

# מבני נתונים

תרגול 3 – מיון בסיס, תור ומסחנית

# היום

- מיון בסיס Radix Sort

- מחסנית

- תור

- עבודה עצמית

# מיון מבוסס השוואות

Comparison Sort



כל אלגוריתמי המיזון המבוססים על פעולת השוואה דורשים לפחות  
כל אלגוריתמי המיזון המבוססים על פעולת השוואה דורשים לפחות

פעולות השוואה במקרה הגרוע על-מנת לבצעם.  
פעולות השוואה במקרה הגרוע על-מנת לבצעם.  
פעולות השוואה במקרה הגרוע על-מנת לבצעם.

כל אלגוריתמי המיזון המבוססים על פעולת השוואה דורשים לפחות  
 $\Omega(n \cdot \log(n))$   
פעולות השוואה במקרה הגרוע על-מנת לבצעם

# מיון מבוסס השוואות

## Comparison Sort

מיון מבוסס השוואות מסדר אלמנטים במערך ע"י השוואות, בדרך כלל ע"י האופרטור  $\leq$

### מיונים לא מבוססי השוואות

Counting Sort ○

Radix Sort ○

...

### מיונים מבוססי השוואות

Bubble Sort מיון בועות ○

Selection Sort מיון בחירה ○

Insertion Sort מיון הכנסה ○

Merge Sort מיון מיזוג ○

Quick Sort מיון מהיר ○

...

# Counting Sort - מיון מנייה

אלגוריתם מיון עבור מספרים שלמים המתבסס על העובדה  
שהמספרים נמצאים בטווח חסום כדי לבצע את המיון בזמן מהיר  
יותר מזה שמסוגלים לו אלגוריתמי המיון הכלליים.

קלט: מערך חד-מימדי

פלט: מערך חד-מימדי ממויין

# מיון יציב Stable Sort

מיון נקרא **מיון יציב** אם הוא שומר על הסדר של הנתונים לאחר המיון גם כשיש שני נתונים זהים.

דוגמה להבחנה בין מיון יציב ללא-יציב:

אם רוצים למיין רשימת שמות על פי שם משפחה, אך אם שמות המשפחה זהים, למיין על פי השם הפרטי, אפשר למיין את המערך על פי שם פרטי ואחר כך למיין שוב על פי שם המשפחה. דבר זה יתאפשר רק אם המיון הוא יציב, אך אם הוא אינו יציב, אזי יכול להיות שהסדר הפנימי של השמות הפרטיים ייהרס.

# Radix Sort - מיון בסיס

## Radix-Sort (A, d)

for  $i \leftarrow 1$  to  $d$

Counting Sort יעיל במקרה שיש מספר קטן של ספרות.

Use Stable Sort algo to sort A on digit i

$$O(d \cdot (n + k))$$

כמות ספרות  
לדוגמא ל-מספר 555 יש 3 ספרות

כמות האיברים במערך



# Radix Sort - מיון בסיס

329

457

657

839

436

720

355

# Radix Sort - מיון בסיס

329	720
457	355
657	436
839 →	457
436	657
720	329
355	839

# Radix Sort - מיון בסיס

329	720	720
457	355	329
657	436	436
839 →	457 →	839 →
436	657	355
720	329	457
355	839	657

# Radix Sort - מיון בסיס

329	720	720	329
457	355	329	355
657	436	436	436
839	457	839	457
436	657	355	657
720	329	457	720
355	839	657	839

Sorted

# שאלה

בהינתן  $n$  מספרים שלמים בטווח  $[1, n^3]$ , הציעו אלגוריתם יעיל למיון האיברים

הצעה: מיון מבוסס השוואות  $O(n \cdot \log n)$

הצעה: מיון מנייה Counting Sort  $O(n + n^3) = O(n^3)$

הצעה: מיון בסיס Radix Sort

כדי לייצג את המספר 1521 יש צורך ב-  $4 = \lfloor \log_{10}(1521) \rfloor + 1$  ספרות

כדי לייצג את  $n^3$  יש צורך ב-  $O(\log n) = \lfloor 3\log_{10}(n) \rfloor + 1 = \lfloor \log_{10}(n^3) \rfloor + 1$  ספרות

$$d = \log n, k = 10 \Rightarrow O(d \cdot (k + n)) = O(\log n (10 + n)) = O(n \log n)$$

# שאלה

בהינתן  $n$  מספרים שלמים בטווח  $[1, n^3]$ , הציעו אלגוריתם יעיל למיון האיברים

הצעה: מיון מבוסס השוואות  $O(n \cdot \log n)$

הצעה: מיון מנייה Counting Sort  $O(n + n^3) = O(n^3)$

הצעה: מיון בסיס Radix Sort **בסיס אחר?**

כדי לייצג את  $n^3$  בבסיס  $n$  יש צורך ב  $\lfloor \log_n(n^3) \rfloor + 1 = \lfloor 3\log_n(n) \rfloor + 1 = 3 + 1 = 4$  ספרות

$$d = 4, k = n \Rightarrow O(d \cdot (k + n)) = O(4(n + n)) = O(n)$$

# שאלה

בהינתן  $n$  אותיות קטנות באנגלית, הציעו אלגוריתם יעיל למיון לקסיקוגרפי כאשר לא כל המילים באותו אורך

1. יהי  $m$  להיות האורך של המילה המקסמלית בקלט.  $m$  הוא קבוע

2. נייצג כל אות בכל מילה בקלט כספרה בבסיס 27  $([0,26])$

כאשר  $a = 1, b = 2, \dots, z = 26$

3. כל אות עם פחות מ- $m$  אותיות – נוסיף אפסים על מנת להשלים ל- $m$

4. נבצע מיון בסיס עבור  $d = m$  ו- $k = 27$

$$d = m, k = 27 \Rightarrow O(d \cdot (k + n)) = O(m(27 + n)) = O(n)$$

5. החזר בחזרה את המספרים למילים

# שאלה

בהינתן  $n$  אותיות קטנות באנגלית, הציעו אלגוריתם יעיל למיון לקסיקוגרפי כאשר לא כל המילים באותו אורך

Lexicographical-sort input: {blue, red, green}

Radix-sort input: { (2,12, 21,5,0), (18,5,4,0,0), (7, 18,5,5,14) }

Radix-sort output: { (2,12, 21,5,0), (7, 18,5,5,14) , (18,5,4,0,0) }

Lexicographical-sort output: {blue, green, red}



פעולות השוואה במקרה הגרוע על-מנת לבצעם.  
פעולות השוואה במקרה הגרוע על-מנת לבצעם.  
פעולות השוואה במקרה הגרוע על-מנת לבצעם.

