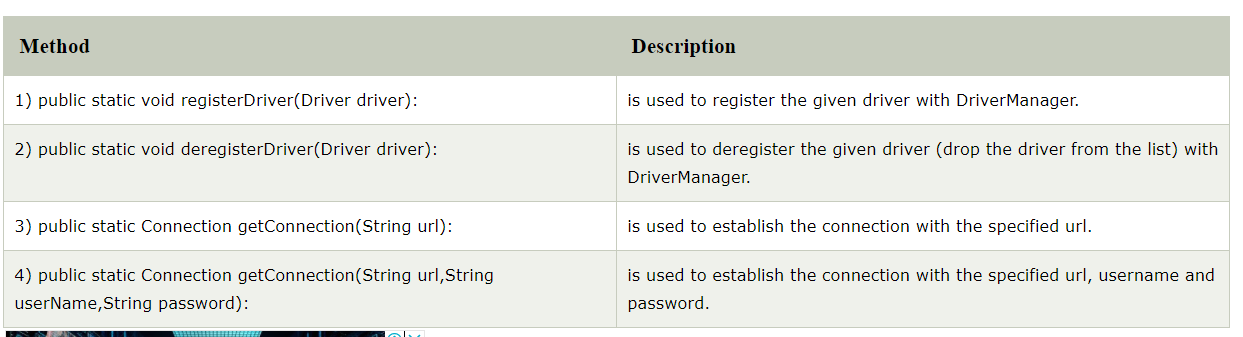
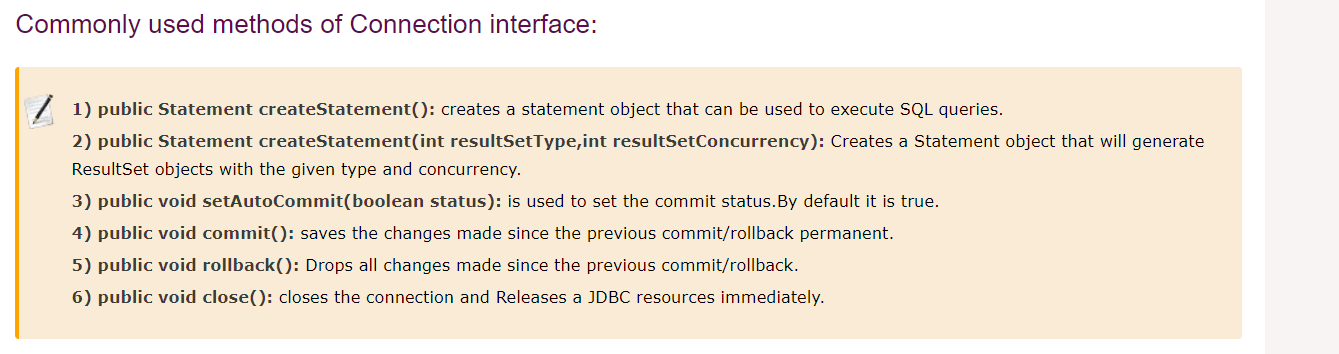
1. **JDBC – V VIMP Code –**

**Ans** : We’ll study few Interfaces and Classes before proceeding with JDBC Code :

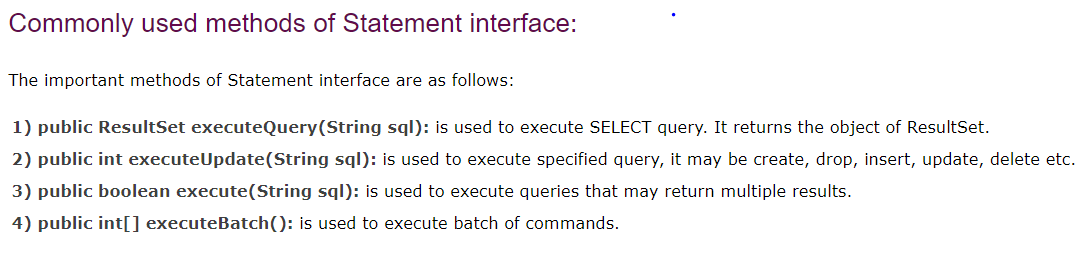
**DriverManager** – It acts as a interface b/w Users and drivers. It keeps track of the drivers that are available and handles establishing a connection between a database and the appropriate driver.

It has 4 methods : 

**Connection Interface** - A Connection is the session between java application and database. The Connection interface provide many methods for transaction management like commit(), rollback() etc.



**Statement** **interface**  - The Statement interface provides methods to execute queries with the database. It provides factory method to get the object of ResultSet.



When suppose we want to use executeUpdate method, then we can write like :

**int** result=stmt.executeUpdate("delete from emp765 where id=33");

System.out.println(result+" records affected");

**ResultSet Interface :** The object of ResultSet maintains a cursor pointing to a row of a table. Initially, cursor points to before the first row. But we can make this object to move forward and backward direction by passing either TYPE\_SCROLL\_INSENSITIVE or TYPE\_SCROLL\_SENSITIVE in createStatement(int,int) method as well as we can make this object as updatable.

Complete JDBC Code:

static String url="jdbc:postgresql://finance-ninja.c4w0mdqode2x.us-east-1.rds.amazonaws.com/financeNinja\_QA";

**static** String *userName*="qa\_write";

**static** String *passWord*="finan3ceQAWrite";

**static** Connection *connection*;

**static** Statement *statement*;

**public** **static** Connection test() **throws** SQLException {

// DriverManager will establish the connection b/w specified url, Username and pwd and its return type is Connection obj reference

*connection*= DriverManager.*getConnection*(*url*, *userName*, *passWord*);

// Now we have to build the connection b/w java app and DB by Connection obj ref and its return type is Statement obj

*statement* = *connection*.createStatement();

// Now we have to execute the SQL Query in DB after making the connection

ResultSet rs=*statement*.executeQuery("");

// Then we have ResultSet object, we can get the results from it,

1. Java Stream Concept –

/\* productCategories.entrySet().stream()

// ...

.forEach(e ->

{

//Map<String, List<String>> productSubCategories = new HashMap<String, List<String>>();

System.out.println(e.getKey() + ":" + e.getValue());

try {

productSubCategories.put(e.getKey(), getSubCategories(e.getValue()));

TempCategory.clear();

} catch (SQLException e1) {

e1.printStackTrace();

}

});

System.out.println(productSubCategories);\*/

1. AutoBoxing in Java
2. Upcasting and Downcasting in Java
3. Regular Expressions in Java
4. Generics, Enum
5. Getter- Setter method in Java
6. **Identifiers in Java –**

Ans : The rule is In Java, an identifier can be a class name, method name, variable name, or label.

Allowed characters in identifier are all alpha numeric ,, only $ and \_ sign are allowed. Identifier shud n’t start with numbers.

Reserved words can’t be used as identifiers. There are 53 Reserved words in Java.

//int 123Test = 55;

//int est@ = 213;

//int float = 12; // float is reserved keyword

All above mentioned identifers are Invalid.

1. **String Methods : V IMP**

Strings are basically defined as an array of characters.

String s1 = "Hello";

s1.concat(" Bye"); // Since String is immutable, it won't change here normally.

System.***out***.println(s1); // Output - Hello

s1 = s1.concat(" How"); // If we have to change String value, we have to give like this.

System.***out***.println(s1); // Output - Hello How. Also a new object ref will assign to s1

String s2 = **new** String("boy");

System.***out***.println(s2); // Output - boy

s2 =s2.concat(" Girl"); // We can concat to that String created by using new operator as well

System.***out***.println(s2); // Output - boy Girl

String s3 = "";

System.***out***.println(s3.isEmpty()); // Output – true // It will check if String is empty

String s4 = " Hello How are How you ";

System.***out***.println(s4); // Output - Hello How are How you

System.***out***.println(s4.trim()); // Output - Hello How are How you // It will trim white spaces from both side

System.***out***.println("TEST Data NEED TO BE IN LOwER".toLowerCase()); // Output - // Lowe case

System.***out***.println("Test data need to be in Upper case".toUpperCase()); // Output - // Upper case

System.***out***.println(s1.charAt(4)); // Output - o // It will give the character at the given location from the String

System.***out***.println(s1.contains(s2)); // Output - false // If the whole String "s2" is part of String s1, then it will return true else false

System.***out***.println(s4.contains(s1)); // Output - true

System.***out***.println(s1.contains("e")); // Output - true

System.***out***.println(s1.substring(2)); // Output - llo How // If we give index like 2 here, it will cut the character at 0 and 1st position from the String & gives output from 2nd position onwards

System.***out***.println(s1.substring(2, 5)); // Output - llo // Will give String b/w these indexes(2,3,4)

System.***out***.println(s1.contentEquals("Hello How")); // Output - true

String s5 = **new** String("Hello");

String s6 = "Hello";

System.***out***.println(s5.contentEquals(s6)); // Output - true // content equals method will just match the content, doesn't matter with object reference variable

System.***out***.println(s5.endsWith("llo")); // Output - true // it search for the word, complete word or just some letter

System.***out***.println(s6.startsWith("Hel")); // Output - true

System.***out***.println(s5.indexOf("e")); // Output - 1 // It gives the location of the given character

System.***out***.println(s4.indexOf("How")); // Output - 8 // Gives the location of whole String

System.***out***.println(s4.indexOf("How", 5)); // Output - 8 // It will give answer as 8, if we see String s4. After 5th position, "How" comes at 8th position

System.***out***.println(s4.indexOf("How", 10)); // Output - 17 // It will give answer as 17, if we see String s4. After 10th position, next "How" comes at 17th position

System.***out***.println(s5.lastIndexOf("o")); // Output – 4

String s2 = **new** String("boy");

String s3 = "Boy";

String s4 = "Boy";

System.***out***.println(s3==s4); // True

System.***out***.println(s3.contentEquals(s4)); // true

System.***out***.println(s3.equals(s4)); // true

System.***out***.println(s2==s4); // false - Check for object reference, if same then true else false

System.***out***.println(s2.contentEquals(s4)); // false - just check for case sensitive contents of String

System.***out***.println(s2.equals(s4)); // check for contents of 2 Strings

System.***out***.println(s2.equalsIgnoreCase(s3));

// Below method is to convert String into an character array

String data = "Welcome to Google";

**char**[] ch = data.toCharArray();

System.***out***.println(ch); // now we can print data from character based on index

System.***out***.println(ch[3]);

1. **Split Method in String –**

We’ll see some examples where we are using split method in String and doing split based on 2-3 signs.

