
UNIVERSITÀ
DEGLI STUDI DI BARI
ALDO MORO

Documentazione di Progetto

---

## Dipartimento di Informatica

Corso di laurea “Informatica e Tecnologie per la Produzione del software”

# Integrazione e Test di Sistemi Software 2022/2023

---

Prof.ssa **Azzurra Ragone**

TEAM: **GAF**

| Membri: | | Contatti |
|---|---|---|
| **Giuliani Federico** | Mat: 742994 | f.giuliani10@studenti.uniba.it |
| **Antonio Iungo** | Mat: 735394 | a.iungo@studenti.uniba.it |
| **Gabriele Leone** | Mat: 736749 | g.leone94@studenti.uniba.it |

# SOMMARIO

# HOMEWORK1:Specification-based testing secondo i sette punti

## 1. Comprendere i requisiti

1. Il programma ha due obiettivi: il primo è il calcolo del quadrato dei numeri pari e il cubo dei numeri dispari, presenti in un array di numeri interi positivi riempito dall'utente, che vengono ordinati in modo crescente. Il secondo è l'ordinamento di una serie di nomi , sempre inseriti dall'utente.
2. Il programma riceve 2 array riempiti dall'utente:
    - numeri() che rappresenta l'array composto da numeri interi, da cui il programma effettuerà le varie operazioni a seconda della loro posizione.
    - nomi() che rappresenta l'array composto da stringhe , su cui verrá eseguito l'ordinamento.
3. Il programma restituisce 2 array in output , il primo contenente la lista di numeri ordinati in ordine crescente , e il secondo contenente le stringhe ordinate in ordine alfabetico. Tutto questo viene eseguito se i quadrati non superano il valore 99 e i cubi il valore 199

## 2. Esplora cosa fa il programma per vari input

```
> Task :Main.main()
1
marco
Codice terminato con successo
```

```
> Task :Main.main()
Non e' possibile inserire una grandezza pari o minore di 0
```

```
> Task :Main.main()
4
16
Marco
lucia
Codice terminato con successo
```

```
> Task :Main.main()
Il numero di elementi inseriti non corrisponde
```

```
> Task :Main.main()
lucia
Codice terminato con successo
```

```
> Task :Main.main()
27
federica
lucia
Codice terminato con successo
```

```
> Task :Main.main()
36
64
1federica
2lucia
Codice terminato con successo
```

```
> Task :Main.main()
Il numero inserito e' troppo grande (offpoint)
```

```
> Task :Main.main()
Non e' possibile inserire un numero negativo
```

```
> Task :Main.main()
Non e' possibile inserire stringhe nulle
```

```
> Task :Main.main()
36
64

lucia
Codice terminato con successo
```

```
C:\Users\giuli\Desktop\esameIntegrazioneTest\src\main\java\org\example\Main.java:18: error: incompatible types:
        int[] numeri = {"ciao",6};
                        ^
```

```
> Task :Main.main()
1
1
123
2
Codice terminato con successo
```

```
> Task :Main.main()
0
0
giulio
marco
Codice terminato con successo
```

```
> Task :Main.main()
1
4
Zero
antonio
Codice terminato con successo
```

```
C:\Users\giuli\Desktop\esameIntegrazioneTest\src\main\java\org\example\Main.java:18: error: incompatible types:
        int[] numeri = {1.2,2};
                        ^
```

```
> Task :Main.main()
16
16
piero
piero
Codice terminato con successo
```

## 3. Esplora input, output e identifica le partizioni

## a) Ingressi Individuali:

- **parametro n:**
  - **n <= 0**
  - **n > 0**

**parametro m:**

- **m <= 0**
- **m > 0**

- **Array nomi [ ]**
  - Stringhe di lunghezza vuota
  - Stringhe di lunghezza 1
  - Stringhe di lunghezza maggiore di 1
  - Stringhe null
  - Numeri

- **Array numeri [ ]**
  - Numeri positivi
  - Numeri = 0
  - Numeri negativi
  - Non interi
  - Numeri OffPoint