כל אלגוריתמי המיון המבוססים על פעולת השוואה דורשים לפחות  
כל אלגוריתמי המיון המבוססים על פעולת השוואה דורשים לפחות  
כל אלגוריתמי המיון המבוססים על פעולת השוואה דורשים לפחות

# לסיכום

	מבוסס השוואות					לא מבוסס השוואות	
	Bubble Sort	Selection Sort	Insertion Sort	Merge Sort	Quick Sort	Radix Sort	Counting Sort
$O$	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(n \cdot \log n)$	$O(n^2)$	$O(d \cdot (n + k))$	$O(n + k)$
$\Omega$	$\Omega(n^2)$ ניתן גם כן ב- $\Omega(n)$	$\Omega(n^2)$	$\Omega(n)$	$\Omega(n \cdot \log n)$	$\Omega(n \cdot \log n)$	$\Omega(d \cdot (n + k))$	$\Omega(n + k)$
$\Theta$	$\Theta(n^2)$	$\Theta(n^2)$		$\Theta(n \cdot \log n)$	$\Theta(n \cdot \log n)$ בהסתברות גבוהה	$\Theta(d \cdot (n + k))$	$\Theta(n + k)$

פעולות השוואה במקרה הגרוע על-מנת לבצעם.  
פעולות השוואה במקרה הגרוע על-מנת לבצעם.  
פעולות השוואה במקרה הגרוע על-מנת לבצעם.

כל אלגוריתמי המיון המבוססים על פעולת השוואה דורשים לפחות  
כל אלגוריתמי המיון המבוססים על פעולת השוואה דורשים לפחות  
כל אלגוריתמי המיון המבוססים על פעולת השוואה דורשים לפחות

# לסיכום

	מבוסס השוואות					לא מבוסס השוואות	
	Bubble Sort	Selection Sort	Insertion Sort	Merge Sort	Quick Sort	Radix Sort	Counting Sort
$O$	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(n \cdot \log n)$	$O(n^2)$	$O(d \cdot (n + k))$	$O(n + k)$
$\Omega$	$\Omega(n^2)$ ניתן גם כן ב- $\Omega(n)$	$\Omega(n^2)$	$\Omega(n)$	$\Omega(n \cdot \log n)$	$\Omega(n \cdot \log n)$	$\Omega(d \cdot (n + k))$	$\Omega(n + k)$
$\Theta$	$\Theta(n^2)$	$\Theta(n^2)$		$\Theta(n \cdot \log n)$	$\Theta(n \cdot \log n)$ בהסתברות גבוהה	$\Theta(d \cdot (n + k))$	$\Theta(n + k)$

פעולות השוואה במקרה הגרוע על-מנת לבצעם.  
פעולות השוואה במקרה הגרוע על-מנת לבצעם.  
פעולות השוואה במקרה הגרוע על-מנת לבצעם.

כל אלגוריתמי המיון המבוססים על פעולת השוואה דורשים לפחות  
כל אלגוריתמי המיון המבוססים על פעולת השוואה דורשים לפחות  
כל אלגוריתמי המיון המבוססים על פעולת השוואה דורשים לפחות

# לסיכום

	מבוסס השוואות					לא מבוסס השוואות	
	Bubble Sort	Selection Sort	Insertion Sort	Merge Sort	Quick Sort	Radix Sort	Counting Sort
$O$	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(n \cdot \log n)$	$O(n^2)$	$O(d \cdot (n + k))$	$O(n + k)$
$\Omega$	$\Omega(n^2)$ ניתן גם כן ב- $\Omega(n)$	$\Omega(n^2)$	$\Omega(n)$	$\Omega(n \cdot \log n)$	$\Omega(n \cdot \log n)$	$\Omega(d \cdot (n + k))$	$\Omega(n + k)$
$\Theta$	$\Theta(n^2)$	$\Theta(n^2)$		$\Theta(n \cdot \log n)$	$\Theta(n \cdot \log n)$ בהסתברות גבוהה	$\Theta(d \cdot (n + k))$	$\Theta(n + k)$

פעולות השוואה במקרה הגרוע על-מנת לבצעם.  
פעולות השוואה במקרה הגרוע על-מנת לבצעם.  
פעולות השוואה במקרה הגרוע על-מנת לבצעם.

כל אלגוריתמי המיון המבוססים על פעולת השוואה דורשים לפחות  
כל אלגוריתמי המיון המבוססים על פעולת השוואה דורשים לפחות  
כל אלגוריתמי המיון המבוססים על פעולת השוואה דורשים לפחות

# לסיכום

	מבוסס השוואות					לא מבוסס השוואות	
	Bubble Sort	Selection Sort	Insertion Sort	Merge Sort	Quick Sort	Radix Sort	Counting Sort
$O$	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(n \cdot \log n)$	$O(n^2)$	$O(d \cdot (n + k))$	$O(n + k)$
$\Omega$	$\Omega(n^2)$ ניתן גם כן ב- $\Omega(n)$	$\Omega(n^2)$	$\Omega(n)$	$\Omega(n \cdot \log n)$	$\Omega(n \cdot \log n)$	$\Omega(d \cdot (n + k))$	$\Omega(n + k)$
$\Theta$	$\Theta(n^2)$	$\Theta(n^2)$		$\Theta(n \cdot \log n)$	$\Theta(n \cdot \log n)$ בהסתברות גבוהה	$\Theta(d \cdot (n + k))$	$\Theta(n + k)$

פעולות השוואה במקרה הגרוע על-מנת לבצעם.  
פעולות השוואה במקרה הגרוע על-מנת לבצעם.  
פעולות השוואה במקרה הגרוע על-מנת לבצעם.