String s1 = "We're, Ridiculously.Advance Committed! Welcome, to Eduerka.Hello";

String[] str = s1.split("[,.]+"); // How it will work here, firstly we have to give square braces and we have to write signs inside those square braces only. here we mentioned 2 things ",." Now it will split the whole string based on this 2 values. How ? It will start watching these 2 signs in it from start till end. Wherever they found these 2 signs, it will split the String

System.***out***.println(str.length);

**for**(**int** i=0;i<str.length;i++) {

System.***out***.println(str[i]);

}

// Output will be :

5

We're

Ridiculously

Advance Committed! Welcome

to Eduerka

Hello

String s2 = "We're, Ridiculously.Advance Committed! Welcome, to Eduerka.Hello";

String[] str1 = s2.split("[,.!]"); // Here we have included 1 more sign

System.***out***.println(str1.length);

**for**(**int** i=0;i<str1.length;i++) {

System.***out***.println(str1[i]);

}

// Output will be :

6

We're

Ridiculously

Advance Committed

Welcome

to Eduerka

Hello

1. **Difference between StringBuffer and StringBuilder in Java –**

Ans : Since we know String is immutable in nature and whenever String manipulations are performed, automatically a new String would be generated by discarding the old one. This thing will create a garbage in heap memory. To Avoid this garbage, concept of StringBuffer and StringBuilder came in Java.

StringBuffer and StringBuilder are classes used for String manipulation. These are mutable objects, which provide methods such as substring(), insert(), append(), delete() for String manipulation.

|  |  |
| --- | --- |
| **StringBuffer** | **StringBuilder** |
| StringBuffer operations are thread-safe and synchronized | StringBuilder operations are not thread-safe are not-synchronized. |
| StringBuffer is to used when multiple threads are working on the same String | StringBuilder is used in a single-threaded environment. |
| StringBuffer performance is slower when compared to StringBuilder | StringBuilder performance is faster when compared to StringBuffer |
| **Syntax:** StringBuffer var = new StringBuffer(str); | **Syntax:** StringBuilder var = new StringBuilder(str); |

Now, we’ll see the difference b/w String, StringBuffer and StringBuilder based on 2 parameters :

1. Mutability – String is immutable and other 2 are not. It means that, if we try to add or concat any other String to String object, it will create a new object in String Pool/Heap memory.

While if we append or manipulate object of StringBuffer and StringBuilder, then it won’t generate any new object. It will update the value in the same reference object only.

We’ll see this with the help of Code :

String Code :

String s1 = "Abc";

System.***out***.println(s1.hashCode()); // 65602

s1 = s1.concat("New");

System.***out***.println(s1); // AbcNew

System.***out***.println(s1.hashCode()); // 1954427390 (It got changed)

s1 = s1 + "Test";

System.***out***.println(s1);

System.***out***.println(s1.hashCode()); // 970933328 (Again it got changed)

StringBuffer Code :

StringBuffer s2 = **new** StringBuffer("Hello");

System.***out***.println(s2.hashCode()); // 366712642

s2.append(" Hi");

System.***out***.println(s2); // Hello Hi

System.***out***.println(s2.hashCode()); // 366712642 (After string manipulation, still its same)

StringBuilder Code :

StringBuilder s3 = **new** StringBuilder("Welcome");

System.***out***.println(s3.hashCode()); // 1829164700

s3.append(" Hi");

System.***out***.println(s3);

System.***out***.println(s3.hashCode()); // 1829164700 (NO change)

1. Performance : StringBuilder is faster than StringBuffer as it offers no synchronization. This is because no extra overhead needs to be added to the system and also does not slows down the processing.

Thread – String is not used in a threaded env since its immutable. StringBuffer will be used in multi-thread env while String Builder is used in single-threaded env.

1. **IMP Question – What is Java String Pool ? Why String Class is made Immutable or Final in Java ? How various String objects get stored in memory.**

Ans :

We can create String objects using a new Operator as well as providing values in double quotes like below :

String s1 = “Cat”;

String s2 = “Cat”;

String s3 = “Dog”

String s4 = new String(“Cat”)

So, now when we create String s4 using above technique, it will create an reference object in Heap Memory. While for other 3 strings which we are creating by just giving doubles quotes, they will be created in “String Pool”.

“String Pool in Java” is a pool of Strings stored in Java Heap Memory. String Pool is an area in JVM used to avoid redundant generation of String Objects, means objects reference can be reused. Now we’ll see how.

In above example, when String s1 got created, it will create a reference obj in String Pool. Now when we try to create another String s2, it will first check the value assigned to s2. And if value is same, then the same reference object which got assigned to s1 will get assigned to s2 as well. But if value is different, then a new reference object will be assigned. Like it will happen for String s3. In this way, we are saving the waste of memory allotment for each String objects or we can save lot of space for Java Runtime.

Also, we can see 1 more thing here :

S1==s2 >> True

S1==s4 >> False (Since object ref are diff)

S1.equals(s4) >> True

S1.equals(s2) >> True

Note : String Pool is possible because String is immutable in Java & its implementation of String interning concept.

The reasons why String is Immutable are below :

1. String Pool – As we know String literals are created in String Pool and the need of creating objects in String Pool is to avoid waste of memory allotment for each String objects separately. So basically if 2 String are having the same value, then same reference object will be shared between these 2 Strings. Now if String is mutable, then it won’t possible to do that. Due to this reason, String are immutable. By making String immutable, this sharing of String literal was possible
2. Security - String has been widely used as parameter for many Java classes, e.g. for opening network connection, you can pass host and port as String, for reading files in Java you can pass path of files and directory as String and for opening database connection, you can pass database URL as String. If String was not immutable, a User might change the Path to something else and this can cause serious security issues.
3. Optimization and Performance –
4. Multi thread – Since we have multi-thread concept in Java, and String is immutable , so when we use String for Multi threading it gives a much cleaner code involving sharing of String between multiple threads without worrying about external synchronization.

**Pros and Cons of String being Immutable & Final :**

Pros – Since String is immutable, its mostly used as key in hash based collections eg. HashMap and Hashtable. Its not required to use Immutable objects in Hashmap but its preferable over mutable objects, because if state of mutable object is changed during its stay inside HashMap, it would be impossible to retrieve it back. Another advantage is thread safety.

Cons - Since String is immutable, it generates lots of temporary use and throw object, which creates pressure for Garbage collector. Now storing String literals in String pool is the solution for this prob. But storing Strings in pool has a hidden risk associated with it. String pool is located in PermGen Space of Java Heap, which is very limited as compared to Java Heap. Having too many String literals will quickly fill this space, resulting in java.lang.OutOfMemoryError: PermGen Space. But from Java7 onwards, this space has shifted from String Pool to normal heap space.

There is another disadvantage of making String final, as it limits its extensibility. Now, you just can not extend String to provide more functionality.

1. **Difference between Mutable and Immutable objects :**

**public** **class** Modifier {

**private** **int** x;

**int** y;

Modifier(**int** x,**int** y){

**this**.x = x;

**this**.y = y;

}

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

**final** Modifier m = **new** Modifier(5,10);

System.***out***.println(m.x);

System.***out***.println(m.hashCode());

m.x = 59; // Since x variable is defined as private only. So if its public or private we can change its value.Also Protected keyword too.

System.***out***.println(m.x);

System.***out***.println(m.hashCode());

}

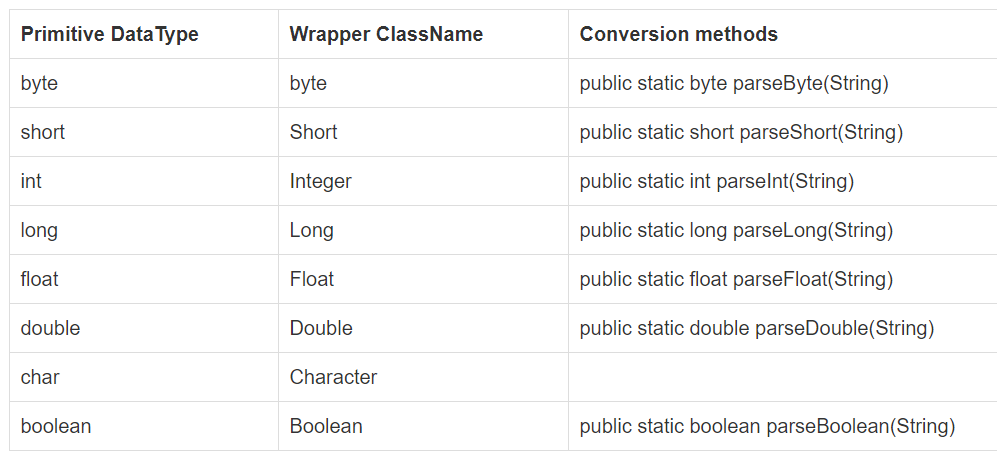
Now, if we see above, we have made the x as private. If we made x as public or protected as well. Then also x value can be changed, it means its object ref will not change. Only if we assign the keyword as final, then value won’t change. We’ll get compilation error on m.x = 59 ;row itself.

