DATASTRUCTURE & ALGORITHMS

**TOPICS TO BE COVERED**

***DATASTRUCTURES:***

1. Linked Lists
2. Doubly Linked Lists
3. Stacks & Queues
4. Binary Search Trees
5. Hash Tables
6. Graphs

***ALGORITHMS:***

1. Bubble Sort
2. Selection Sort
3. Insertion Sort
4. Merge Sort
5. Quick Sort
6. Tree Traversal

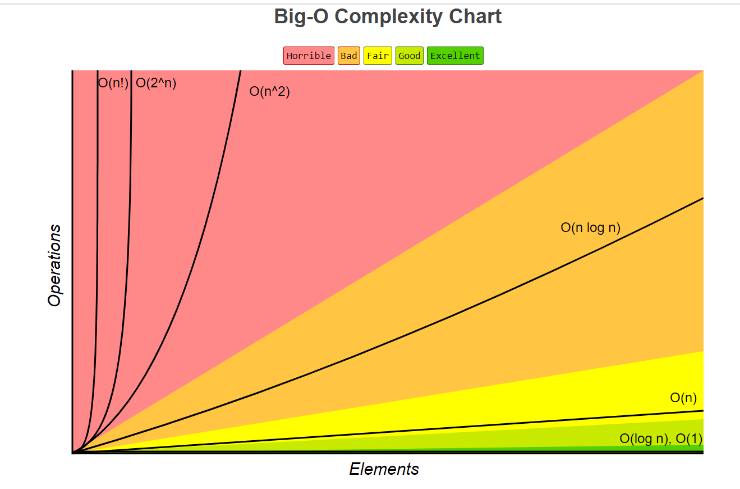
* Breadth First Search
* Depth First Search - Pre Order
* Depth First Search - Post Order
* Depth First Search - In Order

***COURSE TAKEN FROM UDEMY:***

[Java Data Structures & Algorithms + LEETCODE Exercises (udemy.com)](https://dxc.udemy.com/course/data-structures-and-algorithms-java/learn/lecture/27815396#overview)

**BIG O NOTATION**

* Time Complexity – Calculated Based on how long (time) it takes to execute the code
* Space Complexity – Calculated Based on how much space is consumed when code is executed

***CATERGORIES OF BIG – O:***

* *LINEAR TIME – O(n)*
* *LINEAR TIME- DROP CONSTANTS.*
* *QUADRATIC TIME – O(n^2)*
* *DROP NON-DOMINANTS*
* *CONSTANT TIME - O (1)*
* *LOGARITHMIC TIME -O (log n)*
* *QUASILINEAR TIME – O (n log n)*
* *DIFFERENT TERM OF INPUTS*

*For O(n) & O(n^2)*

***LINEAR TIME – O(n)***

*Examples:*

* *Looping through an element in an array*
* *Searching through a Linked List*

package com.big.o;  
  
public class LinearTime {  
//O(n) - LinearTime Example  
 public static void printItems(int n){  
 for(int i =0; i< n ;i++){  
 System.*out*.println(i);  
 }  
 }  
 public static void main(String[] args){  
 *printItems*(10);  
 }  
}

***LINEAR TIME- DROP CONSTANTS***

*O(Xn), where X may be 1,2,3…etc.*

*n + n = 2n , where we drop constants, and it becomes O(n)*

package com.big.o;  
  
public class LinearTime\_dropConstants {  
//O(2n) or O(3n)... O(Xn) = Drop Constants and its O(n) - Linear Time Example  
 public static void printItems(int n){  
 for(int i =0; i< n ;i++){  
 System.*out*.println(i);  
 }  
 for(int j =0; j< n ;j++){  
 System.*out*.println(j);  
 }  
 }  
 public static void main(String[] args){  
  
 *printItems*(10);  
 }  
}

***QUADRATIC TIME – O(n^2)***

***n \* n = n2 🡺 O(n2)***

*EXAMPLES :*

* *Insertion Sort*
* *Selection Sort*
* *Bubble Sort*
* package com.big.o;  
    
  public class QuadraticTime {  
    
   //O(n Exponent of 2) - n \* n = n exponent 2  
    
   public static void printItems(int n){  
   for(int i =0; i< n ;i++){  
   for(int j =0; j< n ;j++){  
   System.*out*.println(i + " " +j);  
   }  
   }  
    