כל אלגוריתמי המיון המבוססים על פעולת השוואה דורשים לפחות  
כל אלגוריתמי המיון המבוססים על פעולת השוואה דורשים לפחות  
כל אלגוריתמי המיון המבוססים על פעולת השוואה דורשים לפחות

# לסיכום

	מבוסס השוואות					לא מבוסס השוואות	
	Bubble Sort	Selection Sort	Insertion Sort	Merge Sort	Quick Sort	Radix Sort	Counting Sort
$O$	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(n \cdot \log n)$	$O(n^2)$	$O(d \cdot (n + k))$	$O(n + k)$
$\Omega$	$\Omega(n^2)$ ניתן גם כן ב- $\Omega(n)$	$\Omega(n^2)$	$\Omega(n)$	$\Omega(n \cdot \log n)$	$\Omega(n \cdot \log n)$	$\Omega(d \cdot (n + k))$	$\Omega(n + k)$
$\Theta$	$\Theta(n^2)$	$\Theta(n^2)$		$\Theta(n \cdot \log n)$	$\Theta(n \cdot \log n)$ בהסתברות גבוהה	$\Theta(d \cdot (n + k))$	$\Theta(n + k)$

פעולות השוואה במקרה הגרוע על-מנת לבצעם.  
פעולות השוואה במקרה הגרוע על-מנת לבצעם.  
פעולות השוואה במקרה הגרוע על-מנת לבצעם.

כל אלגוריתמי המיון המבוססים על פעולת השוואה דורשים לפחות  
כל אלגוריתמי המיון המבוססים על פעולת השוואה דורשים לפחות  
כל אלגוריתמי המיון המבוססים על פעולת השוואה דורשים לפחות

# לסיכום

	מבוסס השוואות					לא מבוסס השוואות	
	Bubble Sort	Selection Sort	Insertion Sort	Merge Sort	Quick Sort	Radix Sort	Counting Sort
$O$	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(n \cdot \log n)$	$O(n^2)$	$O(d \cdot (n + k))$	$O(n + k)$
$\Omega$	$\Omega(n^2)$ ניתן גם כן ב- $\Omega(n)$	$\Omega(n^2)$	$\Omega(n)$	$\Omega(n \cdot \log n)$	$\Omega(n \cdot \log n)$	$\Omega(d \cdot (n + k))$	$\Omega(n + k)$
$\Theta$	$\Theta(n^2)$	$\Theta(n^2)$		$\Theta(n \cdot \log n)$	$\Theta(n \cdot \log n)$ בהסתברות גבוהה	$\Theta(d \cdot (n + k))$	$\Theta(n + k)$

פעולות השוואה במקרה הגרוע על-מנת לבצעם.  
פעולות השוואה במקרה הגרוע על-מנת לבצעם.  
פעולות השוואה במקרה הגרוע על-מנת לבצעם.

כל אלגוריתמי המיון המבוססים על פעולת השוואה דורשים לפחות  
כל אלגוריתמי המיון המבוססים על פעולת השוואה דורשים לפחות  
כל אלגוריתמי המיון המבוססים על פעולת השוואה דורשים לפחות

# לסיכום

	מבוסס השוואות					לא מבוסס השוואות	
	Bubble Sort	Selection Sort	Insertion Sort	Merge Sort	Quick Sort	Radix Sort	Counting Sort
$O$	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(n \cdot \log n)$	$O(n^2)$	$O(d \cdot (n + k))$	$O(n + k)$
$\Omega$	$\Omega(n^2)$ ניתן גם כן ב- $\Omega(n)$	$\Omega(n^2)$	$\Omega(n)$	$\Omega(n \cdot \log n)$	$\Omega(n \cdot \log n)$	$\Omega(d \cdot (n + k))$	$\Omega(n + k)$
$\Theta$	$\Theta(n^2)$	$\Theta(n^2)$		$\Theta(n \cdot \log n)$	$\Theta(n \cdot \log n)$ בהסתברות גבוהה	$\Theta(d \cdot (n + k))$	$\Theta(n + k)$

פעולות השוואה במקרה הגרוע על-מנת לבצעם.  
פעולות השוואה במקרה הגרוע על-מנת לבצעם.  
פעולות השוואה במקרה הגרוע על-מנת לבצעם.

כל אלגוריתמי המיון המבוססים על פעולת השוואה דורשים לפחות  
כל אלגוריתמי המיון המבוססים על פעולת השוואה דורשים לפחות  
כל אלגוריתמי המיון המבוססים על פעולת השוואה דורשים לפחות