Immutable are String objects, we have seen its behavior in other questions.

1. **Java Wrapper Classes –**

Ans : In Java, Wrapper Class is used for converting primitive data type into object and object into a primitive data type. For each primitive data type, a pre-defined class is present which is known as Wrapper class.

Why we Need it - As we knew that in Java when input is given by the user, it is in the form of String. To convert a string into different data types, Wrapper classes are used.



In Java, all the primitive wrapper classes are immutable. When a new object is created the old object is not modified.

Long Class- The Long class is a wrapper class that is used to wrap a value of the primitive type long in an object. In addition, this class provides several methods for converting a long to a String and vice versa.

Method 1 : Converting Long Value to String ( String that represent Long’s value only) - *toString()*

**long** a = 25;

System.***out***.println(Long.*toString*(a));

Method 2 – Converting String value to Long – *parseLong()*

String b = "54";

**long** c = Long.*parseLong*(b);

System.***out***.println(c);

Method 3 : To convert any long value into Long instance - *valueOf()*

**long** a = 25;

Long x = **new** Long(a); // This is 1 way of creating a Long instance using long value

Long y = Long.*valueOf*(a); // this is the another way to get the Long instance from long primitive type

System.***out***.println(x);

System.***out***.println(y);

We can use same above methods with other Wrapper class objects –

**JAVA PROGRAMS – DATA STRUCTURES**

1. **Java Program – How to Print duplicate characters from String :**

Logic behind is We have to iterate through the String, create 1 Hashmap with Character and Integer as its params. Then If the key value is already present in the Map, increase the counter by 1.

Code:

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

String input = "Data Niene Tahree";

String a = input.replace(" ", "").toLowerCase();

HashMap<Character, Integer> charCountMap = **new** HashMap<Character, Integer>();

**for**(**int** i=0;i<a.length();i++) {

**char** ch = a.charAt(i);

**if**(charCountMap.containsKey(ch)) {

charCountMap.put(ch, charCountMap.get(ch)+1);

}

**else** charCountMap.put(ch, 1);

}

**for** (Character c : charCountMap.keySet()) {

**if** (charCountMap.get(c) > 1)

System.***out***.println("duplicate character : " + c + " >>" + " count : " + charCountMap.get(c));

}

Output :

duplicate character : a >> count : 3

duplicate character : t >> count : 2

duplicate character : e >> count : 4

duplicate character : n >> count : 2

So, in above we are just printing those key-value pair whose key have value more than 1.

Now if suppose requirement came to made the HashMap with only those keys whose value is more than 1.

Code & Logic – We have to iterate the Map, we need to note 1 thing, we can’t remove entry or add any new entry from HashMap directly while iterating from HashMap. We can do that with Iterator while iterating value. If we do it directly from HashMap we’ll get exception : java.util.concurrentmodificationexception

Below code will create this Exception, if we try to iterate Map and remove key-value pair directly from it while iterating it:

**for**(Map.Entry<Character, Integer> m : charCountMap.entrySet()) {

**if**(m.getValue()==1) {

System.***out***.println(m.getKey());

charCountMap.remove(m.getKey());

}

**else** **continue**;

}

**NOTE : Exception is “Concurrent Modification”. The**[**ConcurrentModificationException**](http://www.java67.com/2015/10/how-to-solve-concurrentmodificationexception-in-java-arraylist.html)**comes when you use Map.remove(Object key) to remove a key.**

To overcome this issue, we have to use iterator and we can remove it from there, Code will be :

Iterator<Integer> itr = charCountMap.values().iterator();

**while**(itr.hasNext()) {

**if**(itr.next()==1) {

itr.remove();

}

}

Java ConcurrentHashMap class is part of the Concurrency Collection Classes. It’s a hash table implementation, which supports concurrent retrieval and updates. It’s used in a multi-threaded environment to avoid ConcurrentModificationException.

If we try to modify the collection while iterating over it, we get ConcurrentModificationException. Java 1.5 introduced Concurrent classes in the java.util.concurrent package to overcome this scenario. ConcurrentHashMap is the Map implementation that allows us to modify the Map while iteration. The ConcurrentHashMap operations are thread-safe. ConcurrentHashMap doesn’t allow null for keys and values.

Code :

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

Map<String,String> myMap = **new** ConcurrentHashMap<String, String>();

myMap.put("1", "1");

myMap.put("2", "1");

myMap.put("3", "1");

myMap.put("4", "1");

myMap.put("5", "1");

myMap.put("6", "1");

Iterator<String> it = myMap.keySet().iterator();

**while**(it.hasNext()) {

String key = it.next();

**if**(key.equals("3")){

myMap.put(key+"New", "Data");

}

**else** **continue**;

}

System.***out***.println(myMap);

Output :

{1=1, 2=1, 3=1, 4=1, 5=1, 3New=Data, 6=1}

1. **Java Program - To reverse a String :**

Method 1 : By using charAt(index) method :

String input = "Test Data";

String revString = "";

**for**(**int** i=input.length()-1;i>=0;i--) {

revString = revString + input.charAt(i);

System.***out***.print(input.charAt(i)); // 1st way to just print it here with giving sop(print) only

}

System.***out***.println();

System.***out***.println(revString); // Other way to print reverse String

Method 2: Using StringBuffer or StringBuilder –

String input = "Test Data";

StringBuffer s = **new** StringBuffer(input);

s.reverse();

System.***out***.println(s);

StringBuilder sb = **new** StringBuilder(input);

System.***out***.println(sb.reverse());

Method 3: Using Recursion –

**public** **class** Modifier {

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

String input = "Welcome to Mumbai";

Modifier m = **new** Modifier();

System.***out***.println(m.rev(input));

}

**public** String rev(String str) {

**if**(str.length()==0) {

**return** " ";

}**else** **if**(str.length()==1) {

**return** str;

}

**else** **return** str.charAt(str.length()-1) + rev(str.substring(0,str.length()-1));

}

}

1. **Java Program - Entering input from keyboard**

Scenario 1 : Enter input from keyboard only once :

**public** **static** **void** main(String[] args) **throws** IOException {

InputStreamReader io = **new** InputStreamReader(System.***in***);

BufferedReader b = new BufferedReader(io);

System.out.println("Enter input");

String input = b.readLine();

System.out.println("input Value is:" + input);

}

Output will be:

Enter input

211Test

After that Execution will end here.

Scenario 2 : Enter input one by one till User enter some input which will stop the execution or further entry of data after that.

**public** **static** **void** main(String[] args) **throws** IOException {

InputStreamReader io = **new** InputStreamReader(System.***in***);

BufferedReader b = **new** BufferedReader(io);

String input = "";

**while**(!input.equals("stop")){

System.***out***.println("enter course:");

input = b.readLine();

System.***out***.println("input Value is:" + input);

}

}

User will enter input 2-3 times, and after that just enter “stop”. After that further entry will be stopped.

1. **Java Program – How to validate 1 String is reverse of another String :**

**Code**:

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

String a1 = "nine";

String a2 = "enin";

// 1st Method - Use StringBuffer

System.***out***.println(a1.equals(a2));

System.***out***.println(**new** StringBuffer(a2).reverse());

System.***out***.println(a1.equals(**new** StringBuffer(a2).reverse().toString()));

// 2nd method - Just reversing the 2nd String and use equals method on 2 String

String dummy = "";

**for**(**int** i=a2.length()-1;i>=0;i--) {

dummy = dummy + a2.charAt(i);

}

**if**(a1.equals(dummy)) {

System.***out***.println("Pass");

}

**else** System.***out***.println("False");

}

1. **Java Program – To check if 2 Strings are anagram**

**Code & Logic :**

**Method 1 : Using Arrays.sort() method :**

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

String s1= "Angel";

String s2 = "Angle";

**char** [] ch1 = s1.toLowerCase().toCharArray();

**char** [] ch2 = s2.toLowerCase().toCharArray();

Arrays.*sort*(ch1);

Arrays.*sort*(ch2);

System.out.println(Arrays.equals(ch1, ch2)); // output is true

Assert.*assertEquals*(ch1, ch2); //Pass

// Arrays.sort() method has Time Complexity of O(n logn) but below Hashmap technique will have complexity O(n)

}

**Method 2 : Using HashMap technique, just print all the characters in the HashMap and compare it**

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

String s1= "Angel";

String s2 = "AgLne";

HashMap<Character, Integer> h1 = **new** HashMap<Character, Integer>();

HashMap<Character, Integer> h2 = **new** HashMap<Character, Integer>();

h1 = *logic*(s1);

h2 = *logic*(s2);

System.***out***.println(h1.equals(h2));

}

**public** **static** HashMap<Character, Integer> logic(String input) {

String lowerCaseInput = input.toLowerCase();

HashMap<Character, Integer> dummy = **new** HashMap<Character, Integer>();

**for**(**int** i=0;i<lowerCaseInput.length();i++) {

**if**(dummy.containsKey(lowerCaseInput.charAt(i))){

dummy.put(lowerCaseInput.charAt(i), dummy.get(lowerCaseInput.charAt(i))+1);

}

**else** dummy.put(lowerCaseInput.charAt(i), 1);

}

**return** dummy;

}

// This program has Time complexity of O(n)

Overall, O(n) is better than O(n logn)

1. **Java Program – How to Sort an String**

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

String s1= "Angel";

System.***out***.println(*sortedLogic*(s1));

