Learning Java by Design Examples

PROJECT 10: INDEXING

DR. ERIC CHOU

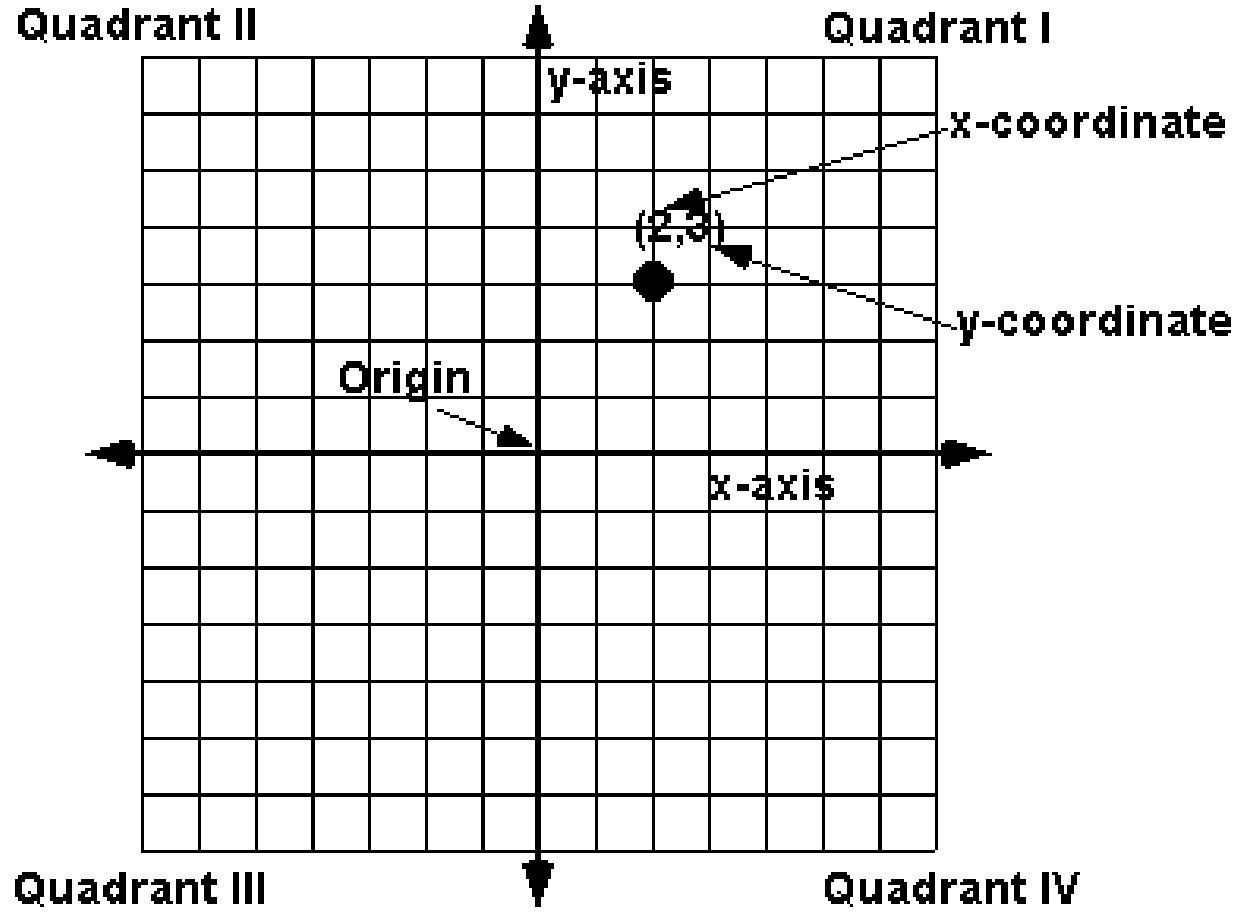
IEEE SENIOR MEMBER

ACTIVITY 1

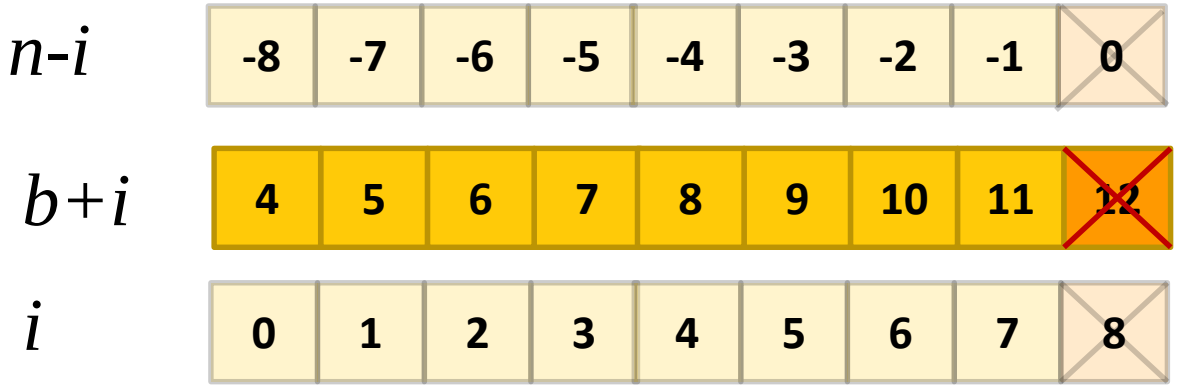
Discrete Space

Cartesian Coordinate

Continuous Space



Base and local index



Index = Base + local_Index

= $b+i$

Index = Length - local_neg_Index

= $n-i$

Base = b = 4, Length = n = $a.length$

Indexing for ASCII code

- Total number of ASCII letters: ‘Z’-’A’+1 (alphabet size)
- Indexing for a letter (‘B’): ‘A’ + (‘B’-’A’)
- Traversing through the whole alphabet: ‘A’ + i; i is from 0 to 25
- Letter indexing is also used for histogram: a[‘X’-’A’] to store the number of occurrence for the letter ‘X’

Sum up an array of 8 elements with index from the last element to the first.

