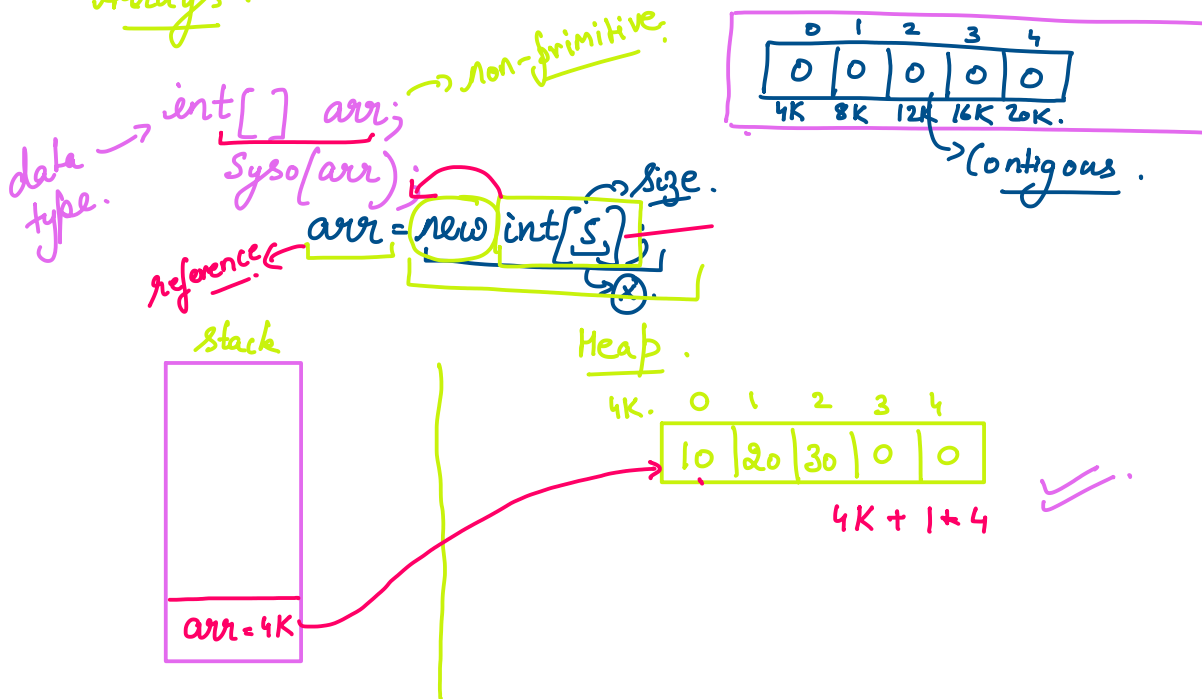


Lecture 4

Saturday, 11 January 2020 2:44 PM

Arrays.

```
public static void main(String[] args) {
```

```
    int[] arr = new int[5];
```

```
    arr[0] = 10;
```

```
    arr[1] = 20;
```

```
    arr[2] = 30;
```

```
    for (int i = 0; i < arr.length; i++) {
        Sys.out(arr[i]);
    }
```

```
    int i = 1; int j = 2;
```

```
    Sys.out(arr[i], arr[j]);
```

```
    swap(arr, i, j);
```