## b) Combinazioni di input:

**Combinazioni n (numero di nomi) ed m (numero di numeri)**

1. n diverso da m
2. n o m negativi
3. n uguale ad m
4. n oppure m sono numeri di tipo non intero
5. n uguale ad m entrambi uguali 0

## c) Classi di output:

1. Array di stringhe (output):
   a. Array nullo
   b. Array vuoto
   c. Singolo elemento
   d. Elementi multipli

2. Ogni stringa individuale (output):
    a. Vuoto
    b. Singolo carattere
    c. Caratteri multipli

3. Array di Numeri (output):
    a. Array vuoto
    b. Array con meno elementi di quelli iniziali
    c. Array di numeri ordinati:
    d. Singolo elemento
    e. Caratteri multipli

4. Array di nomi ordinati:
    a. Elementi multipli
    b. Caratteri multipli

# 4. Identifica i casi limite

Possiamo trovare diversi casi limite in base all'intervallo considerato:
Per l'intervallo 0 <= x^3 < 199:

- Gli ***on point*** sono i numeri interi x che soddisfano l'equazione x^3 = n, dove n è un numero intero compreso tra 0 e 199 inclusi. Possiamo calcolarli trovando le radici cubiche intere di ogni numero intero tra 0 e 199. Ci sono 4 cubi perfetti in questo intervallo:  0,1, 27,125 dati da 0,1,3,5.
- Gli ***off point*** sono i numeri interi piú vicini al limite che non soddisfano i requisiti,in questo intervallo possiamo individuare il 7 come numero off point.
- Gli ***in point*** sono i numeri interi x che soddisfano l'equazione x^3 = 199, ovvero non ce ne sono.
- Gli ***out point*** sono i numeri interi x per i quali x^3 è maggiore o uguale a 199, ovvero tutti i numeri interi maggiori di 5.

Per l'intervallo 0 <= x^2 < 99:

- Gli ***on point*** sono i numeri interi x che soddisfano l'equazione x^2 = n, dove n è un numero intero compreso tra 0 e 99. Possiamo calcolarli trovando le radici quadrate intere di ogni numero intero tra 0 e 99. Ci sono 5 quadrati perfetti in questo intervallo: 0, 4, 16, 36, 64 dati da 0,2,4,6,8.
- Gli ***off point*** sono i numeri interi piú vicini al limite che non soddisfano i requisiti,in questo intervallo possiamo individuare il 10 come numeri off point
- Gli ***in point*** sono i numeri interi x che soddisfano l'equazione x^2 = 99, ovvero non ce ne sono.

- Gli ***out point*** sono i numeri interi x per i quali x^2 è maggiore o uguale a 99, ovvero tutti i numeri interi maggiori di 8.

- Infine per quanto riguarda i numeri accettabili in input , abbiamo come off Point il numero 100 che non permette la corretta esecuzione del programma.

# 5. Elaborare casi di test

**Testiamo innanzitutto i casi limite , dove il programma non prosegue con la stampa/ genera eccezione**

## Casi Eccezionali:

T1: m uguale ad n ed entrambi uguali a 0
T2: n negativo
T3: m negativo
T4: n diverso da m
T5: n non intero
T6: m non intero
T7: numero inserito nell'array numeri è negativo
T8: numero inserito nell'array numeri non è intero
T9: numero inserito nell'array numeri é Offpoint
T10: elemento inserito nell'array nomi è null