   }  
   public static void main(String[] args){  
    
   *printItems*(10);  
   }  
  }

***DROP NON-DOMINANTS***

***O(n2+ n) = O(n2) ,where + n is dropped***

*If n =100, (n2) = 1000 and n = 100*

*Here* ***(n2) is Dominant*** *and* ***n is Non – Dominant***

package com.big.o;  
  
public class QuadraticTime\_NonDominant\_Drop {  
 /\*O(n2+ n ) = O(n2) ,where + n is dropped  
 If n =100, (n2) = 1000 and n = 100  
 Here (n2) is Dominant and n is Non – Dominant\*/  
 public static void printItems(int n){  
 for(int i =0; i< n ;i++){  
 for(int j =0; j< n ;j++){  
 System.*out*.println(i + " " +j);  
 }  
 }  
 for( int k = 0; k < n ; k++){  
 System.*out*.println(k);  
 }  
 }  
 public static void main(String[] args){  
 *printItems*(10);  
 }  
}

***CONSTANT TIME – O(1)***

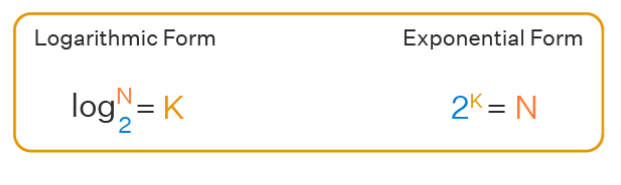
*n + n 🡺 No of Operations is only one, so O(1)*

*As n grows, No of Operations always stays constant*

*Examples:*

* *Random Access of an element in an array*
* *Insertion at the beginning of the LinkedList*
* package com.big.o;  
    
  public class ConstantTime {  
   /\*  
   n + n ⎝ No of Operations is only one, so O(1)  
   As n grows, No of Operations always stays constant\*/  
    
    
   public static void printItems(int n){  
   System.*out*.println(n+n);  
   }  
   public static void main(String[] args){  
    
   *printItems*(10);  
   }  
  }

***LOGARITHMIC TIME – O(log n)***



*If N = 8,*

*Log2N = K 🡺 Log28 = 3, i.e. 23=8*

*EXAMPLES:*

* *Binary Search*

*Here I want to find number 1 from an array.*

8

7

6

5

1

2

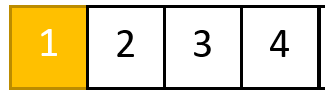
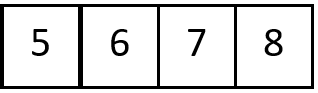
4

3

**OMIT IF TARGET NOT FOUND**

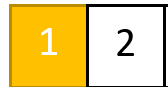
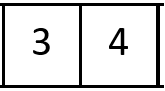
Split the Array into Two Halves equally

**Step = 1**

**OMIT IF TARGET NOT FOUND**

**Step = 2**

**Step = 3**

**OMIT IF TARGET NOT FOUND**

***QUASILINEAR TIME – O(n log n)***

*EXAMPLES*

* *Quick Sort*
* *Merge Sort*
* *Heap Sort*

***DIFFERENT TERM OF INPUTS***

* *Linear Time – O(n)*
* *Quadratic Time - O(n^2)*

*Linear Time – O(n) 🡺 if Two Inputs a, b then O (a +b)*

package com.big.o;  
  
public class DiffTermInputs1 {  
 public static void printItems(int a, int b){  
 for(int i =0; i< a ;i++){  
 System.*out*.println(i);  
 }  
 for(int j =0; j< b ;j++){  
 System.*out*.println(j);  
 }  
 }  
 public static void main(String[] args){  
  
