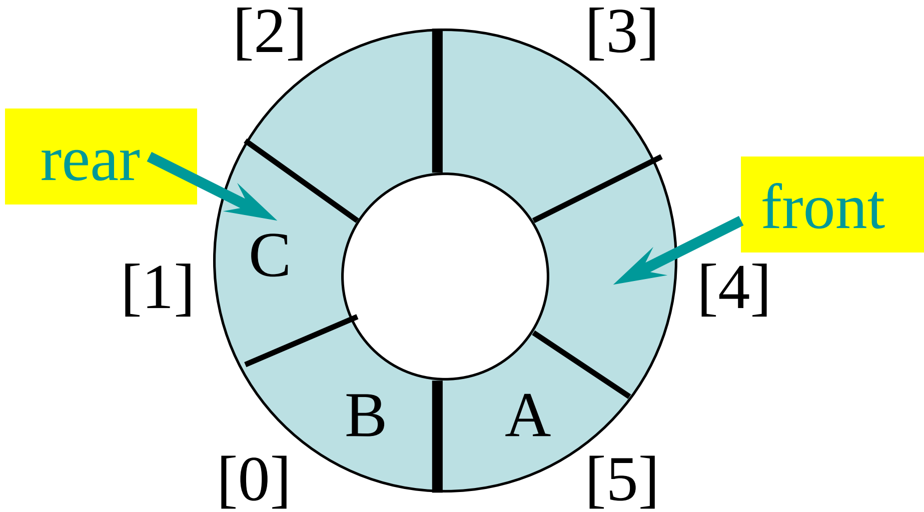


Empty That Queue

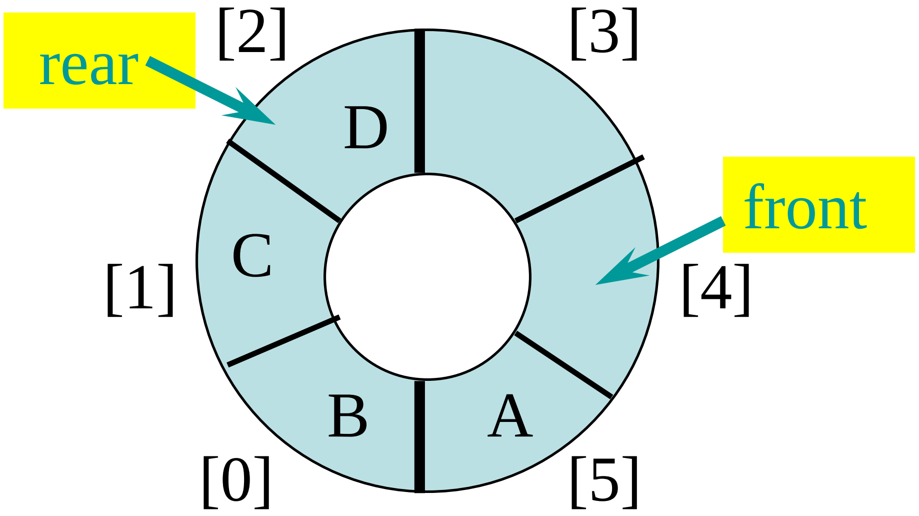


- When a series of removes causes the queue to become empty, front = rear.
- When a queue is constructed, it is empty.
- So initialize front = rear = 0.

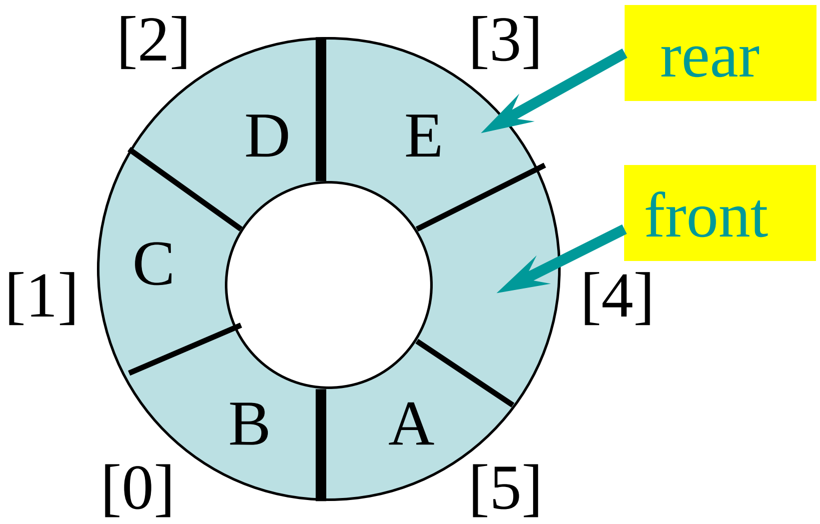
A Full Tank Please



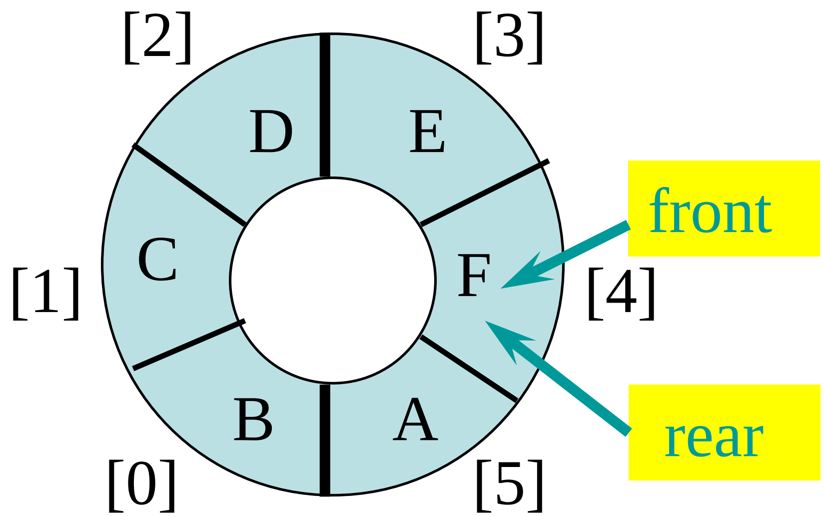
A Full Tank Please



A Full Tank Please



A Full Tank Please



- When a series of adds causes the queue to become full, front = rear.
- So we cannot distinguish between a full queue and an empty queue!

Ouch!!!!!

- Remedies.
  - Don't let the queue get full.
    - When the addition of an element will cause the queue to be full, increase array size.
    - This is what the text does.
  - Define a boolean variable lastOperationIsPush.
    - Following each push set this variable to true.
    - Following each pop set to false.
    - Queue is empty iff (front == rear) && !lastOperationIsPush
    - Queue is full iff (front == rear) && lastOperationIsPush

Ouch!!!!!

- Remedies (continued).
  - Define an integer variable size.
    - Following each push do size += 1.
    - Following each pop do size -= 1.
    - Queue is empty iff (size == 0)
    - Queue is full iff (size == arrayLength)
  - Performance is slightly better when first strategy is used.