## Per n=m ci possono essere le seguenti combinazioni:

T11:Nomi [] contiene stringhe non vuote di lunghezza maggiore di 1 e Numeri[] contiene n numeri positivi
T12:Nomi [] contiene stringhe non vuote di lunghezza maggiore di 1 e Numeri[] contiene almeno un numero uguale a 0
T13: Nomi [] contiene stringhe non vuote di lunghezza maggiore di 1 e Numeri[] contiene almeno 1 numero il cui cubo/quadrato supera il limite
T14:Nomi [] contiene stringhe non vuote di lunghezza maggiore di 1 e Numeri[] contiene almeno un numero uguale a 1
T15: Nomi [] contiene stringhe di lunghezza 1 e Numeri[] contiene n numeri positivi
T16: Nomi [] contiene stringhe di lunghezza 1 e Numeri[] contiene almeno un numero uguale a 0
T17: Nomi [] contiene stringhe di lunghezza 1 e Numeri[] contiene almeno 1 numero il cui cubo/quadrato supera il limite
T18 Nomi [] contiene stringhe di lunghezza 1 e Numeri[] contiene almeno un numero uguale a 1
T19: Nomi [] contiene almeno 1 stringa vuota e Numeri[] contiene n numeri positivi

T20: Nomi [] contiene almeno 1 stringa vuota e Numeri[] contiene almeno un numero uguale a 0
T21: Nomi [] contiene almeno 1 stringa vuota e Numeri[] contiene almeno 1 numero il cui cubo/quadrato supera il limite
T22: Nomi [] contiene almeno 1 stringa vuota e Numeri[] contiene almeno un numero uguale a 1
T23: Nomi [] contiene numeri e Numeri[] contiene n numeri positivi
T24: Nomi [] contiene numeri e Numeri[] contiene almeno un numero uguale a 0
T25: Nomi [] contiene numeri e Numeri[] contiene almeno 1 numero il cui cubo/quadrato supera il limite
T26: Nomi [] contiene numeri e Numeri[] contiene almeno un numero uguale a 1
T27: Il numero di elementi contenuti negli array numeri e nomi non corrisponde alle variabili m ed n

# 6. Automatizzare i casi di test

## T1: m uguale ad n ed entrambi uguali a 0

```
//T1 m uguale ad n ed entrambi uguali 0
@ParameterizedTest
@MethodSource("provideIntegers")
void NequalsMequals0(int n, int m , String[] nomi,int[] numeri){

    Main.effettuaOperazioni(n,m,nomi,numeri);
    assertEquals(m, actual: 0);
    assertEquals(n, actual: 0);
    assertEquals(n,m);
}

1 usage
static Stream<Arguments> provideIntegers() {
    return Stream.of(
            Arguments.of( ...arguments: 0,0,new String[]{"Ciao"},new int[]{1})
    );
}
```

T2: n negativo

```java
//T2 n negativo
@ParameterizedTest
@MethodSource("test2")
public void nShouldBeNegative(int n, int m , String[] nomi,int[] numeri) throws Exception
{
    Main.effettuaOperazioni(n,m,nomi,numeri);
    assertTrue( condition: n < 0 );
}

1 usage
static Stream<Arguments> test2() {
    return Stream.of(
            Arguments.of( ...arguments: -1,3,new String[]{"Ciao"},new int[]{1})
    );
}
```

T3: m negativo

```java
//T3 m negativo
@ParameterizedTest
@MethodSource("test3")
public void mShouldBeNegative(int n, int m , String[] nomi,int[] numeri) throws Exception
{
    Main.effettuaOperazioni(n,m,nomi,numeri);
    assertTrue( condition: m < 0 );
}
1 usage
static Stream<Arguments> test3() {
    return Stream.of(
            Arguments.of( ...arguments: 4,-10,new String[]{"Ciao"},new int[]{1})
    );
}
```

T4: n diverso da m

```java
//T4 n diverso da m
@ParameterizedTest
@MethodSource("test4")
public void MNotEqualsN(int n, int m , String[] nomi,int[] numeri) throws Exception
{
    Main.effettuaOperazioni(n,m,nomi,numeri);
    assertTrue( condition: m != n);
}

1 usage
static Stream<Arguments> test4() {
    return Stream.of(
            Arguments.of( ...arguments: 4,-10,new String[]{"Ciao"},new int[]{1})
    );
}
```