 *printItems*(5,4);  
 }  
}

*Quadratic Time - O(n^2)🡺 if Two Inputs a, b then O (a \* b)*

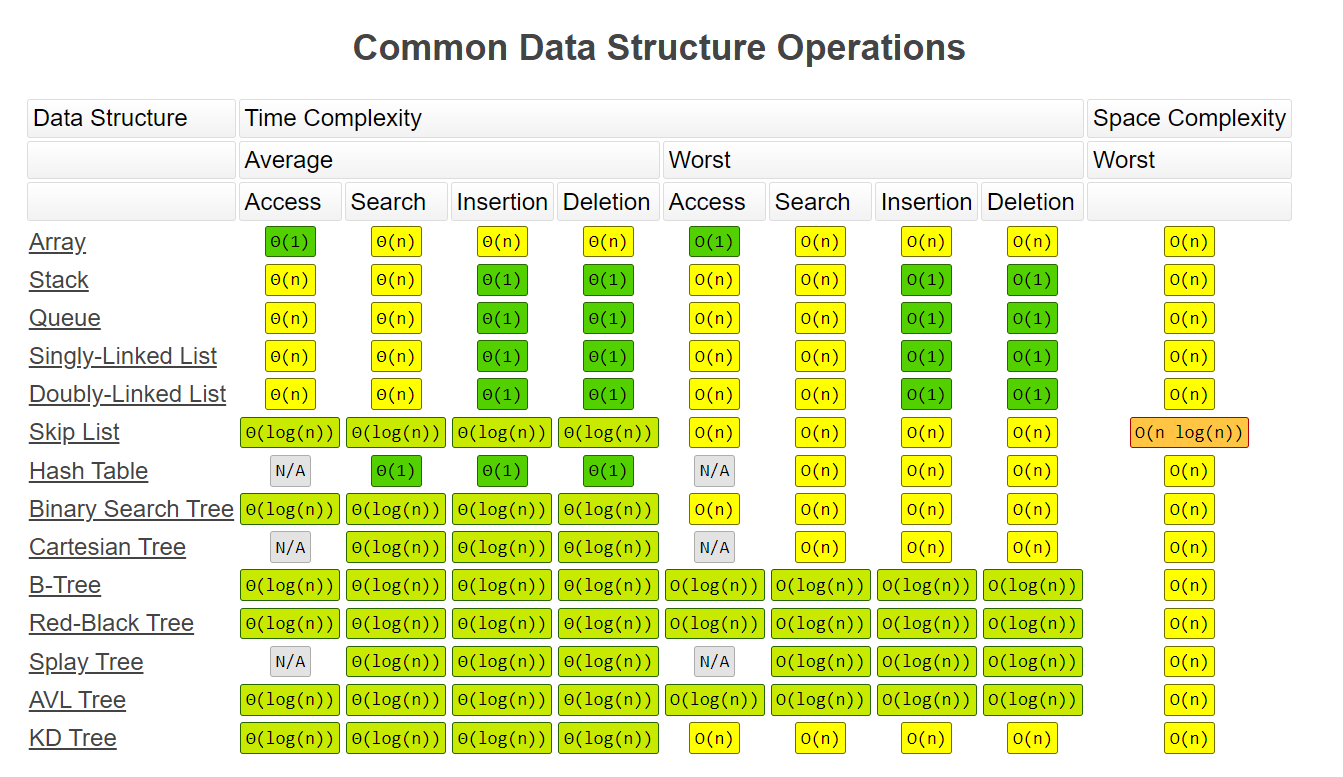
package com.big.o;  
  
public class DiffTermInputs2 {  
 public static void printItems(int a, int b){  
 for(int i =0; i< a ;i++){  
 for(int j =0; j< b ;j++){  
 System.*out*.println(i + " " +j);  
 }  
 }  
 }  
 public static void main(String[] args){  
  