}

**public** **static** String sortedLogic(String input) {

**char** [] ch1 = input.toLowerCase().toCharArray();

Arrays.*sort*(ch1);

**return** **new** String(ch1);

}

1. **Java Program – How to validate that given String has all Unique characters :**

**Method 1 : Doing iteration inside string itself**

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

String s1 = "Welcom";

String lowercase = s1.toLowerCase().trim().replace(" ", "");

Boolean flag = **false**;

**for**(**int** i=0;i<lowercase.length();i++) {

**for**(**int** j=i+1;j<lowercase.length();j++) {

**if**(lowercase.charAt(i)==lowercase.charAt(j)) {

flag = **false**;

**break**;

}

**else** {

flag = **true**;

}

}

**if**(flag.equals(**false**)) {

System.***out***.println("Entered input doesn't have unique characters");

**break**;

}

}

**if**(flag.equals(**true**)) System.***out***.println("Entered input has unique characters");

}

**Method 2 : Using HashMap, put all the characters of String into HashMap with key as its character and value as its occurrences in the String .**

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

String s1= "Angelg";

HashMap<Character, Integer> h1 = **new** HashMap<Character, Integer>();

h1 = *logic*(s1);

System.***out***.println(h1);

**for**(Map.Entry<Character, Integer> m : h1.entrySet()) {

**if**(m.getValue()>1) {

System.***out***.println("Fail");

}

**else** **continue**;

}

}

**public** **static** HashMap<Character, Integer> logic(String input) {

String lowerCaseInput = input.toLowerCase();

HashMap<Character, Integer> dummy = **new** HashMap<Character, Integer>();

**for**(**int** i=0;i<lowerCaseInput.length();i++) {

**if**(dummy.containsKey(lowerCaseInput.charAt(i))){

dummy.put(lowerCaseInput.charAt(i), dummy.get(lowerCaseInput.charAt(i))+1);

}

**else** dummy.put(lowerCaseInput.charAt(i), 1);

}

**return** dummy;

}

1. **Java Program – To find first non repeated character in String –**

**Code & Logic –** Here suppose String is “Welcocm”. Here c is the first repeating character, so “l” is the first non- repeating character. We have to find “l” only.

As we saw in above previous program, we were finding that whether String has unique characters or not.

So we’ll use the same code again, only modification is we have to find the character where it was failing ie, where it stop the code execution. That character is the key for this program, means that character is the first repeating character.

Then, we’ll find the location of that char, then n-1 location character is our final output.

**Method 1 : Using Iteration in String itself :**

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

String s1 = "Welcocm";

**char** ch = 0;

String lowercase = s1.toLowerCase().trim().replace(" ", "");

Boolean flag = **false**;

**for**(**int** i=0;i<lowercase.length();i++) {

**for**(**int** j=i+1;j<lowercase.length();j++) {

**if**(lowercase.charAt(i)==lowercase.charAt(j)) {

ch = lowercase.charAt(i);

flag = **false**;

**break**;

}

**else** {

flag = **true**;

}

}

**if**(flag.equals(**false**)) {

System.***out***.println("Entered input doesn't have unique characters");

System.***out***.println(ch);

**break**;

}

}

**if**(flag.equals(**true**)) System.***out***.println("Entered input has unique characters");

**int** index = lowercase.indexOf(ch);

System.***out***.println("Expected Output : " + lowercase.charAt(index-1));

}

**Method 2 : Using HashMap :**

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

String s1= "Angelg";

LinkedHashMap<Character, Integer> h1 = **new** LinkedHashMap<Character, Integer>();

h1 = *logic*(s1);

System.***out***.println(h1);

**int** count = 0;

**for**(Map.Entry<Character, Integer> m : h1.entrySet()) {

count = count+1;

**if**(m.getValue()>1) {

System.***out***.println(m.getKey());

System.***out***.println("Fail");

**break**;

}

**else** **continue**;

}

System.***out***.println(count);

System.***out***.println(s1.toLowerCase().charAt(count-2));

}

**public** **static** LinkedHashMap<Character, Integer> logic(String input) {

String lowerCaseInput = input.toLowerCase();

LinkedHashMap<Character, Integer> dummy = **new** LinkedHashMap<Character, Integer>();

**for**(**int** i=0;i<lowerCaseInput.length();i++) {

**if**(dummy.containsKey(lowerCaseInput.charAt(i))){

dummy.put(lowerCaseInput.charAt(i), dummy.get(lowerCaseInput.charAt(i))+1);

}

**else** dummy.put(lowerCaseInput.charAt(i), 1);

}

**return** dummy; }

1. **Java Program – To find all the substrings of String in Java**

For Ex: If input is “abb”  then output should be “a”, “b”,”b”, “ab”, “bb”, “abb”.

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

String s1 = "Welc";

**for**(**int** i=0;i<s1.length();i++) {

**for**(**int** j=i+1;j<=s1.length();j++) {

System.***out***.println(s1.substring(i, j));

}

}

}

1. **V IMP Java Program – Related to Longest Substring length with unique characters inside a String**

Program is, we have 1 String with value “teshuitabcgteop” ..Now we have to find that longest substring in this String that have all the unique characters and also should be continuous.

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

String a = "teshuitabkolcgteop";

**int** tempSize = 0;

**int** permSIze = 0;

HashMap<Character, Integer> hs = **new** HashMap<Character, Integer>();

**for**(**int** i=0;i<a.length();i++) {

**if**(hs.containsKey(a.charAt(i))) {

tempSize = hs.size();

hs.clear();

**if**(tempSize>permSIze) {

permSIze = tempSize;

}

hs.put(a.charAt(i), 1);

}

**else** hs.put(a.charAt(i), 1);

}

System.***out***.println(permSIze);

}

// Output will be 8 --

1. **V IMP JAVA Program Related to Substring – How to find Longest common substring in an Array of Strings**

String[] arr = {"grace", "gracful" ,"disgracful", "gracfully"};

String permString = arr[0];

// Code 1 - Need to find the smallest string in the String array, it will help in reducing further no. of loops

**for**(**int** i=1;i<arr.length;i++) {

**if**(permString.length()>arr[i].length()) {

permString = arr[i];

}

}

System.***out***.println(permString);

// Code 2 - Now, we will run the loop within the above found smallest String and get all the substrings in it and then check in all

// individual elements of String array whether that substring is present or not. If its present for all the String array elements, then

// we have to play with max length of the substring

**int** count = 0;

**int** maxLength = 0;

**int** tempLength = 0;

String finalString = "";

String oldString = "";

**for**(**int** i=0;i<permString.length();i++) {

**for**(**int** j=i+1;j<=permString.length();j++) {

//System.out.println(permString.substring(i, j));

**for**(String a : arr) {

**if**(a.contains(permString.substring(i, j))) {

count+=1;

}

**else** **break**;

}

**if**(count==arr.length) {

tempLength = permString.substring(i, j).length();

oldString = permString.substring(i, j);

}

**if**(tempLength>maxLength) {

maxLength = tempLength;

finalString = oldString;

}

count = 0;

}

}

System.***out***.println(finalString);

// Output is “grac”

1. **VIMP Program – To find the longest substring of Vowels**

**Ans** : Code will be :

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

String a = "geeouoksforgeiaeks";

String tempString = "";

String permString = "";

**for**(**int** i=0;i<a.length();i++) {

**if**(*vowelCheck*(a.charAt(i))==**true**) {

tempString = tempString.concat(Character.*toString*(a.charAt(i)));

}

**else** {

**if**(tempString.length()>permString.length()) {

permString = tempString;

}

tempString = "";

}

}

System.***out***.println(permString);

}

**public** **static** **boolean** vowelCheck(**char** c) {

**return** (c == 'a' || c == 'e' || c == 'i'

|| c == 'o' || c == 'u');

}

// Output is eeouo

1. **JAVA Program – To find the Longest Pallindrome Substring**

**Ans** : We’ll see below code, but 1 very imp thing we should not, when we want to compare 1 String and 1 StringBuffer object content, then we can’t use “equals” method directly. We have to use “contentEquals()” method.

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

String a = "forgeeksskeegfor";

String tempString = "";

String permString = "";

**for**(**int** i=0;i<a.length();i++) {

**for**(**int** j=i+1;j<a.length();j++) {

String dummy = a.substring(i,j);

**if**(*palindromeCheck*(dummy)==**true**) {

tempString = dummy;

**if**(tempString.length()>permString.length()) {

permString = tempString;

}

}

}

}

System.***out***.println(permString);

}

**static** **boolean** palindromeCheck(String abc) {

StringBuffer sb = **new** StringBuffer(abc);

sb.reverse();

**if**(abc.contentEquals(sb)) {

**return** **true**;

}

**else** **return** **false**;

}

Output – geeksskeeg

Time Complexity - O(n\*n)

1. **JAVA Program – To check if the 2 Strings have a common substring –**

**Ans** : Code will be :

String s1 = "HELLO";

String s2 = "WORLD";

**boolean** flag = **false**;

System.***out***.println(flag);

**for**(**int** i=0;i<s1.length();i++) { // TILL 4

**for**(**int** j=i+1;j<=s1.length();j++) { // 1TO 5 means 0 1 2 3 4

//System.out.println(s1.substring(i,j)); // H HE

String subString1 = s1.substring(i,j);

**if**(s2.contains(subString1)) {

flag = **true**;

**break**;

}

**else** flag = **false**;

}

}

System.***out***.println(flag);

1. **Java Program – To check if a String contains only digits –**

**Method 1 : Using Ascii code , Ascii code of Integer vary from 48 to 57.**

String s1 = "1234567890";

**char**[] ch = s1.toCharArray();

Boolean flag = **false**;

**for**(**int** i=0;i<ch.length;i++) {

**int** ascii = ch[i];

**if**(ascii>=48 && ascii<=57) {

flag = **true**;

}

**else** {

flag = **false**;

**break**;

}

}

Assert.*assertTrue*(flag.equals(**true**));

**if**(flag==**false**) {

System.***out***.println("Not numeric input");

}

**else** System.***out***.println("Numeric Input");

**Method 2 : Using different-2 regular expressions :**

Simple One : ("^[0-9]+$")

String s1 = "1234567890";

**if**(s1.matches("^[0-9]+$")) {

System.***out***.println("Digit only");

}

**else** System.***out***.println("not numeric input");

//Assert.assertTrue(s1.matches("^[0-9]+"));

Another one : ("-?\\d+(\\.\\d+)?")