T5: n non intero

```java
//T5 n non intero
@ParameterizedTest
@MethodSource("test5")
public void MisInteger ( Object n, int m, String[] nomi,int[] numeri)  {
    boolean local1=false;
    try {
        int intValue = Integer.parseInt(n.toString()); // Conversione del primo argomento a un intero
    } catch (NumberFormatException e) {
        local1=true;
    }
    assertTrue(local1);
}
1 usage
static Stream<Arguments> test5 () {
    return Stream.of(
            Arguments.of( ...arguments: "p", 1, new String[]{"Ciao"}, new int[]{1}),
            Arguments.of( ...arguments: "p", 2, new String[]{"Ciao", "Ciao", "Ciao"}, new int[]{4, 1, 4})
    );
}
```

T6: m non intero

```java
//T6 m non intero
@ParameterizedTest
@MethodSource("NisInteger")
public void NisInteger ( int n, Object m, String[] nomi,int[] numeri)  {
    boolean local1=false;
    try {
        int intValue = Integer.parseInt(m.toString()); // Conversione del primo argomento a un intero
    } catch (NumberFormatException e) {
        local1=true;
    }
    assertTrue(local1);
}
static Stream<Arguments> NisInteger () {
    return Stream.of(
            Arguments.of( ...arguments: 1,"p", new String[]{"Ciao"}, new int[]{1}),
            Arguments.of( ...arguments: 1,"p", new String[]{"Ciao", "Ciao", "Ciao"}, new int[]{4, 1, 4})
    );
}
```

T7: numero inserito nell’array numeri è negativo

```java
//T7: numero inserito nell'array numeri è negativo
@ParameterizedTest
@MethodSource("provideIntArraysWithNegative")
void testNegativeNumberInArray(int n, int m,String[] nomi,int[] numeri) throws NegativeNumberException {
    Main.effettuaOperazioni(n,m,nomi,numeri);
    assertTrue(Arrays.stream(numeri).anyMatch(numero -> numero < 0 ),
            message: "numeri contiene almeno 1 numero negativo: " + Arrays.toString(numeri));
}

1 usage
private static Stream<Arguments> provideIntArraysWithNegative() {
    return Stream.of(
            Arguments.of( ...arguments: 1, 1,new String[]{"ciao", "hola"},new int[]{1,-1}),
            Arguments.of( ...arguments: 2, 2,new String[]{"ciao", "hola", "hello"},new int[]{1, 1,-1})
    );
}
```

T8: numero inserito nell'array numeri non è intero

```java
//T8 numero inserito nell'array numeri non è intero
@ParameterizedTest
@MethodSource("numberinArrayisnotInteger")
public void numberinArrayisnotInteger ( int n, int m, String[] nomi,Object[] numeri)  {
    boolean local1=false;
    try {
        int intValue = Integer.parseInt(numeri.toString());
    } catch (NumberFormatException e) {
        local1=true;
    }
    assertTrue(local1);
}
static Stream<Arguments> numberinArrayisnotInteger () {
    return Stream.of(
            Arguments.of( ...arguments: 1,1, new String[]{"Ciao"}, new Object[]{3, "c"}),
            Arguments.of( ...arguments: 2,2, new String[]{"Ciao", "Ciao", "Ciao"}, new Object[]{"cio", 1, 4})
    );
}
```

T9: elemento inserito nell'array numeri è offoint

```java
//T9: elemento inserito nell'array numeri e' offpoint
@ParameterizedTest
@MethodSource("testOffpointNumber")
void testOffpointNumber(int n, int m,String[] nomi,int[] numeri) throws NullPointerException {
    Main.effettuaOperazioni(n,m,nomi,numeri);
    assertTrue(Arrays.stream(numeri).anyMatch(numero -> numero >= 100 ),
            message: "numeri contiene almeno un numero offpoint: " + Arrays.toString(numeri));
}

private static Stream<Arguments> testOffpointNumber() {
    return Stream.of(
            Arguments.of( ...arguments: 1, 1,new String[]{"c",null},new int[]{1,1}),
            Arguments.of( ...arguments: 2, 2,new String[]{"ciao", null,"c"},new int[]{1, 3, 2})
    );
}
```