# מחסנית

## Stacks

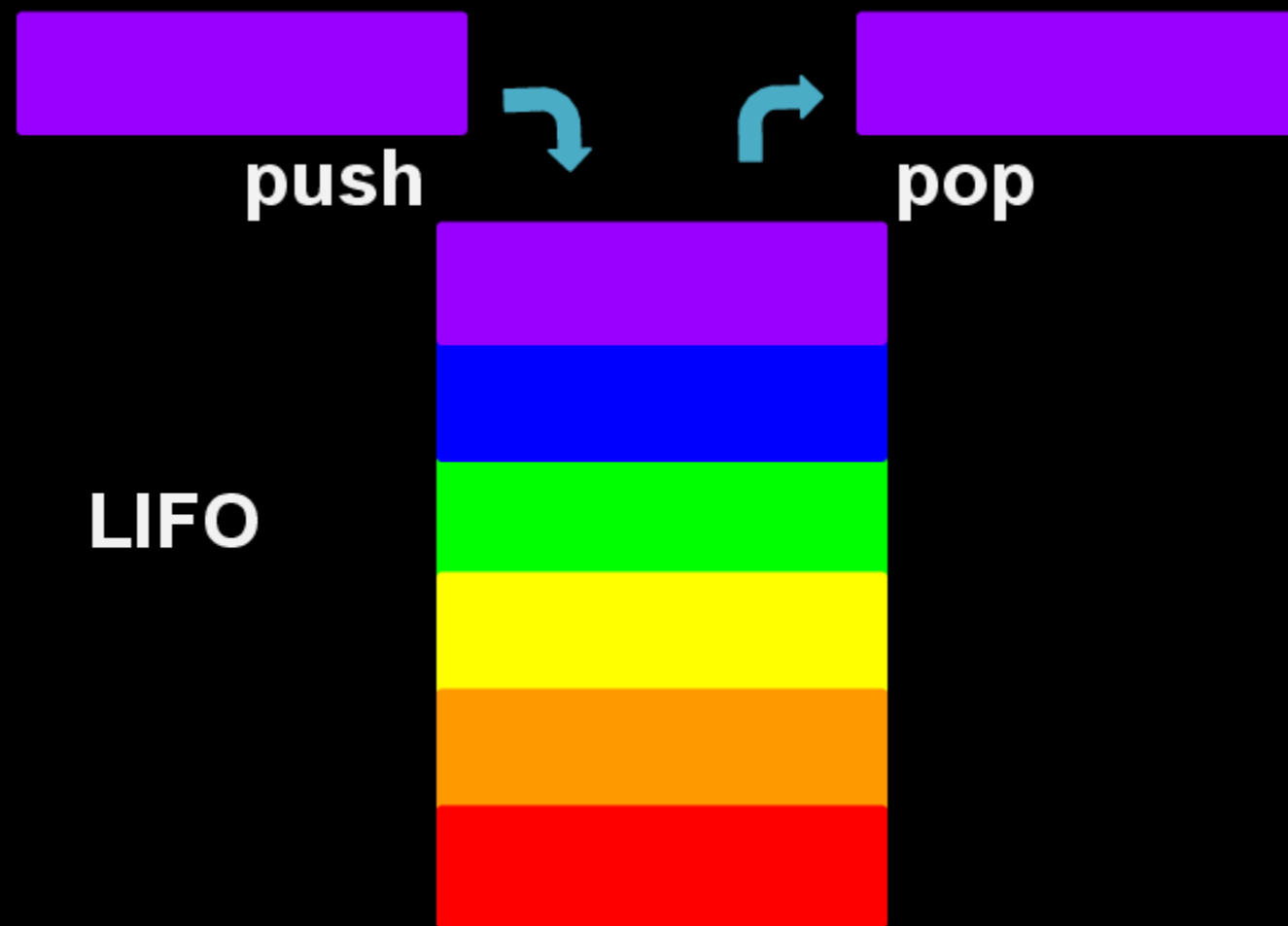




פעולות השוואה במקרה הגרוע על-מנת לבצעם.  
פעולות השוואה במקרה הגרוע על-מנת לבצעם.  
פעולות השוואה במקרה הגרוע על-מנת לבצעם.

כל אלגוריתמי המיון המבוססים על פעולת השוואה דורשים לפחות  
כל אלגוריתמי המיון המבוססים על פעולת השוואה דורשים לפחות  
כל אלגוריתמי המיון המבוססים על פעולת השוואה דורשים לפחות

# מחסנית



# מחסנית Stack

```
class Stack<T> {
```

```
    ...
```

```
}
```

# Java ArrayList

The ArrayList class is a resizable array, which can be found in the java.util package.

## Constructor Summary

### **ArrayList()**

Constructs an empty list.

### **ArrayList(Collection c)**

Constructs a list containing the elements of the specified collection, in the order they are returned by the collection's iterator.

### **ArrayList(int initialCapacity)**

Constructs an empty list with the specified initial capacity.

## Method Summary

### **void add(int index, Object element)**

Inserts the specified element at the specified position in this list.

### **boolean add(Object o)**

Appends the specified element to the end of this list.

### **boolean addAll(Collection c)**

Appends all of the elements in the specified Collection to the end of this list, in the order that they are returned by the specified Collection's Iterator.

### **boolean addAll(int index, Collection c)**

Inserts all of the elements in the specified Collection into this list, starting at the specified position.

### **void clear()**

Removes all of the elements from this list.

### **Object clone()**

Returns a shallow copy of this ArrayList instance.

### **boolean contains(Object elem)**

Returns true if this list contains the specified element.

### **void ensureCapacity(int minCapacity)**

Increases the capacity of this ArrayList instance, if necessary, to ensure that it can hold at least the number of elements specified by the minimum capacity argument.

### **Object get(int index)**

Returns the element at the specified position in this list.

### **int indexOf(Object elem)**

Searches for the first occurrence of the given argument, testing for equality using the equals method.

### **boolean isEmpty()**

Tests if this list has no elements.

### **int lastIndexOf(Object elem)**

Returns the index of the last occurrence of the specified object in this list.

### **Object remove(int index)**

Removes the element at the specified position in this list.

### **protected void removeRange(int fromIndex, int toIndex)**

Removes from this List all of the elements whose index is between fromIndex, inclusive and toIndex, exclusive.

### **Object set(int index, Object element)**

Replaces the element at the specified position in this list with the specified element.

# Javadoc - ArrayList

[API](#) documentation in [HTML](#) format from [Java](#) source code.

# מחסנית Stack

```
class Stack<T> {
```

תכנות גנרי הוא הדרך לכתיבת תוכניות שאינן תלויות בטיפוסי המשתנים.

```
private ArrayList<T> arr;
```

```
private int size = 0;
```

```
...
```

```
}
```

Queue will be implemented with static array

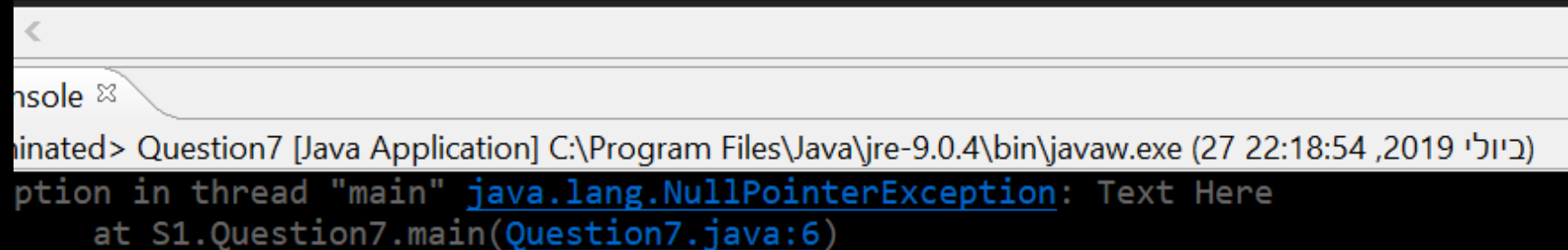
# מחסנית Stack

```
class Stack<T> {  
  
    // Constructor  
    public Stack(int capacity) {  
        arr = new ArrayList<T>(capacity);  
    }  
  
    ...  
}
```