String s1 = "1234567890";

**if**(s1.matches("-?\\d+(\\.\\d+)?")) {

System.***out***.println("Numeric");

}**else** System.***out***.println("Not Numeric");

Another one:

String s2 = "123987";

Pattern pattern = Pattern.*compile*(".\*\\D.\*");

System.***out***.println(!pattern.matcher(s2).matches()); // Output – true

1. **Java Program – Pallindrome program**

Simple Logic – Just take a String and reverse it. After reverse it, save it in some different String and then compare the new string with the original one. That’s’ all.

Just see the ways of reversing the String using recursion, StringBuffer or just String iteration

1. **To find the entered number is Prime number or not ?**

Ans : Please check below code :

**public** **static** **void** main(String[] args) **throws** InterruptedException {

**int** a = 241;

**int** count = 0;

**for**(**int** i=2;i<a;i++) {

**if**(a%i==0) {

System.***out***.println("Not a prime number");

**break**;

}

**else** {

count = count +1;

}

}

**if** (count == a-2) {

System.***out***.println("Prime no");

}

**TIME COMPLEXITY AND SPACE COMPLEXITY OF ALGORITHMS :**

1. **How to analyze an Algorithm :**

Factors would be :

* Time – Algos are the procedures for solving the problem. How much time the program is taking to execute, so after analyzing what time we get is the time we get in the form of function. That’s called as time function.
* Space – Once the program execute on JVM, we must know how much memory it will consume.
* Data Transfer/ Network consumption - Every app is either internet based, web based or say cloud based. So data transfer and network consumption is also a imp criteria. How much data is going to transfer
* CPU Registers – How much CPU registers the program/ algo is consuming.

Example 1: Algo –

Algorithm swap(a,b)

{

Temp = a; -- 1 unit of time

A=b; -- 1 unit of time

B= temp; -- 1 unit of time

}

So finally time function/ Time complexity will be - f(n) = 3

It’s an constant value, 3. Each step is going to execute only once, so we will say 1 unit of time. Total is 3.

Space complexity – there are total 3 variable that need to be defined –

A, b, temp

Each variable has 1 word of space.

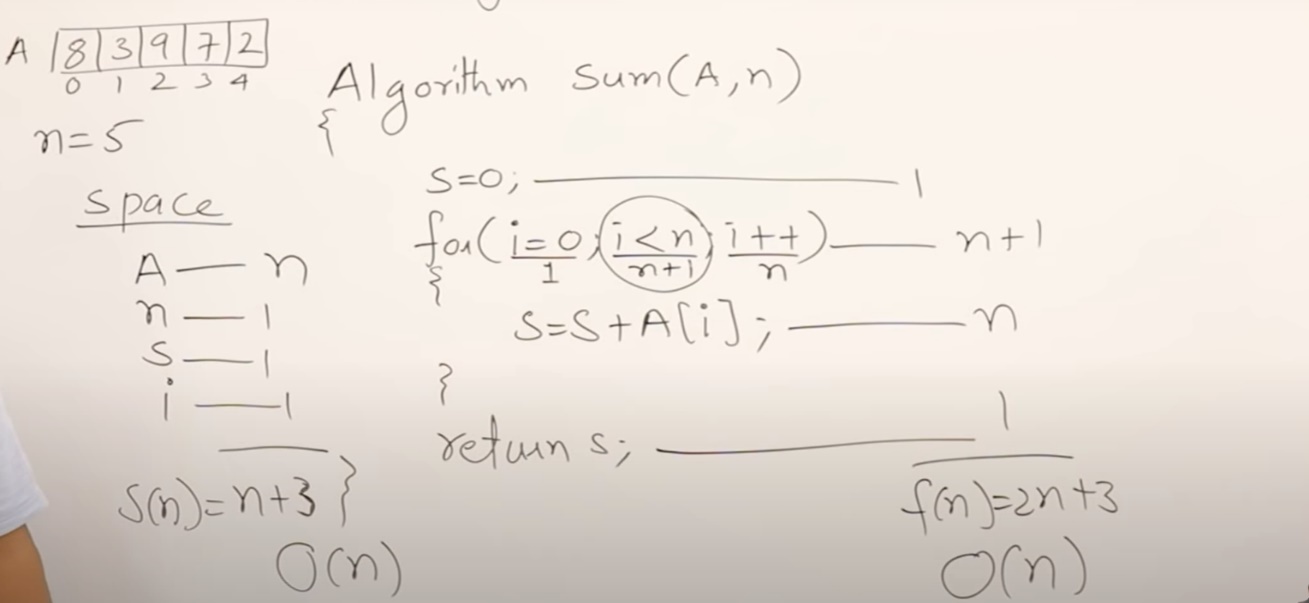
So space complexity is S(n) = 3

We say it as 3 words, y we say like that, since we don’t know in the program whether these variables are exactly going to define as int, String or any other data type.

So, 1 more point to note, we mostly represent it as Order basis – O(1)

O(1) repsrents constant 1. Whether the value of Time and space complexity is 3 or 100. If its an constant number, we will say it as O(1). Generally we define the complexity of the programs in Order base.

Example 2 : Frequency Count Method –



In above method, A is an array and n is the length of that array. The time taken by the algo known by assigning 1 unit of time for each statement.

Now, s=0 , that’s a simple statement – it will take 1 unit of time.

Next we have a loop, which will execute till i<n. It means if suppose n =5. Then it will execute 5 times. But actually it will execute (n+1) times.

See, 5 times it will run and pass. 6th time it will run again, check and fail the condition. It will not go inside for loop. But this statement, this condition will be run.

So time complexity will be (n+1)

Next statement inside for loop, will run n times. So time complexity will be n

Last statement will run 1 time. So time complexity will be 1

Now, total Time Complexity will be – 1 + (n+1) + n + 1 = 2n + 3

F(n) = 2n+3

Now what’s the degree of that polynomial. That’s just degree 1, its just n. So we say Order of n.

O(n)

Now, space complexity –

A is an array of size n. So its space complexity will be n

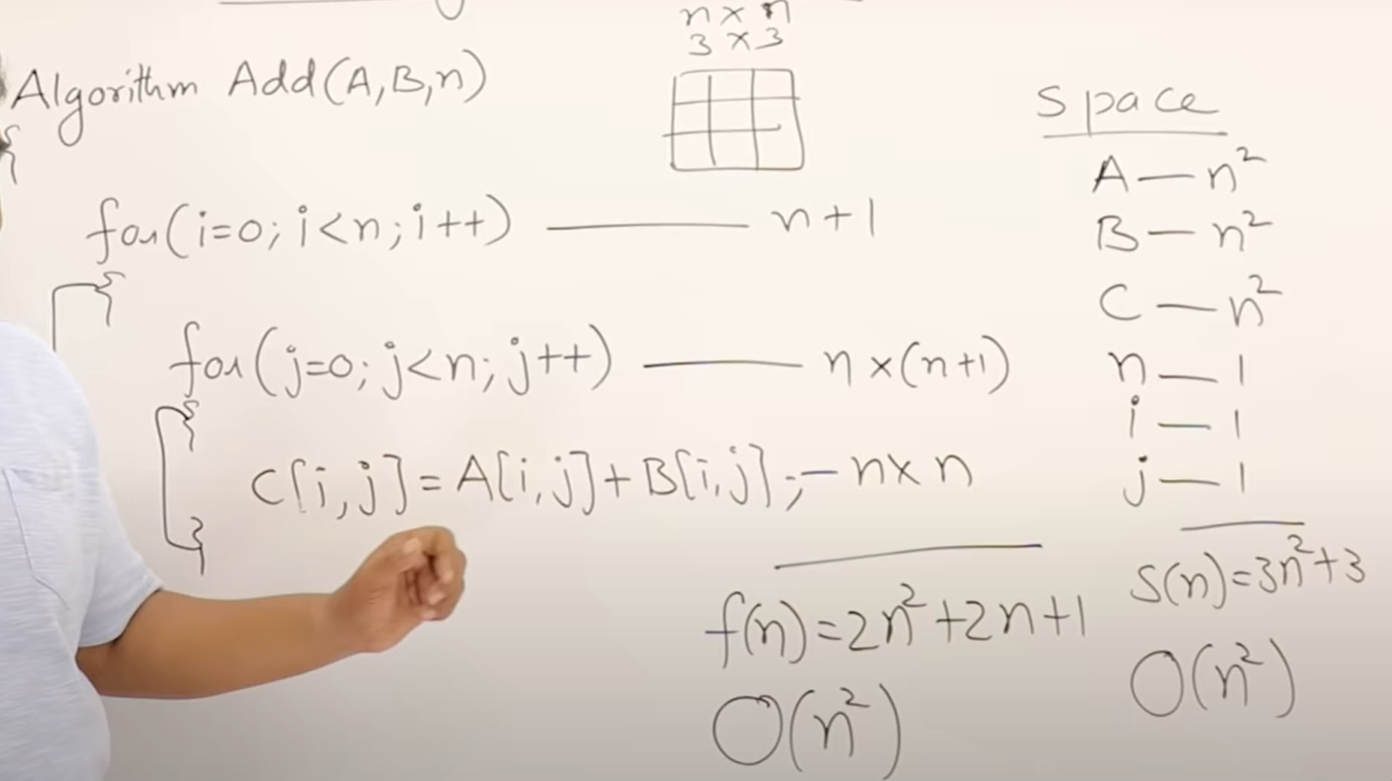
Other variables are – n, s and i. Assigning 1 unit to each of it.