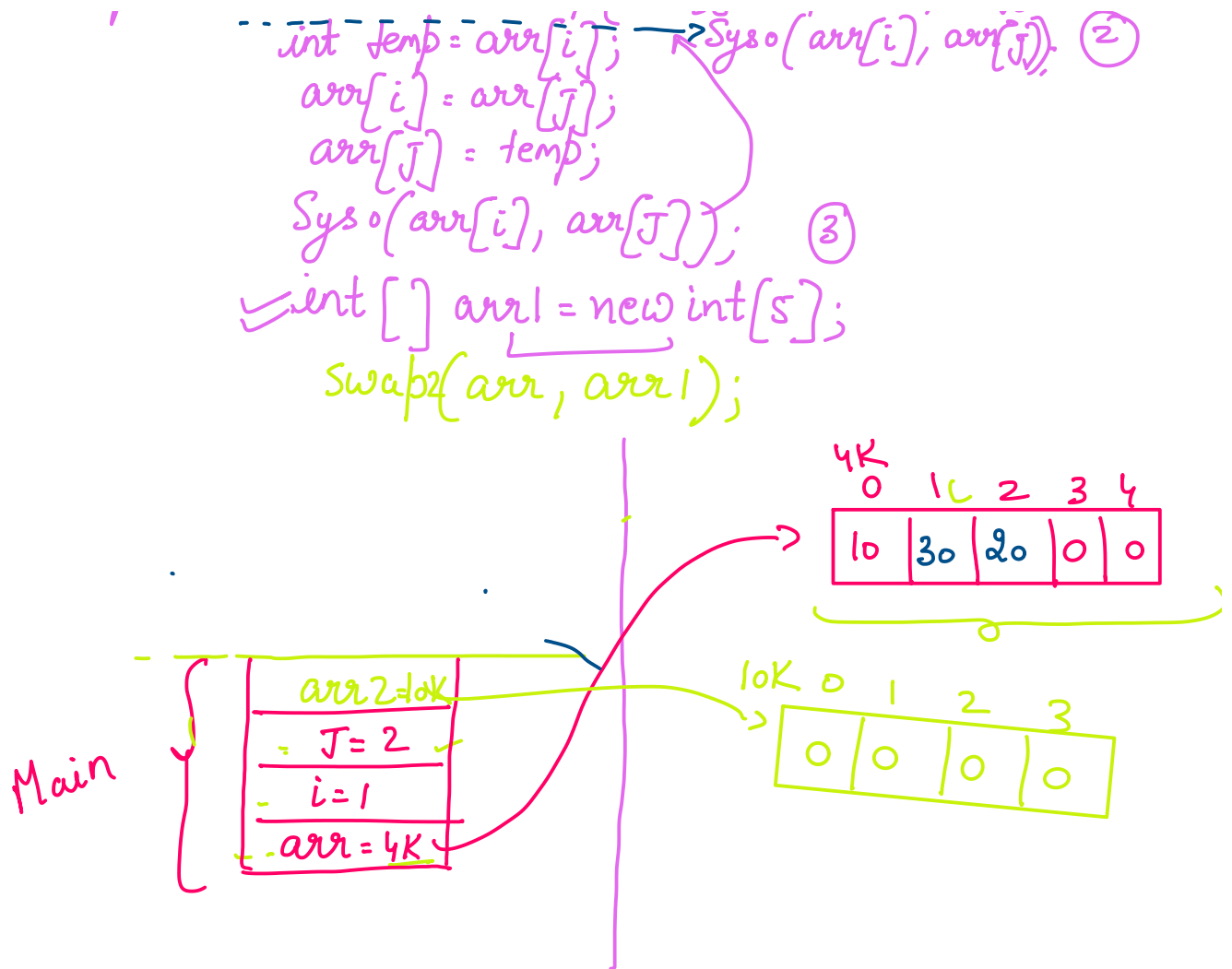
```
    Sys.out(arr[i], arr[j]);
```

```
    int[] arr2 = new int[4];
```

```
}
```

primitive → don't persist things
non- " → can persist things.

```
public static void swap(int[] arr, int i, int j) {
```



arr[1] = 20 value

i = 1; → index

J = 2

Syso(arr.length) // 5 ✓
 Syso(arr[arr.length]); × Runtime Error.
 Syso(arr[arr.length-1]); ○

```

Swap2(int[] arr1, int[] arr2) {
    int[] temp = arr1;
    arr1 = arr2;
    arr2 = temp;
}

```

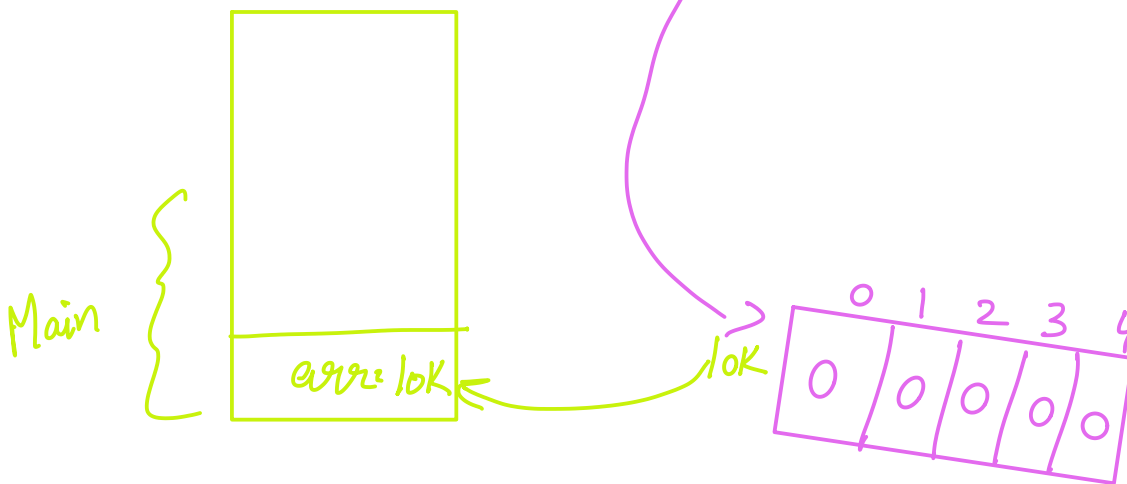
3

```
int[] arr;
```

```
arr = new int[5];
```

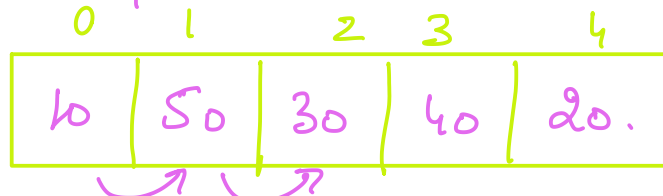
```
o syso(arr[2]); // o
```

```
arr: new int[5];
```



Sating Algorithms.