# מחסנית Stack

```
class Stack<T> {  
    // Adds new element to the Stack  
    // Input: element of type T  
    public void push(T element) {  
        if(element == null)  
            throw new NullPointerException();  
        arr.add(size++, element);  
    }  
    ...  
}  
  
    public static void main(String[] args) {  
        throw new NullPointerException("Text Here");  
    }  
}
```

- push



The screenshot shows a Java IDE window titled "Question7 [Java Application]". The console output displays a runtime exception: `Exception in thread "main" java.lang.NullPointerException: Text Here at S1.Question7.main(Question7.java:6)`. The exception message "Text Here" is highlighted in blue, matching the string passed to the `throw` statement in the code above.

# מחסנית Stack

```
class Stack<T> {
```

```
// Remove the TOP element
```

```
// Returns the first element in the stack */
```

```
public T pop() {
```

```
    if(arr.isEmpty())
```

```
        throw new NoSuchElementException("No elements present in Stack");
```

```
    else
```

```
        return arr.get(--size);
```

```
}
```

```
...
```

```
}
```

- push
- pop



# מחסנית Stack

```
class Stack<T> {
```

```
    /* Returns the TOP element in the stack */
```

```
    public T top() { return arr.get(size - 1); }
```

- push
- pop
- top

...

}

# מחסנית Stack

```
class Stack<T> {
```

```
    /* Returns true iff the stack is empty */
```

```
    public boolean isEmpty() { return arr.isEmpty(); }
```

- push
- pop
- top
- isEmpty

...

}

# מחסנית Stack

```
class Stack<T> {
```

```
    /* clear the stack */
```

```
    public void clear() { arr.clear(); size = 0; }
```

- push
- pop
- top
- isEmpty
- clear

```
    ...
```

```
}
```

# מחסנית Stack

```
class Stack<T> {
```

```
    /* returns true iff the element is in the stack */  
    public boolean search(T element) { return arr.contains(element); }
```

- push
- pop
- top
- isEmpty
- clear
- search

...

}

# מחסנית Stack

```
class Stack<T> {
```

```
    /* Returns the size of the stack */  
    public int size() { return size; }
```

- push
- pop
- top
- isEmpty
- clear
- search
- size

...

}

# מחסנית Stack

## Testing

```
public static void main(String[] args) {  
    MyStack<Integer> s = new MyStack<Integer>(5);  
    s.push(1);  
    s.push(2);  
    s.push(3);  
    s.push(4);  
    s.push(5);  
    System.out.println(s.size()); // 5  
    System.out.println(s.top()); // 5  
    System.out.println(s.pop()); // 5  
    System.out.println(s.top()); // 4  
    System.out.println(s.isEmpty()); // False  
}
```

## Sort a Stack ?

# מחסנית Stack



Sort a Stack ?

# Stack מחסנית

## Inverse Insertion Sort Complexity

$O(n)$  start with reverse sorted stack

$O(n^2)$  start with sorted stack



```
import java.util.Stack;
```

```
    public static void main(String[] args) {  
        Stack<Integer> st = new Stack<Integer>();  
        st.add(1);  
        st.add(2);  
        st.add(3);  
        //          Go Out ->  
        //          3  
        //          2  
        //          1  
        // Insert  
        System.out.println(st.isEmpty()); // false  
        System.out.println(st.size()); // 3  
        System.out.println(st.peek()); // 3  
        System.out.println(st.pop()); // 3  
        System.out.println(st.peek()); // 2  
    }
```

## תרגיל (Stack)

כתוב פונקציה סטטית שמקבלת מחרוזת

אשר מכילה רק את התווים "{,},[,],(,)"

ומחזירה true אם המחרוזת תקינה מתמטית.

דוגמה: מחרוזת חוקית: `[]{}{}()`

מחרוזת לא חוקית: `(){}()`

```
// Function to check if given expression is balanced or not
private static boolean Question5(String s) {
    // take a empty stack of characters
    Stack<Character> st = new Stack<Character>();
    // traverse the input expression
    for(int i=0, n = s.length(); i<n ; i++) {
        char curr = s.charAt(i);
        // if current char in the expression is a opening brace, push it to the stack
        if(curr == '(' || curr == '{' || curr == '[' ) {
            st.push(curr);
        }
        else { // Its } or ) or ]
            // return false if mismatch is found
            if(st.isEmpty())
                return false;
            // pop character from the stack
            char top = st.pop();
            if( top == '(' && curr != ')' ||
                top == '[' && curr != ']' ||
                top == '{' && curr != '}' )
                return false;
        }
    }
    // expression is balanced only if stack is empty at this point
    return st.isEmpty();
}
```