So, total Space complexity will be – n + 3

S(n) = n+3

Degree of polynomial will be – O(n).

Example 3 –



Explanation – Outer loop will be – n+1

Inner loop and statement written inside it will get now - n complexity

Now, inner loop and statement written inside it will also get some its own complexity,

So, inner loop will get (n+1) also. Total complexity of inner loop will be n \* (n+1)

Statement written inside inner loop will have – n \* n

So, total time complexity will be f(n) = 2n^2 + 2n + 1

So, highest degree is 2. We ‘ll write it as Order of n^2.

O(n^2) – Order of n square.

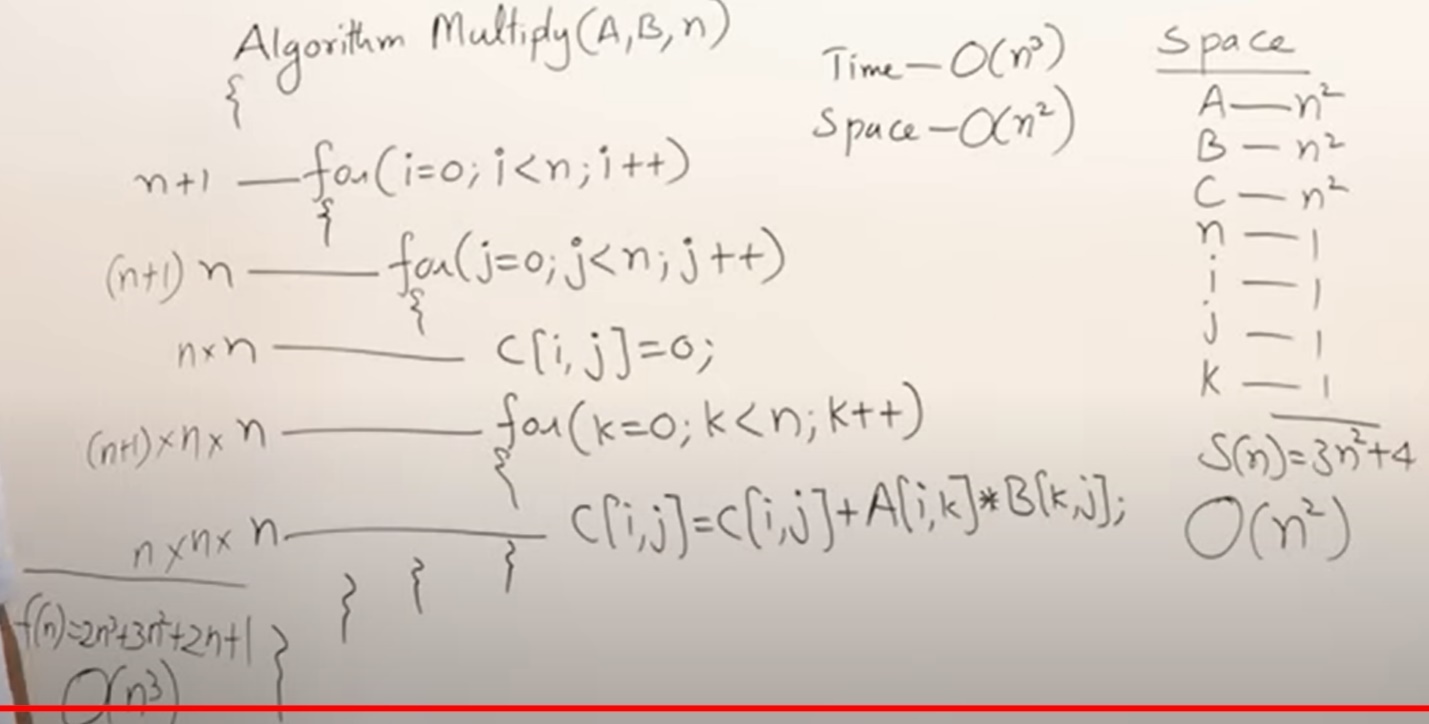
Space complexity – Since A, B and C are 2 dimensional array ( its an assumption for this program). So complexity will be n^2

Other variables are – n, I, j. Their complexity will be 1.

So total is – 3n^2 + 3

And degree will be – O(n^2) – Order of n square.

Example 4 :



Explanation – Above program

It has 3 loops, it will get time complexity

Total time complexity will come as shown in the pic.

Then, highest degree is 3. SO, it will be Order of n cube – O(n^3).

And we can see Space complexity as well. Its self explanatory.

Example 5 :

for(i=n;i>0;i--){

statement; // n times it will execute.

}

Complexity will be O(n).

Example 6 :

for(i=1;i<n;i=i+2){

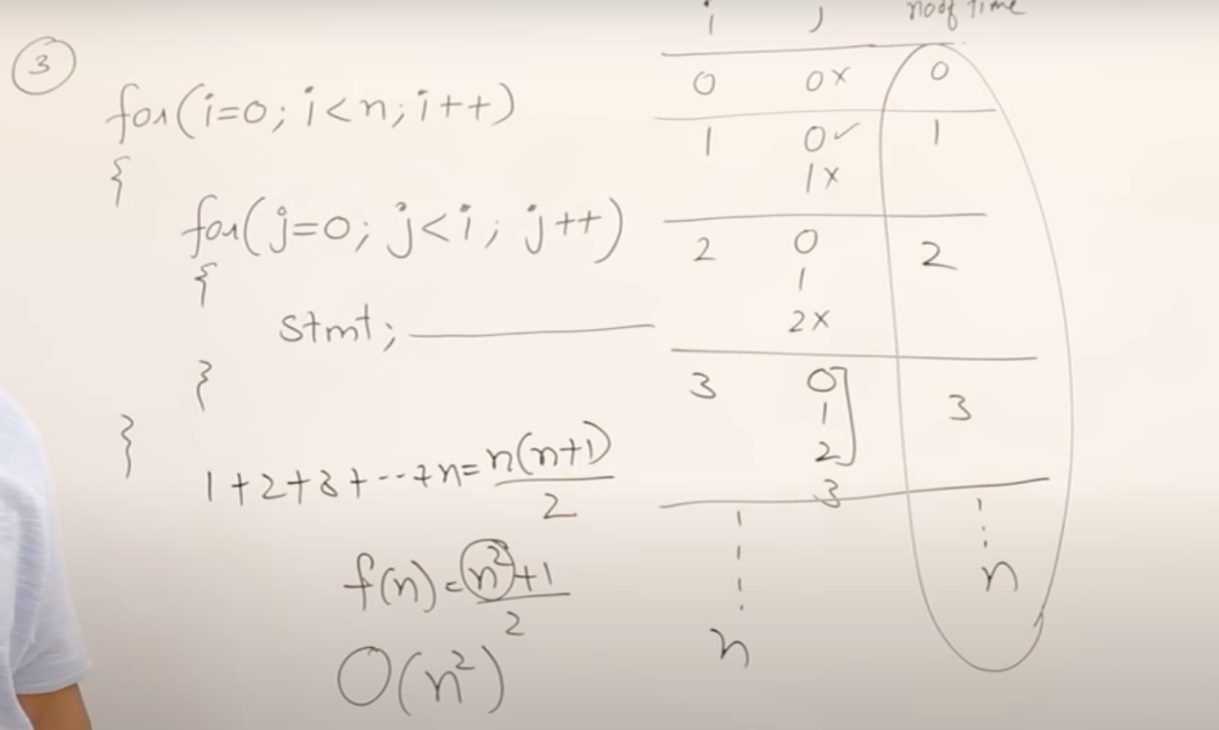
statement; // n/2 times it will execute.

}

Complexity : f(n) = n/2

Here degree of polynomial is still n, so Order will O(n)

Example 7 :



Explanation – In above algo, if we see the 2nd loop, j value depends on i value to work upon. So, we will identify the no. of times inner statement works.

In above pic, no of times is used for the statement for inner loop.