Write a program to sum up

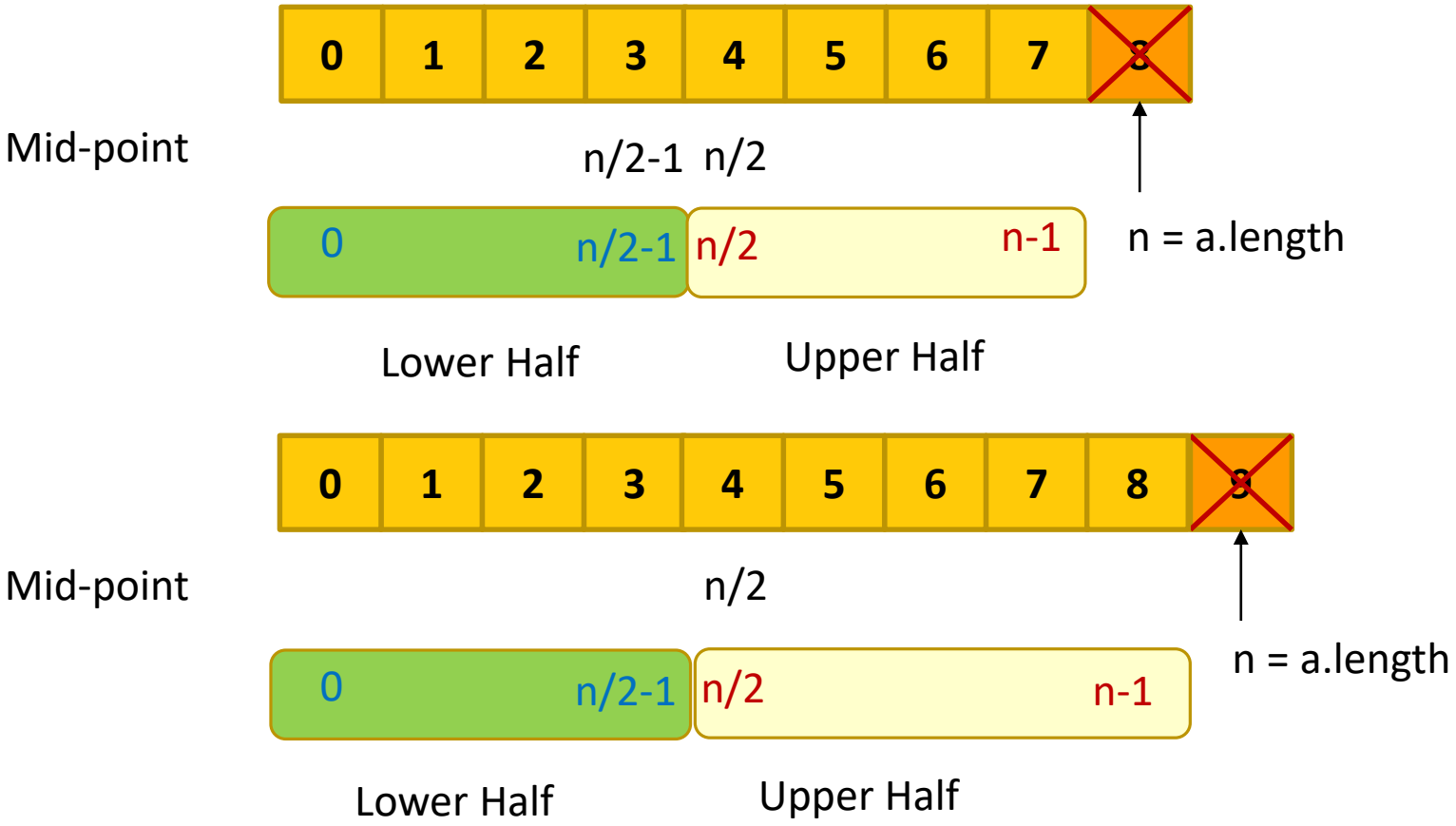
Int[] a = {1, 2, 3, 4, 5, 6, 7, 8};

starting from 8 down to 1 with proper indexing.

ACTIVITY 2

Half an Array

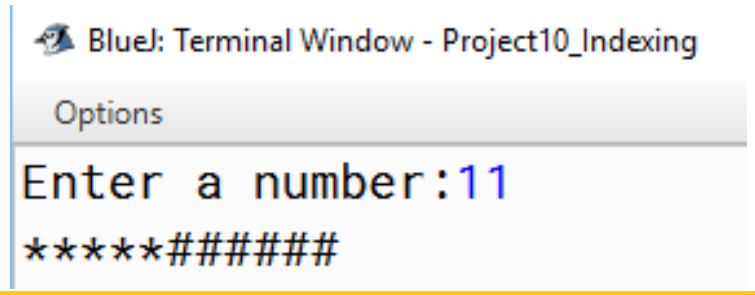
Divide an array into half

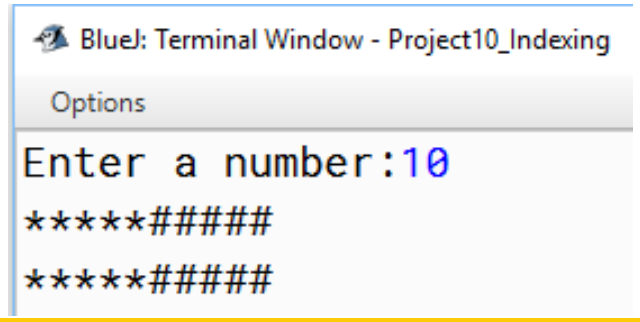


Activity 2:

Divide an index space from 0 to n-1 (totally n elements) in to two halves. If n is odd, the first half should be shorter.

Write a program and enter a number n from console. Print the first half with letter “*”, the second half with letter “#”. If n is odd, “*” should be one less than “#”.