T10: elemento inserito nell'array nomi è null

```java
//T10: elemento inserito nell'array nomi è null
@ParameterizedTest
@MethodSource("testNameNullInArray")
void testNameNullInArray(int n, int m,String[] nomi,int[] numeri) throws NullPointerException {
    Main.effettuaOperazioni(n,m,nomi,numeri);
    assertTrue(Arrays.stream(nomi).anyMatch(nome -> nome == null ),
            message: "nomi contiene una stringa null: " + Arrays.toString(nomi));
}

private static Stream<Arguments> testNameNullInArray() {
    return Stream.of(
            Arguments.of( ...arguments: 1, 1,new String[]{"c",null},new int[]{1,1}),
            Arguments.of( ...arguments: 2, 2,new String[]{"ciao", null,"c"},new int[]{1, 3, 2})
    );
}
```

T11:Nomi [] contiene stringhe non vuote di lunghezza maggiore di 1 e Numeri[] contiene n numeri positivi

```java
//T11:Nomi [] contiene stringhe non vuote e Numeri[] contiene n numeri positivi
@ParameterizedTest
@MethodSource("testNameIntInArray")
void testNameIntInArray(int n, int m,String[] nomi,int[] numeri)  {
    Main.effettuaOperazioni(n,m,nomi,numeri);

    assertFalse(Arrays.stream(numeri).anyMatch(numero -> numero <= 0),
            message: "i numeri sono tutti positivi: " + Arrays.toString(numeri));

    assertFalse(Arrays.stream(nomi).anyMatch(nome -> nome == ""),
            message: "il nome e' vuoto: " + Arrays.toString(nomi));

}

private static Stream<Arguments> testNameIntInArray() {
    return Stream.of(
            Arguments.of( ...arguments: 1, 1,new String[]{"ciao", "halo"},new int[]{1,1}),
            Arguments.of( ...arguments: 2, 2,new String[]{"ciao", "halo","hello"},new int[]{1, 1,4})
    );
}
```

T12: Nomi [] contiene stringhe non vuote di lunghezza maggiore di 1 e Numeri[] contiene almeno un numero uguale a 0

```java
// T12: Nomi [] contiene stringhe non vuote e Numeri[] contiene almeno un numero uguale a 0
@ParameterizedTest
@MethodSource("testNumber0InArray")
void testNumber0InArray(int n, int m,String[] nomi,int[] numeri) {
    Main.effettuaOperazioni(n,m,nomi,numeri);

    assertFalse(Arrays.stream(nomi).anyMatch(nome -> nome == ""),
            message: "il nome e' vuoto: " + Arrays.toString(nomi));

    assertTrue(Arrays.stream(numeri).anyMatch(numero -> numero == 0),
            message: "Lo 0 non è presente nell'array numeri: " + Arrays.toString(numeri));
}

private static Stream<Arguments> testNumber0InArray() {
    return Stream.of(
            Arguments.of( ...arguments: 1, 1,new String[]{"ciao", "ll"},new int[]{1,0}),
            Arguments.of( ...arguments: 2, 2,new String[]{"ciao", "halo","hello"},new int[]{1, 2,0})
    );
}
```

T13: Nomi [] contiene stringhe non vuote di lunghezza maggiore di 1 e Numeri[] contiene almeno 1 numero il cui cubo/quadrato supera il limite

```java
//T13 Nomi [] contiene stringhe non vuote e Numeri[] contiene almeno 1 numero il cui cubo/quadrato supera il limite
@ParameterizedTest
@MethodSource("testStringNotEmptyandNumberThatExceedLimit")
void testStringNotEmptyandNumberThatExceedLimit(int n, int m,String[] nomi,int[] numeri) {
    Main.effettuaOperazioni(n,m,nomi,numeri);

    assertFalse(Arrays.stream(nomi).anyMatch(nome -> nome == ""),
            message: "il nome e' vuoto: " + Arrays.toString(nomi));

    assertTrue(Arrays.stream(numeri).anyMatch(numero -> (numero % 2 == 0) && numero >= 10 || numero > 5 ),
            message: "numri contiene un numero il cui cubo/quadrato supera il limite: " + Arrays.toString(numeri));
}

private static Stream<Arguments> testStringNotEmptyandNumberThatExceedLimit() {
    return Stream.of(
            Arguments.of( ...arguments: 1, 1,new String[]{"ciao", "ll"},new int[]{7,3}),
            Arguments.of( ...arguments: 2, 2,new String[]{"ciao", "halo","hello"},new int[]{11,2,1})
    );
}
```