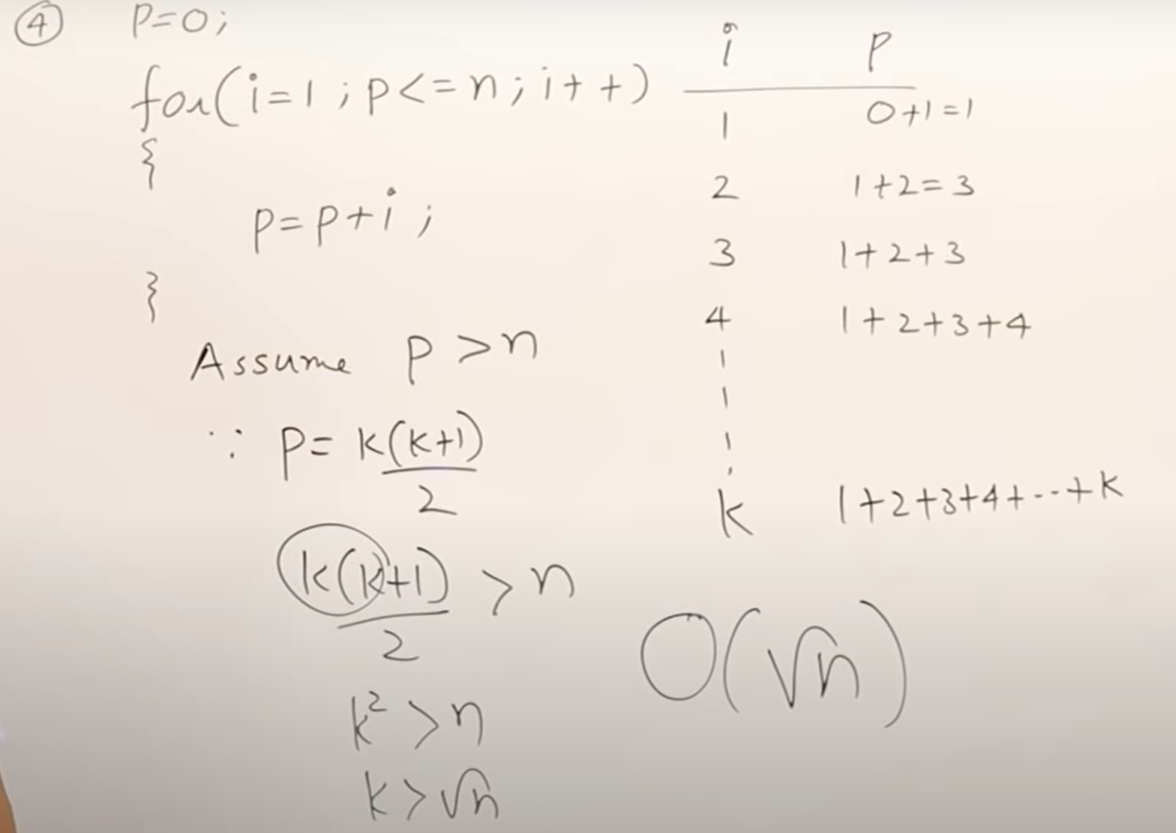
Basically this statement will not work based on a continuous loop, that the reason, we came to its loop like, 1+2+3+4=…..n

So, f(n) = n(n+1)/2

F(n) = n^2 + n/2

Order degree will be O(n^2) – degree of a polynomial is n square.

Example 8 :



Explanation – In this algo, loop will work with a condition that p<=n. p value is increasing though, but suppose if we see that n value is 100. For n being 100, p will run some “k” times. While we don’t know the exact value of n, we’ll define 1 variable called “k”. That’s just a temp variable using to find program complexity.

In above loop, for each incremented value of i, like 1,2,3,4… K. till k

Value of p will also be like “1+2+3+4 …..+k”.

So, will get p value as p = k(k+1)/2

Now, this loop will start when p>n,

So calculation wud be :

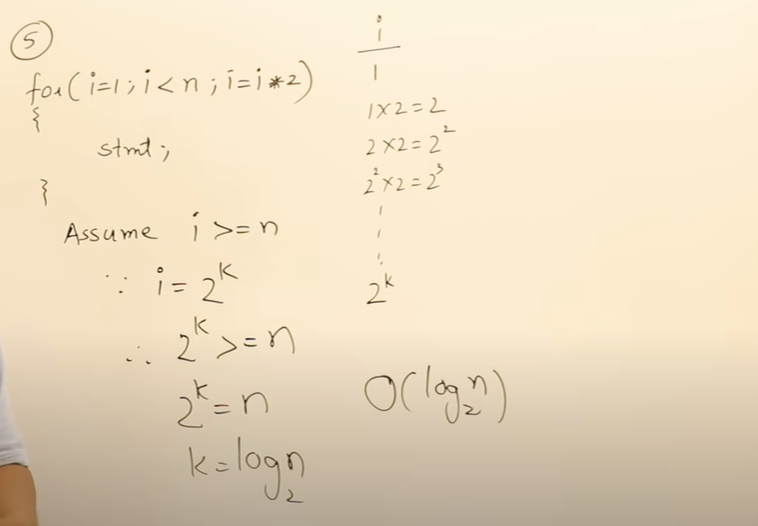
P =k(k+1)/2

K^2 > n;

k> square root of n.

In O degree of the polynomial, we say like O(square root of n)

Example 9 :



Explanation – I loop will go like 2, 2 power 2, 2 power 3, and go on….

Loop will stop when i>=n

And we will take a variable k such that, i = 2^k

Means, 2^k <=n

Or we can say, 2^k = n;

K = log n base 2

So, Time complexity will be log n base 2

Order will be O(log n base 2)

Note : Whenever we see like counter i is like multiplying by 2 or 3. Then mostly time complexity will come as log n

1. **Ceil and Floor functions in C++ or use in Complexity technique :**

**Ans** : the floor and ceiling functions map a real number to the greatest preceding or the least succeeding integer, respectively.

**floor(x) :**Returns the largest integer that is smaller than or equal to x (i.e : rounds downs the nearest integer).

Here, x is the floating point value, return the largest integer smaller than or equal to x.

Format is double floor(double x)

Examples : Input : 2.5

Output : 2

Input : -2.1

Output : -3

Input : 2.9

Output: 2

**ceil(x) :**Returns the smallest integer that is greater than or equal to x (i.e : rounds up the nearest integer).

Here, x is the floating value, returns the smallest integers greater than or equal to x.

Format : double ceiling(double x)

Input : 2.5

Output : 3

Input : -2.1

Output : -2

Input : 2.9

Output: 3

Info : A **StackOverflowError** is a runtime error in java. It is thrown when the amount of call stack memory allocated by JVM is exceeded. A common case of a StackOverflowError being thrown, is when call stack exceeds due to excessive deep or infinite recursion.

Java Program to illustrate floor & ceil functions –

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

Final f = **new** Final();

Double a = f.floor(12.9);

System.***out***.println(a);

Double b = f.ceil(12.9);

System.***out***.println(b);

System.***out***.println(f.floor(0.95));

System.***out***.println(f.ceil(0.95));

}

**public** **double** floor(**double** x) {

**return** Math.*floor*(x);

}

**public** **double** ceil(**double** x) {

**return** Math.*ceil*(x);

}

Output :

12.0

13.0

0.0

1.0

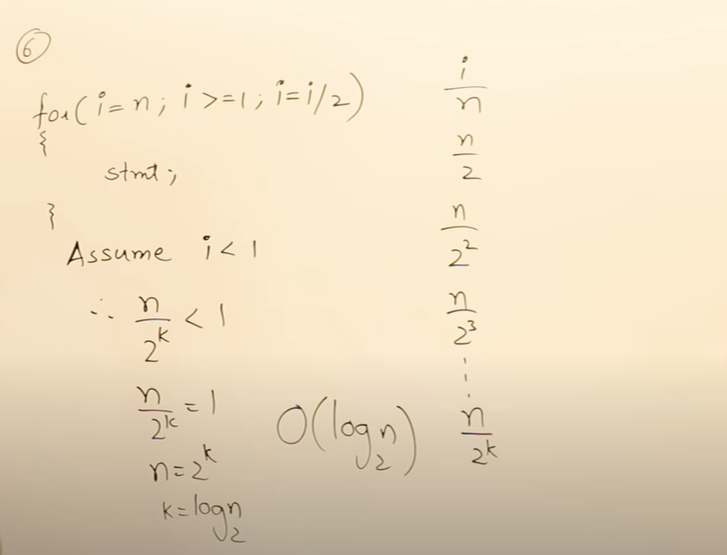
1. **Some More examples on Time complexity –**

Example 1 :

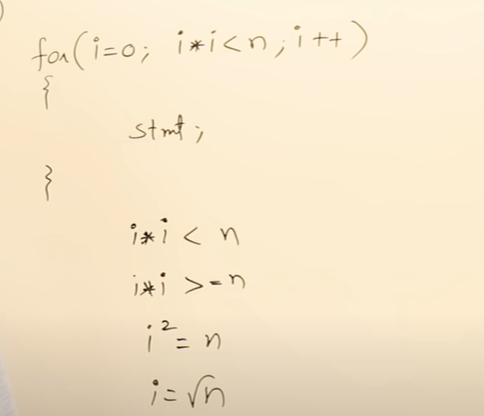
Explanation – In below example code, i value is always getting dividing by 2. And it started value from n So in the end it got value of n/2^k

After further calculations, it will get value of (log n base 2)

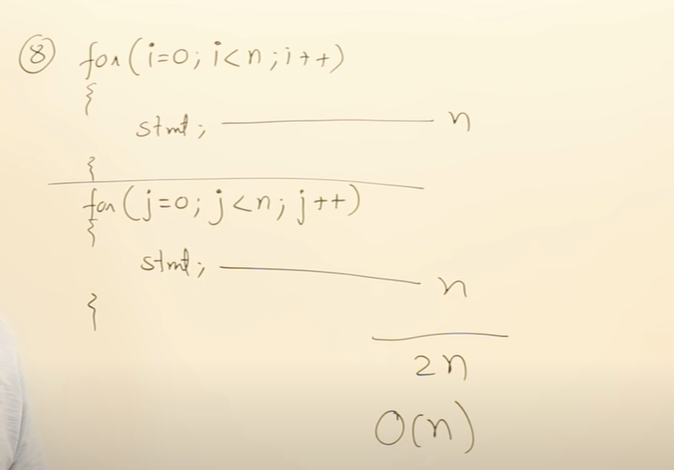
O(log n base 2)