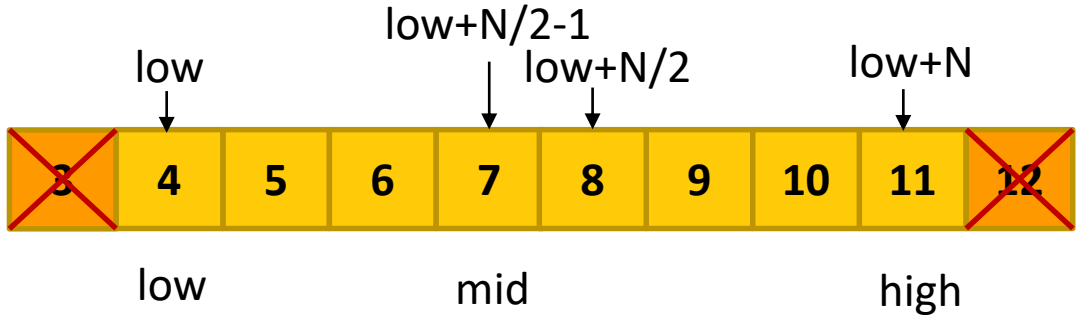


ACTIVITY 3

Binary Search

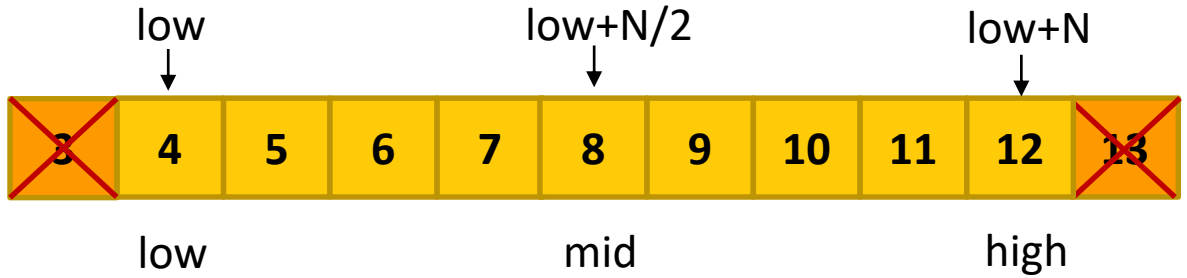
low and high

low+N/2 is high middle
(low+high) is low middle



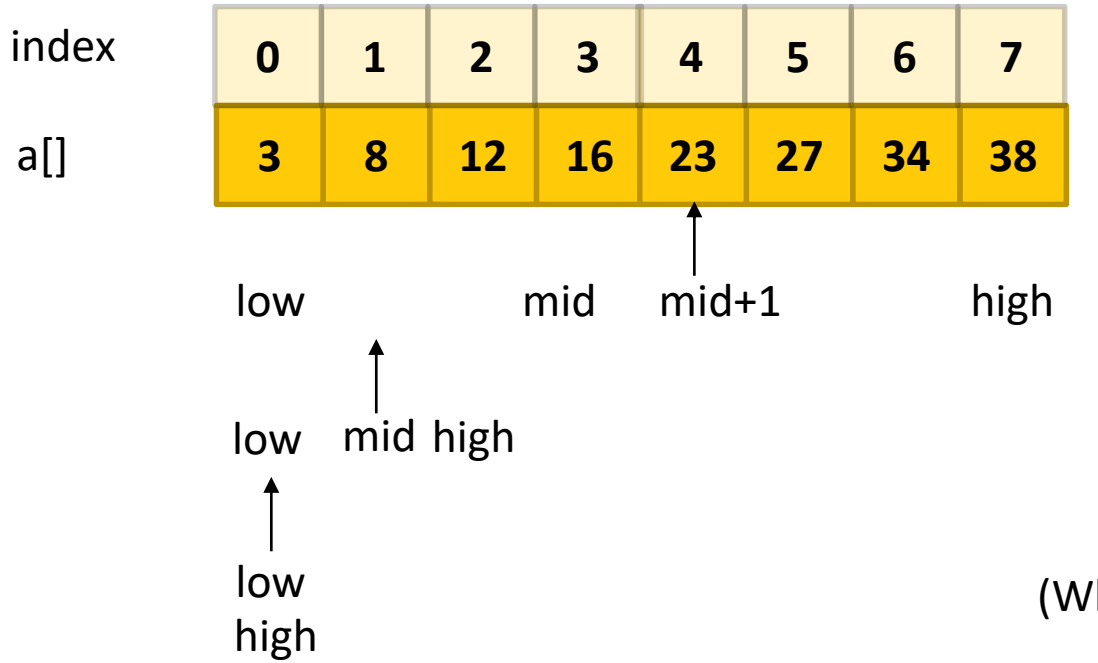
mid is the end of first half

N=Length = high-low+1 = high – (low-1) = 11 – 4 + 1 = 11 – 3 = 8
mid = (low+high)/2 = (2* low + N -1)/ 2 = low + (N-1)/2
First Half = [low, low+N/2-1], Second Half = [low+N/2, low+N-1]



Length = high-low+1 = high – (low-1) = 11 – 4 + 1 = 11 – 3 = 8
mid = (low+high)/2

Activity 4:



Find the index for the key=28.
If it is not found, return -1.

mid = (low + high)/2
Note: low middle is used in here.

Stop when found

(When high == low, stop return whatever)

Activity 3: Find the index of a key value in an Array

Write a program to find the index of a search key. If not found, return -1. Try the following two arrays.

Pre-condition, the array is already sorted.

```
static int[] a = {3,8,12,16,23,34,37,38};
static int[] b = {3,8,12,16,23,34,37,38,60};
```

