Time Complexities

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Access** | **Search** | **Insertion** | **Deletion** |
| **Array** | O (1) | O (n) | O(n) | O(n) |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

**Arrays Methods**

1. Print Array as String – Arrays.toString(arr);
2. Create an ArrayList from Array

Integer [] arr = new int[7];

List<Integer> lst = new ArrayList<>(Arrays.asList(arr));

1. Convert Arraylist to Array

List<Integer> lst = new ArrayList<>();

Integer[] arr = new Integer[lst.size()];

lst.toArray(arr);

1. Convert Array to a Set

String[] arr = new String[];

Set<String> s = new HashSet<>(Arrays.asList(arr));

1. Reverse elements of an Array

String[] arr = new String[];

ArrayUtils.reverse(arr);

1. Iterating through array

For(int item:nums){

System.out.println(item);

}

For(int i=0;i<nums.length;i++){

System.out.println(nums[i]);

}

1. Concatenate 2 arrays

Int[] arr1 = new int[7];

Int[] arr2= new int[8];

Int[] arr3 = ArrayUtils.addAll(arr1,arr2);

1. Java.util.Arrays methods

Arrays.sort(object[] a) 🡪 Sorts Array in ascending order

Arrays.fill(arr,-1) 🡪 Fills all elements of array with specific value

Arrays.binarySearch(int[] arr,int key) 🡪 returns index of the element If its present else returns –(insertion point +1)

Arrays.equals(int[] arr1,int[] arr2) 🡪 Returns true if both arr1 and arr2 have same elements and same length

1. Arrays Comparator , comparing 2D Arrays

**import** java.util.Arrays;

**import** java.util.Comparator;

Int[][] arr = new int[][]{};

Arrays.sort(arr,new Comparator<int[]>(){

public int compare(int[] a, int[] b){

return a[0] – b[0];

}

});

Return value :

<0 indicates accending order

0 indicates same value

>0 indicates descending value

**Primitive Types**

|  |  |  |
| --- | --- | --- |
| **Data Type** | **Size** | **Description** |
| int | 4 bytes | Stores value between -2^32 to +2^31-1 |
| char | 2 bytes | Stores single character ASCII Character/Letter  0 to 65535(Unsigned)  A – Z is 65 to 90  a – z is 97 to 122 |
| byte | 1 byte | -128 to 127 |
| short | 2 bytes | -2^16 to +2^16-1(-32768 to + 32767) |
| long | 8 bytes | Stores whole numbers -2^64 to +2^64-1 |
| float | 4 bytes | Stores fractional numbers . Decimals sufficient for storing 6 to 7 decimal digits |
| double | 8 bytes | Stores fractional numbers. Decimals sufficient for storing upto 15 digits |
| boolean | 1 byte | Stores true or false values |

Declaring primate types

char ch =’a’;

int n = 1;

long lg = 1500000000L;

float myNum = 1.234f

double myNum = 19.999d

**Conversion Table**

|  |  |  |
| --- | --- | --- |
| **From** | **To** |  |
| Integer | String | Integer.valueOf(str) |
| Integer | Character | Character.forDigit(n);  //Different behavior for int to char  Int n = 1;char ch = (char)(n+’0’); //prints 1  Int n = 1;char ch = (char)n; //it will store ASCII character of given number which is start of heading which is not printable  Int n =’1’;char ch = (char)n; //prints 1 |
| Character | Integer | Character.getNumericValue(ch)  Char ch = ‘6’;  int n = ch – ‘0’; |
| Character | String | String.valueOf(ch);  Character.toString(ch); |
| String | Integer | Integer.valueOf(str)  Integer.parseInt(str); |
| String | Character | s.charAt(i);  str.toCharArray() |
| Display Integer in Binary Format |  | Integer.toBinaryString(n) |

**Important in-built Methods**

|  |  |
| --- | --- |
| **Function** | **Description** |
| Math.sqrt(dbl) | Sqrt(x)   * public static double sqrt(double a)   Returns the correctly rounded positive square root of a double value. Special cases:   * + If the argument is NaN or less than zero, then the result is NaN.   + If the argument is positive infinity, then the result is positive infinity.   + If the argument is positive zero or negative zero, then the result is the same as the argument.   Otherwise, the result is the double value closest to the true mathematical square root of the argument value. |
| Math.pow(x,y) | X^y   * public static double pow(double a,double b)   Returns the value of the first argument raised to the power of the second argument. Special cases:   * + If the second argument is positive or negative zero, then the result is 1.0.   + If the second argument is 1.0, then the result is the same as the first argument.   + If the second argument is NaN, then the result is NaN. |
| Math.floor(dbl) | * public static double floor(double a)   Returns the largest (closest to positive infinity) double value that is less than or equal to the argument and is equal to a mathematical integer. |
| Math.ceil(dbl) | * public static double ceil(double a)   Returns the smallest (closest to negative infinity) double value that is greater than or equal to the argument and is equal to a mathematical integer. |
| Random | Import java.util.Random  Random rand = new Random();  rand.nextInt() 🡪 returns a number between 0 and 2^32 inclusive  rand.nextInt(16) 🡪 returns a number between 0(included) and 16 (excluded) |
| Math.sin(x) | sinxsin⁡x |
| Math.cos(x) | cosxcos⁡x |
| Math.tan(x) | tanxtan⁡x |
| Math.asin(x) | sin−1x |
| Math.acos(x) | cos−1x |
| Math.atan(x) | tan−1x |
| Math.exp(x) | ex |
| Math.log(x) | lnx |
| Math.log10(x) | log10x |
| Math.round(x) | the closest integer to x, as athe closest integer to x, as a long |
| Math.abs(x) | |x| |
| Math.max(x,y) | The maximum of the two values |
| Math.min(x,y) | The minimum of the two values |
| Math.PI | Double  3.14159265358979323846 |
| Math.E | Double  2.7182818284590452354 |

**Bit Manipulation**

|  |  |
| --- | --- |
| **Operation to Check** | **How to Check** |
| Check whether lowest significant bit is 1 or 0 | X & 1 |
| Reset Lowest bit to 0 | X & (X -1) // drops lowest set bit of x  Example – 7 & 6 makes  111 & 110 🡪 110 which causes lowest bit to 0 |
| Retain Only Lowest bit and reset all other bits | X & ~(X -1) |
|  |  |