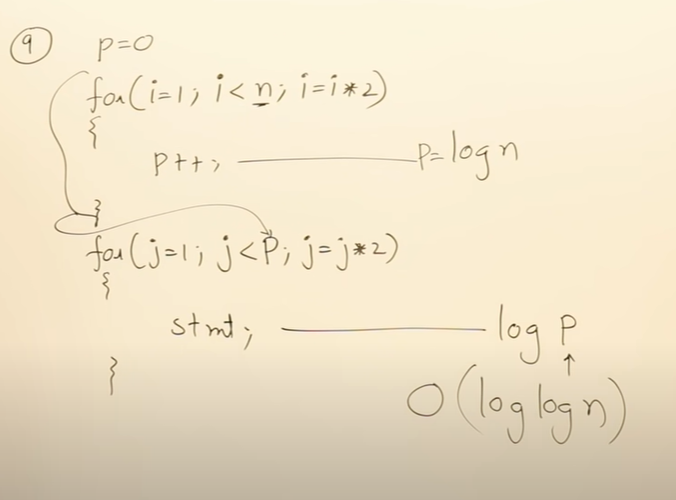
Example 2 :



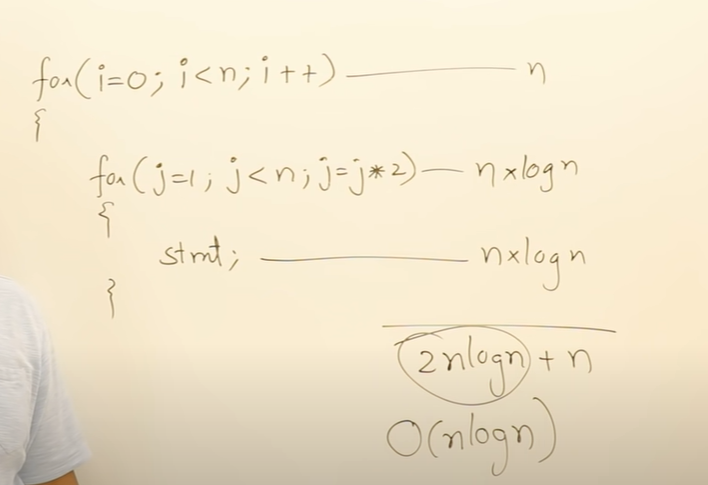
Example 3 :



Example 4:

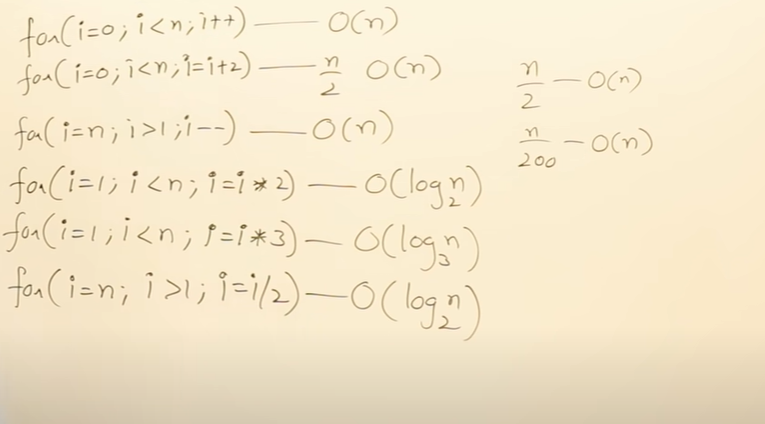


Example 5 :



Example 6 :

If we summarize what we have seen, we got the below formulas :



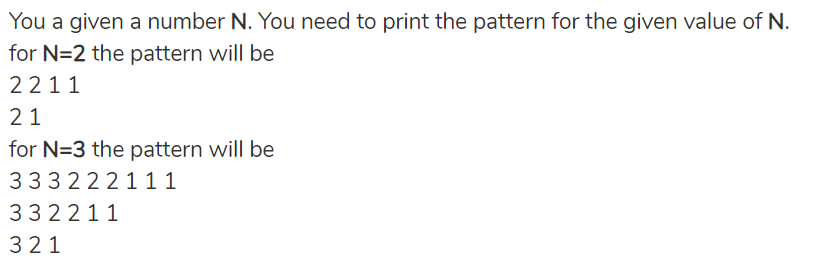
1. **Exception Handling –**

Ans : Exception is an error event that can happen during the execution of a program and disrupts its normal flow. Java provides a robust and object oriented way to handle exception scenarios, known as **Java Exception Handling**

When the exception occurs in a method, the process of creating the exception object and handing it over to runtime environment is called **“throwing the exception”**.

**IMP Point to note** : Note that Java Exception handling is a framework that is used to handle runtime errors only, compile time errors are not handled by exception handling in java.

1. **Code to solve the below pattern :**



Code will be :

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

**int** k=4;

**for**(**int** j=k;j>=1;j--) {

**for**(**int** i=k;i>=1;i--) {

**int** count = 0;

**while**(count!=j) {

System.***out***.print(i+" ");

count+=1;

}

}

System.***out***.println();

}

1. **Java Program – To find all occurrences of 1 String into another String**

We have 2 Strings given below, we have to find occurrence of pat inside txt

String txt = "AABAACAADAABAABA";

String pat = "AABA";

String spaceTxt = txt.replace(" ", "");

**for**(**int** i=0;i<spaceTxt.length();i++) {

**for**(**int** j=i+1;j<=spaceTxt.length();j++) {

**if**(spaceTxt.substring(i,j).equals(pat)) {

System.***out***.println(i);

}

}

}

1. **Java Program to check whether the given Integer array has number in Fibonacci series ?**

**Ans:**

**To check whether the input Array is Fibonacci series or not ..**

**int** arr[] = {0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55};

**boolean** flag = **false**;

**for**(**int** i=0;i<arr.length-2;i++) {

**if**(arr[i]+arr[i+1]==arr[i+2]) {

flag = **true**;

}

**else** flag = **false**;

}

System.***out***.println(flag);

**Java Program to create a Fibonacci series of n=10;**

**Using simple for loop –**

**int** n = 10;

**int** firstNum = 0;

**int** secondNum = 1;

System.***out***.print(firstNum+" "+secondNum+" ");

**for**(**int** i=0;i<n;i++) {

**int** sum = firstNum + secondNum;

firstNum = secondNum;

secondNum = sum;

System.***out***.print(sum+" ");

}

**Using Recursion –**

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

// 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55 89 etc.

**int** n = 10;

**int** firstNum = 0;

**int** secondNum = 1;

System.***out***.print(firstNum+" "+secondNum+" ");

*recur*(firstNum,secondNum);

}

**static** **int** *count* = 0;

**public** **static** **void** recur(**int** a, **int** b) {

**while**(*count* < 10) {

**int** finalSum = a+b;

System.***out***.print(finalSum+" ");

*count*+=1;

*recur*(b,finalSum);

}

}

1. **JAVA Program – To find the continuous sub array whose Sum is equal to the given Sum-**

For ex, we have 1 array like int arr2[] = {1, 2, 3, 4, 5, 6,7, 8, 9, 10};

And we have Sum = 15.

Now we have to find the first occurring sub array from the left if sum equals to subarray.

Code:

**int** arr1[] = {1, 2, 3, 7, 5};

**int** arr2[] = {1, 2, 3, 4, 5, 6,7, 8, 9, 10};

// Sum is 12 for the 1st array and 15 for 2nd array

**int** sum1 = 15;

**int** dummySum=0, firstIndex = 0, lastIndex = 0;

**boolean** flag = **false**;

**for**(**int** i=0;i<arr2.length;i++) {

**for**(**int** j=i;j<arr2.length;j++) {

**if**(dummySum==sum1) {

System.***out***.println(firstIndex);

System.***out***.println(lastIndex);

flag=**true**;

**break**;

}**else** **if**(dummySum < sum1) {

dummySum = dummySum + arr2[j]; //

firstIndex = i;

lastIndex= j;

}

**else** **if**(dummySum>sum1) {

dummySum=0;

firstIndex = 0;

lastIndex = 0;

**break**;

}

}

**if**(flag==**true**) {

**break**;

}

}

Time Complexity of above program is O(n\*n).

Now, we have 1 more option which will have Time Complexity of O(n).

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

**int** arr[] = {1, 2, 3, 6, 5, 6,7,8, 9, 10}; // 5 6 7= 18 3 4 5 6 // final is 2 and 5

**int** s = 18;

**int** i=0,sum=0;

**for**(**int** j=0;j<arr.length;j++){

sum+=arr[j]; // sum is 22+7=29

**while**(sum>s)

sum-=arr[i++];

**if**(sum==s){

System.***out***.println((i+1)+" "+(j+1)); // i become 5 and j become

**return**;

}

}

System.***out***.println(-1);

}

1. **JAVA Program – To find the difference of Index –**

Problem statement is – We have an integer array. Then we have to find the maximum value of j-i when we have a condition that A[i]<=A[j]

Now, what is I and j. These are just indexes of the single array itself.

Suppose we have Array available int A[] = {32, 8, 10, 3, 2, 80, 30, 33, 1};

Now, we’ll see that element at i=0 index is 32 and element at j=7 index is 33. And difference b/w these 2 indexes is j-I = 7-0 =7

So**, output is 7 which** is the max diff b/w 2 indexes subject to the above condition.

Code:

**int** A[] = {32, 8, 10, 3, 2, 80, 30, 33, 1};

**int** tempdiff = 0;

**int** permdiff = 0;

**for**(**int** i=0;i<A.length;i++) { // i is 34 and j is 1

**for**(**int** j=A.length-1;j>i;j--) {

**if**(A[i]<=A[j]) {

tempdiff= j-i;

**if**(tempdiff>permdiff) {

permdiff = tempdiff;

}

}

**else** **continue**;

}

}

System.***out***.println(permdiff);

1. JAVA Program -