 *printItems*(5,4);  
 }  
}

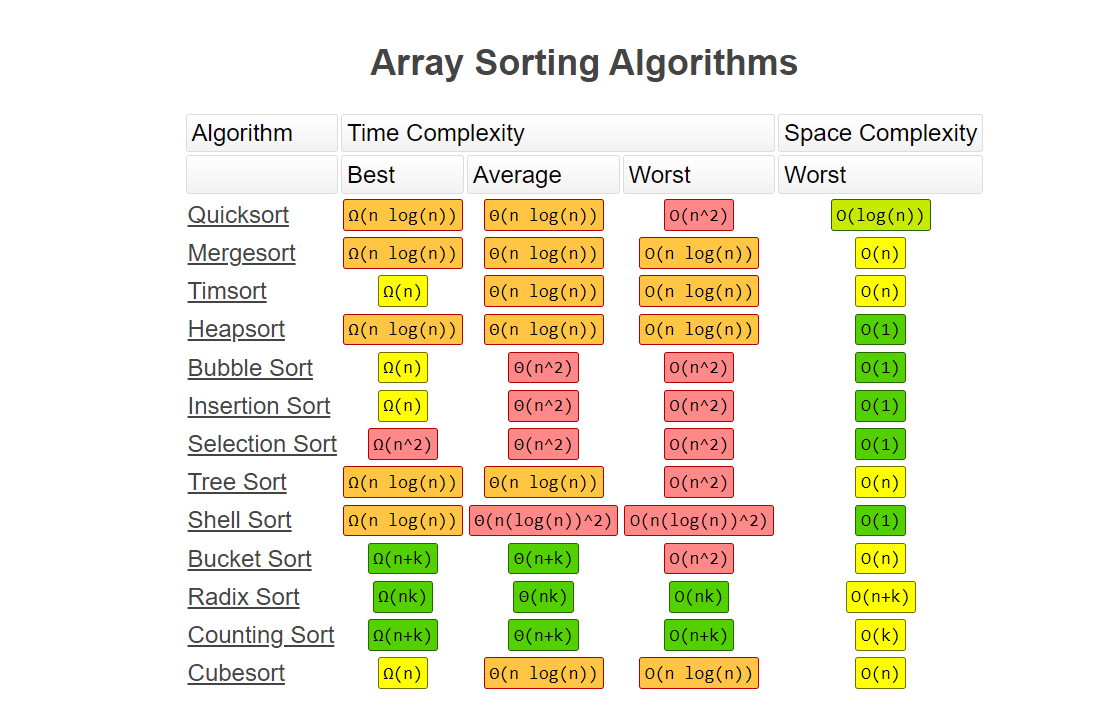
***BIG O ARRAYLIST***

* *If Adding or Removing an Element at Last Index of An array, then Time Complexity is O(1)*
* *If Adding or Removing an Element at First or Middle Index of an array, then Time Complexity is O(n)*
* *If fetching the value from an array using that specific value, then Time Complexity is O(n)*
* *If fetching the value from an array using that specific index, then Time Complexity is O(1)*

package com.big.o;  
  
import java.util.ArrayList;  
import java.util.Arrays;  
import java.util.List;  
  
public class BigOarrayList {  
 public static void main(String[] args){  
 List<Integer> myList = new ArrayList<Integer>(Arrays.*asList*(3,5,7,9,11));  
 myList.add(13);  
 System.*out*.println("Element Added " +myList);  
 myList.remove(5);  
 System.*out*.println("Element Removed " +myList);  
 myList.add(1,0);  
 System.*out*.println("Element Removed " +myList);  
 myList.remove(1);  
 System.*out*.println("Element Removed " +myList);  
 }  
}

***BIG O CHEAT SHEET***





***CLASSES :***

package com.classes;  
  
public class Cookie {  
 private String color;  
  
 public Cookie(String color) {  
 this.color = color;  
 }  
 public String getColor() {  
 return color;  
 }  
 public void setColor(String color) {  
 this.color = color;  
 }  
  
}

package com.classes;  
  
// Press Shift twice to open the Search Everywhere dialog and type `show whitespaces`,  
// then press Enter. You can now see whitespace characters in your code.  
public class Main {  
 public static void main(String[] args) {  
  
 Cookie cookieOne = new Cookie("Yellow");  
 Cookie cookieTwo = new Cookie("Red");  
 cookieOne.setColor("Pink");  
 System.*out*.println(cookieOne.getColor());  
 System.*out*.println(cookieTwo.getColor());  
 }  
}