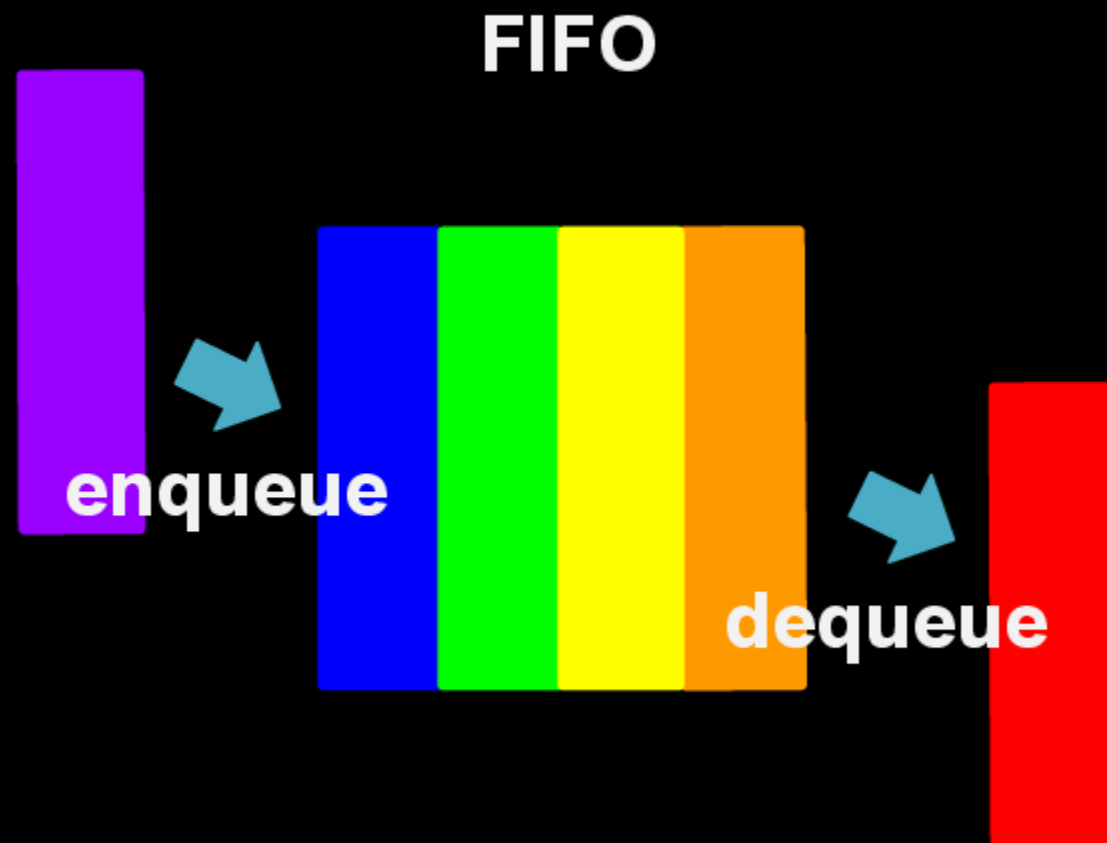
פעולות השוואה במקרה הגרוע על-מנת לבצעם.  
פעולות השוואה במקרה הגרוע על-מנת לבצעם.  
פעולות השוואה במקרה הגרוע על-מנת לבצעם.

כל אלגוריתמי המיון המבוססים על פעולת השוואה דורשים לפחות  
כל אלגוריתמי המיון המבוססים על פעולת השוואה דורשים לפחות  
כל אלגוריתמי המיון המבוססים על פעולת השוואה דורשים לפחות