ACTIVITY 4

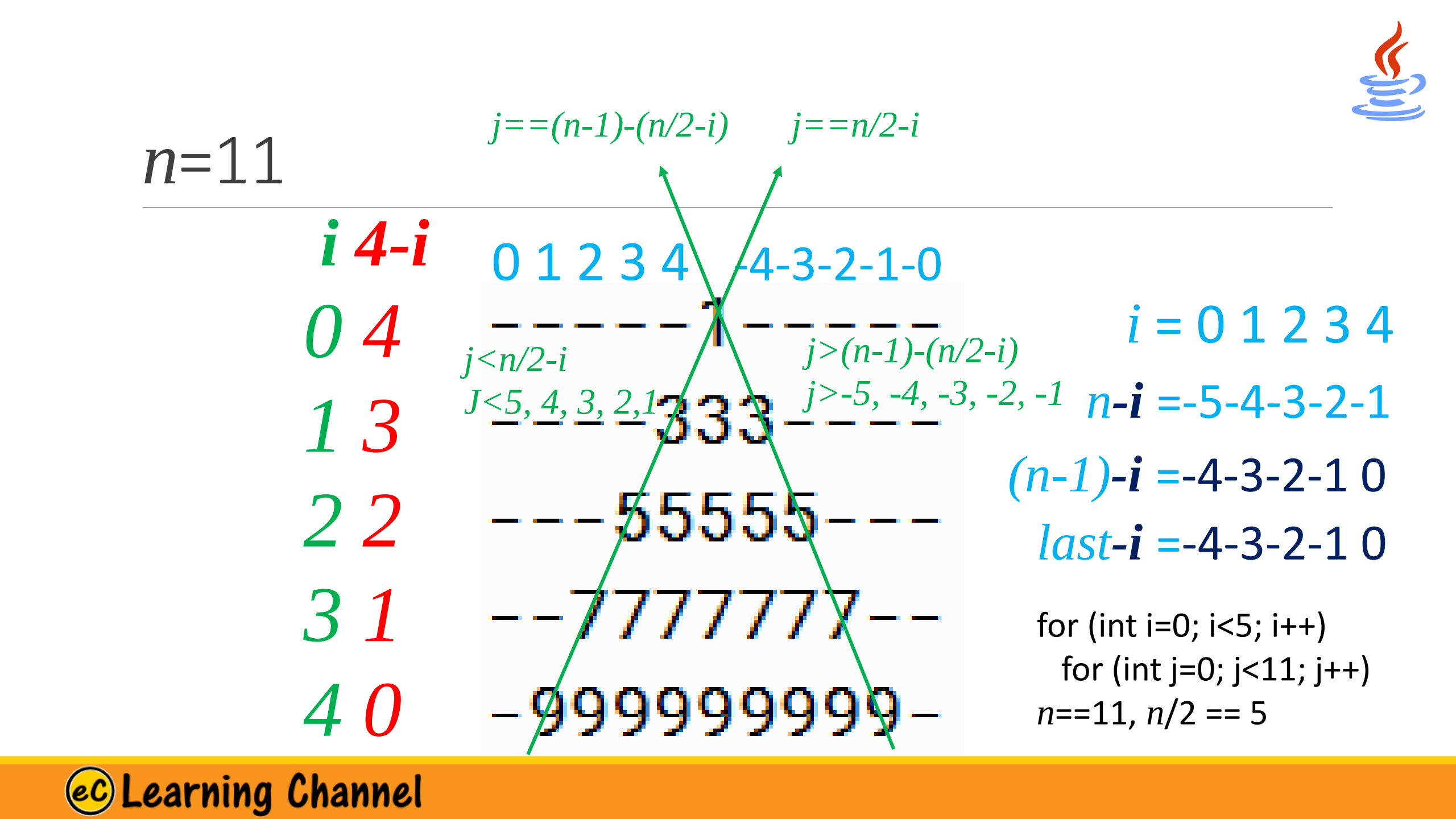
Pyramid Problem

Activity 3:

NumberPyramid

Using nested loops, print the following.

```
-----1-----
----333----
---55555---
--7777777--
-999999999-
```


n=11
j==(n-1)-(n/2-i)
j==n/2-i
i 4-i
0 1 2 3 4 -4-3-2-1-0
0 4
1 3
2 2
3 1
4 0
j<n/2-i
J<5, 4, 3, 2,1
j>(n-1)-(n/2-i)
j>-5, -4, -3, -2, -1
i = 0 1 2 3 4
n-i =-5-4-3-2-1
(n-1)-i =-4-3-2-1 0
last-i =-4-3-2-1 0
for (int i=0; i<5; i++)
for (int j=0; j<11; j++)
n==11, n/2 == 5
eC Learning Channel

ACTIVITY 5

Divide and Conquer

Activity 5:

Many of the patterns we have seen for traversing arrays can also be written recursively. It is not common, but it is a useful exercise.

1. Write a method called **maxInRange** that takes an array of integers and two indexes, **lowIndex** and **highIndex**, and finds the maximum value in the array, but only considering the elements between **lowIndex** and **highIndex**, including both.
   - This method should be recursive. If the length of the range is 1, that is, if **lowIndex == highIndex**, we know immediately that the sole element in the range must be the maximum. So that's the base case.
   - If there is more than one element in the range, we can break the array into two pieces, find the maximum in each of the pieces, and then find the maximum of the maxima.

Activity 5: Think Java Exercise 8-8

1. Methods like **maxInRange** can be awkward to use. To find the largest element in an array, we have to provide the range for the entire array.

```
double max = maxInRange(a, 0, a.length - 1);
```

Write a method called **max** that takes an array and uses **maxInRange** to find and return the largest element.

ACTIVITY 6

Summary

Summary

Concepts covered in this lab:

1. Positive indexing and negative indexing (Count up and count down)
2. Calculation of array length (high-low+1)
3. Mid-point: low mid-point and high mid-point and their application.

   (low+high)/2, **low+N/2**
4. Divide the array into half. [low, low+N/2-1], [low+N/2, high]
5. Binary Search (mid=(low+high)/2)
6. In range operation (Divide and conquer) for recursive or iterative operation.