- ① Bubble Sort
- ② Selection
- ③ Insertion



Bubble Sort. \hookrightarrow Passes $\Rightarrow \underline{n-1} = \text{arr. length} - 1$

```

basses for (int i = 1; i <= arr.length - 1; i++) {
    // ...
}

```

jw-

(4)

3

```

for (int j=0; j < arr.len
    if (arr[j] > arr[j+1])
        swap(j, j+1)
    }

```

Pass 1. \rightarrow 10, 50, 30, 40, 20

10, 30, 50, 40, 20

10, 30, 40, 50, 20

10, 30, 40, 20, 50

Pass 3 \rightarrow 10, 30, 20, 40, 50

10, 30, 20, 40, 50

10, 20, 30, 40, 50

Selection Sort

arr \rightarrow { 10, 30, 40, 50 }

Indices: 0, 1, 2, 3

Arrows point from indices 1 and 2 to the values 30 and 40 respectively, which are circled.

```

for (int i=0; i < arr
    min=i; //1

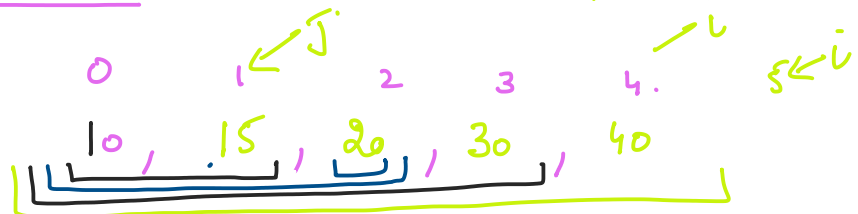
```

```

min_value = arr[i]
for (int j = i+1; j < arr.length; j++)
    if (arr[j] < min_value)
        min_value = arr[j]
    }
    swap(i, min_value);
}

```

Insertion Sort



```

Insertion Sort (int[] arr) {

```

```

    for (int i = 1; i < arr.length; i++) {
        for (int j = i; j > 0 && arr[j] < arr[j-1]; j--)
            swap(j, j-1);
    }
}

```

arr = { 3, -5, -1, 10, 4, 2 }

Max Subarray Sum *

3
 3, -5
 3, -5, -1
 -5, -1
 -5, -1, 10, 4
 3, -5, -1, 10, 4, 2

3, -5 =>
 3, -5, -

3, -5, -

```

public static int approach1(int[] arr) {
    int max = -∞
    int sum = 0;
    for (int i = 0; i < arr.length; i++) {
        for (int j = i; j < arr.length; j++) {
            int sum = 0;
            for (int k = i; k <= j; k++) {
                sum = sum + arr[k];
            }
            if (sum > max) {
                max = sum;
            }
        }
    }
}

```

Diagram for approach 1: Array [3, -5, -1, 10, 4]. A box highlights the subarray [3, -5, -1] with indices 0, 1, 2. A calculation (10) + 4 = 16 is shown. Another calculation // 3 - 2 = 1 is shown. A third calculation // 3 - 2 > 3 is shown.

```

public static int approach2(int[] arr) {
    int sum = 0
    int max = -∞
    for (int i = 0; i < arr.length; i++) {
        for (int j = i; j < arr.length; j++) {
            sum = sum + arr[j];
            if (sum > max) {
                max = sum;
            }
        }
    }
    return max;
}

```

Diagram for approach 2: Array [3, -5, -1, 10, 4, 2]. A box highlights the subarray [3, -5, -1, 10, 4, 2] with indices 0 to 5. A calculation (13) is shown. A calculation // 3 is shown. A calculation Sum = 0; is shown.

Kadane's

```

public static int approach3(int[] arr) {
    int max = -∞
    int sum = 0;
    for (int i = 0; i < arr.length; i++) {
        sum = sum + arr[i];
        if (sum > max) {
            max = sum;
        }
    }
}

```

Diagram for approach 3: Array [0, 1, 2, 3, 4, 5]. A box highlights the subarray [0, 1, 2, 3, 4, 5] with indices 0 to 5. A calculation // 3 is shown.

$\{3, -8, -1, 10, 4, 2\}$

OneNote

max = sum

// 3 11

if (sum < 0) {

sum = 0;

// 0

}

}

}