# תור Queue



# תור Queue



# תור Queue

```
class Queue<T> {
```

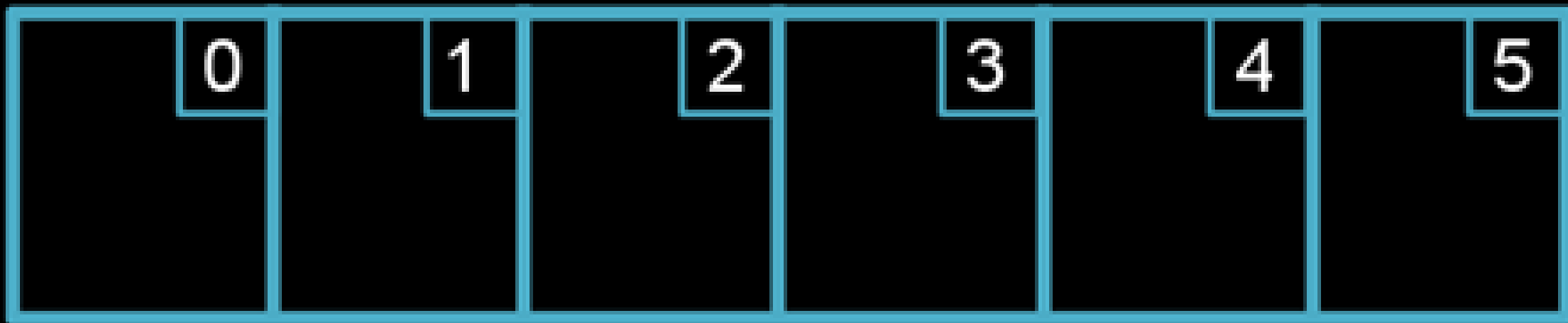
```
    ...
```

```
}
```

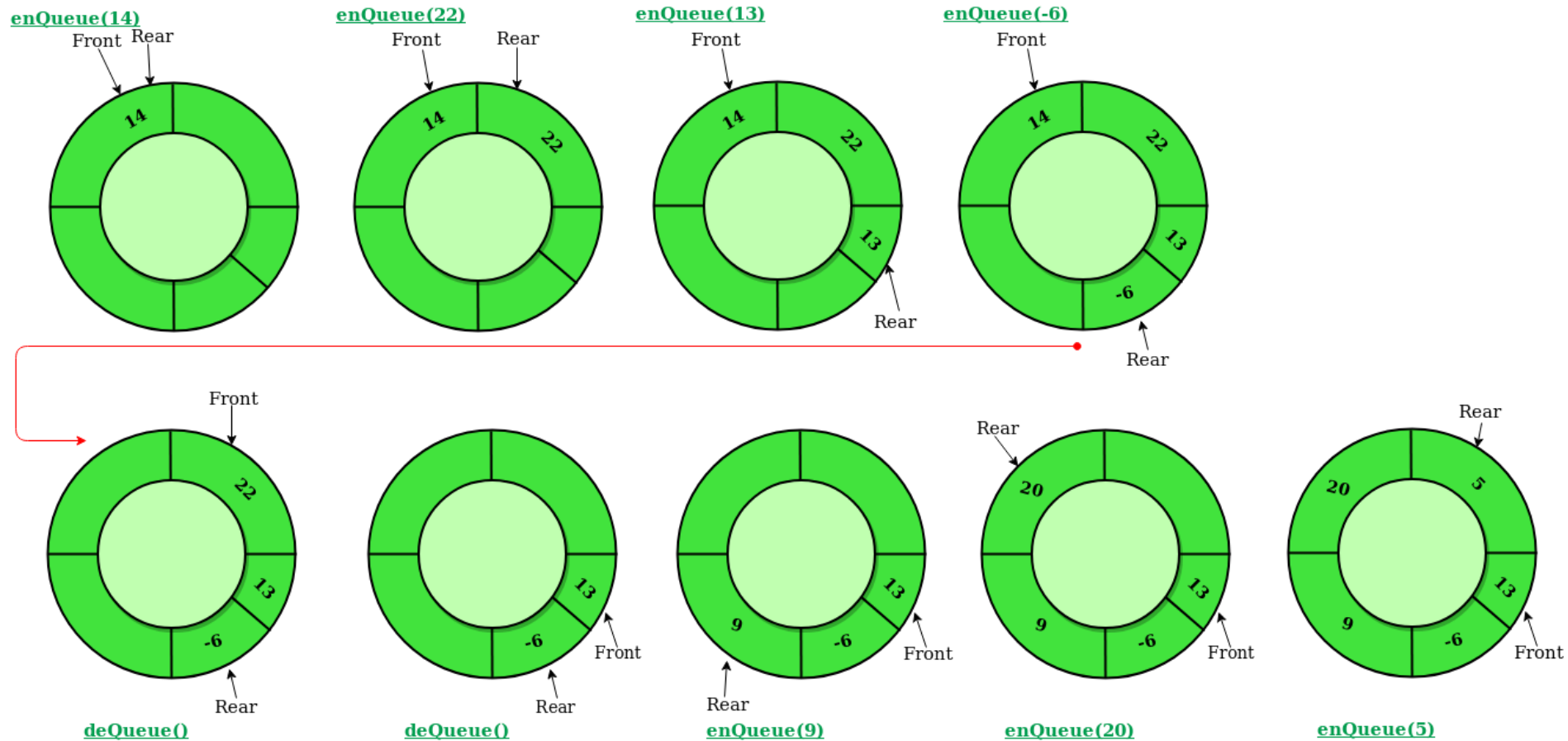
פעולות השוואה במקרה הגרוע על-מנת לבצעם.  
פעולות השוואה במקרה הגרוע על-מנת לבצעם.  
פעולות השוואה במקרה הגרוע על-מנת לבצעם.

כל אלגוריתמי המיון המבוססים על פעולת השוואה דורשים לפחות  
כל אלגוריתמי המיון המבוססים על פעולת השוואה דורשים לפחות  
כל אלגוריתמי המיון המבוססים על פעולת השוואה דורשים לפחות

# תור Queue



# תור Queue



# תור Queue

```
class Queue<T> {  
    private T[] arr; // array to store queue elements  
    private int size; // size of the queue  
    private int head; // front points to front element in the queue  
  
    ...  
}
```



# תור Queue

```
class Queue<T> {  
    // Constructor to initialize queue  
    public Queue(int size) {  
        arr = (T[]) new Object[size];  
        size = 0;  
        head = 0;  
    }  
    ...  
}
```

# תור Queue

חשוב!

- enqueue

```
class Queue<T> {  
  
    // Utility function to add an item to the queue  
    public void enqueue(T element) {  
        ensureCapacity();  
        arr[(head + size) % arr.length] = element;  
        size++;  
    }  
  
    // ensures that data has the capacity to hold additional elements  
    // data is reallocated in case it doesn't  
    private void ensureCapacity() {  
        if (size >= arr.length){  
            T[] newData = (T[]) new Object[arr.length*2 + 1];  
            for(int i=0; i<arr.length; i++)  
                newData[i] = arr[i];  
            arr = (T[]) newData;  
        }  
    }  
    ...  
}
```

# תור Queue

```
class Queue<T> {  
  
    // Utility function to remove front element from the queue  
    public T dequeue() {  
        if(isEmpty())  
            throw new NoSuchElementException("No elements present in Queue");  
        T ans = (T)arr[head];  
        size--;  
        head = (head + 1) % arr.length; // Next Position  
        return ans;  
    }  
    ...  
}
```

- enqueue
- dequeue

# תור Queue

```
class Queue<T> {  
    /* Returns the FIRST element in the queue */  
    public T peek() {  
        if(isEmpty())  
            throw new NoSuchElementException("No elements present in Queue");  
        return arr[head];  
    }  
    ...  
}
```

- enqueue
- dequeue
- Peek

# תור Queue

```
class Queue<T> {  
  
    /* Returns true iff the queue is empty */  
    public boolean isEmpty() {  
        return size == 0;  
    }  
  
    . . .  
}
```

- enqueue
- dequeue
- Peek
- isEmpty

# תור Queue

```
class Queue<T> {  
  
    /* clear the queue */  
    public void clear() {  
        size = 0;  
        arr = (T[]) new Object[arr.length];  
        head = 0;  
    }  
  
    ...  
}
```

- enqueue
- dequeue
- peek
- isEmpty
- clear

# תור Queue

```
class Queue<T> {  
  
    /* returns true iff the element is in the queue */  
    public boolean contains(T element) {  
        for(int i=0; i<arr.length; i++)  
            if(arr[i] == element)  
                return true;  
        return false;  
    }  
  
    ...  
}
```

- enqueue
- dequeue
- peek
- isEmpty
- clear
- contains

# תור Queue

```
class Queue<T> {
```

```
    /* Returns the size of the queue */  
    public int size() { return size; }
```

```
    ...
```

```
}
```

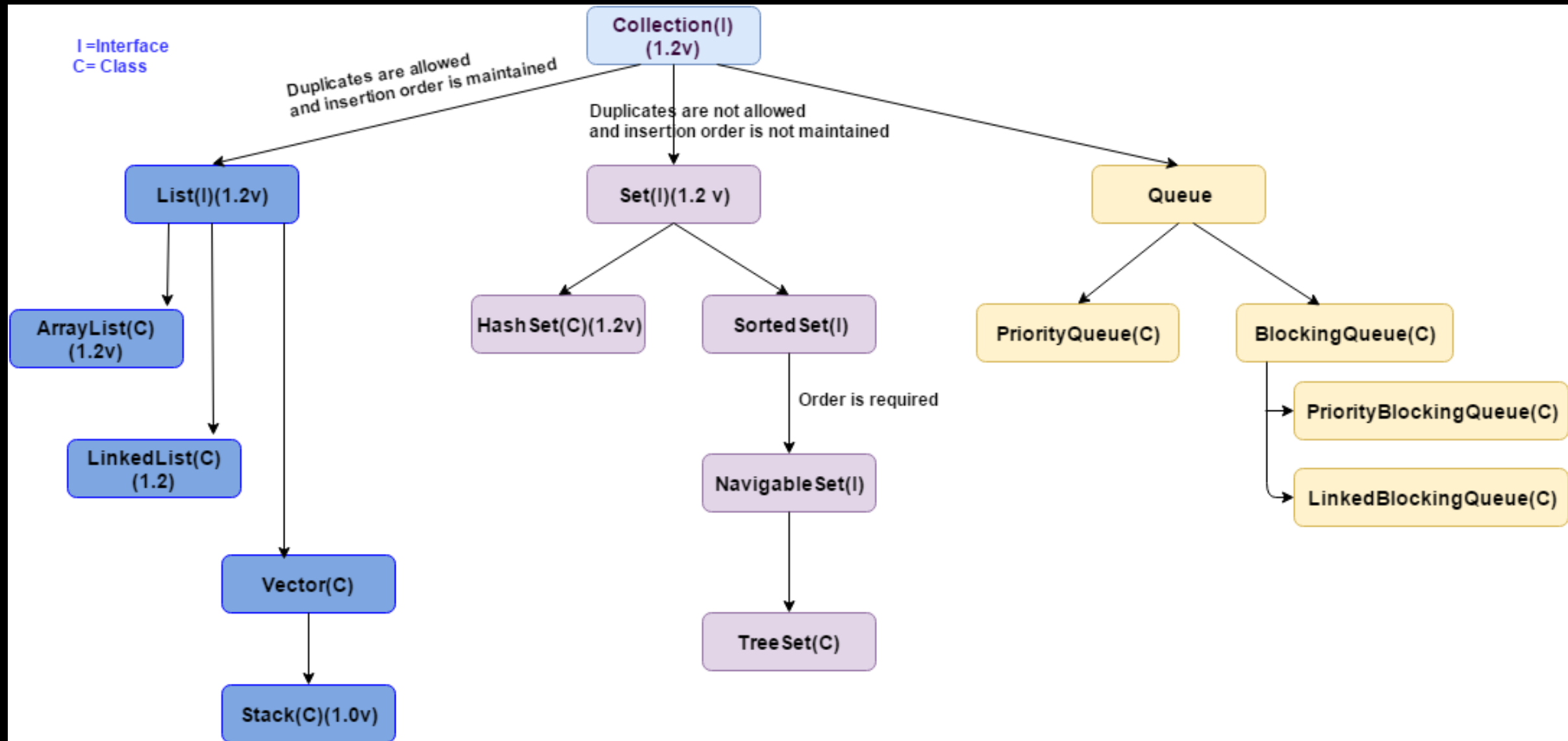
- enqueue
- dequeue
- peek
- isEmpty
- clear
- contains
- size



# תור Queue

```
MyQueue<Integer> q = new MyQueue<Integer>(3);
q.enqueue(1);
q.enqueue(2);
q.enqueue(3);
System.out.println(q.size()); // 3
q.enqueue(4);
q.enqueue(5);
System.out.println(q.size()); // 5
System.out.println(q.isEmpty()); // False
System.out.println(q.dequeue()); // 1
System.out.println(q.dequeue()); // 2
```

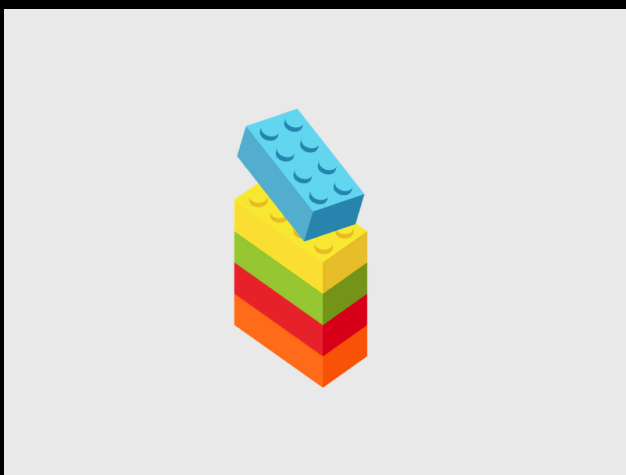
I=Interface  
C= Class



```
import java.util.concurrent.ArrayBlockingQueue;

public static void main(String[] args) {
    ArrayBlockingQueue<Integer> qe = new ArrayBlockingQueue<Integer>(3);
    qe.offer(1);
    System.out.println(qe.offer(2)); // True
    qe.offer(3);
    //          GO OUT ->
    //          1
    //          2
    //          3
    // Insert
    while(!qe.isEmpty())
        System.out.println(qe.remove());
    // 1
    // 2
    // 3
}
```

---



## Queue

Time complexity in big O notation

Algorithm	Average	Worst case
Space	$O(n)$	$O(n)$
Search	$O(n)$	$O(n)$
Insert	$O(1)$	$O(1)$
Delete	$O(1)$	$O(1)$

## Stack

Time complexity in big O notation

Algorithm	Average	Worst case
Space	$O(n)$	$O(n)$
Search	$O(n)$	$O(n)$
Insert	$O(1)$	$O(1)$
Delete	$O(1)$	$O(1)$

# עבודה עצמית



# Special credit to CS50, HarvardX

<https://study.cs50.net/>

Queue

Stack