T14:Nomi [] contiene stringhe non vuote di lunghezza maggiore di 1 e Numeri[] contiene almeno un numero uguale a 1

```java
//T14 Nomi [] contiene stringhe non vuote di lunghezza maggiore di 1 e Numeri[] contiene almeno un numero uguale a 1
@ParameterizedTest
@MethodSource("testStringNotEmptyandNumberone")
void testStringNotEmptyandNumberone(int n, int m,String[] nomi,int[] numeri) {
    Main.effettuaOperazioni(n,m,nomi,numeri);

    assertFalse(Arrays.stream(nomi).anyMatch(nome -> nome == ""),
            message: "il nome e' vuoto: " + Arrays.toString(nomi));

    assertTrue(Arrays.stream(numeri).anyMatch(numero -> numero == 1),
            message: "numeri contiene un numero uguale a 1: " + Arrays.toString(numeri));
}

private static Stream<Arguments> testStringNotEmptyandNumberone() {
    return Stream.of(
            Arguments.of( ...arguments: 1, 1,new String[]{"ciao", "ll"},new int[]{1,3}),
            Arguments.of( ...arguments: 2, 2,new String[]{"ciao", "halo","hello"},new int[]{11,2,1})
    );
}
```

T15: Nomi [] contiene stringhe di lunghezza 1 e Numeri[] contiene n numeri positivi

```java
//T15 Nomi [] contiene stringhe di lunghezza 1 e Numeri[] contiene n numeri positivi
@ParameterizedTest
@MethodSource("String1andNumberPositive")
void String1andNumberPositive(int n, int m,String[] nomi,int[] numeri){
    Main.effettuaOperazioni(n,m,nomi,numeri);

    assertFalse(Arrays.stream(nomi).anyMatch(nome -> nome.length() != 1 ),
            message: "nomi contiene almeno 1 stringa di lunghezza 1: " + Arrays.toString(nomi));

    assertFalse(Arrays.stream(numeri).anyMatch(numero -> numero <= 0),
            message: "i numeri sono tutti positivi: " + Arrays.toString(numeri));
}
private static Stream<Arguments> String1andNumberPositive() {
    return Stream.of(
            Arguments.of( ...arguments: 1, 1,new String[]{"s", "c"},new int[]{2,1}),
            Arguments.of( ...arguments: 2, 2,new String[]{"a", "s","d"},new int[]{3,2,8})
    );
}
```

T16: Nomi [] contiene stringhe di lunghezza 1 e Numeri[] contiene almeno un numero uguale a 0

```java
//T16 Nomi [] contiene stringhe di lunghezza 1 e Numeri[] contiene almeno un numero uguale a 0
@ParameterizedTest
@MethodSource("String1and1number0")
void String1and1number0(int n, int m,String[] nomi,int[] numeri){
    Main.effettuaOperazioni(n,m,nomi,numeri);

    assertFalse(Arrays.stream(nomi).anyMatch(nome -> nome.length() != 1 ),
            message: "nomi contiene almeno 1 stringa di lunghezza 1: " + Arrays.toString(nomi));

    assertTrue(Arrays.stream(numeri).anyMatch(numero -> numero == 0),
            message: "Lo 0 non è presente nell'array numeri: " + Arrays.toString(numeri));
}

private static Stream<Arguments> String1and1number0() {
    return Stream.of(
            Arguments.of( ...arguments: 1, 1,new String[]{"s", "c"},new int[]{0,1}),
            Arguments.of( ...arguments: 2, 2,new String[]{"a", "s","d"},new int[]{3,1,0})
    );
}
```

T17: Nomi [] contiene stringhe di lunghezza 1 e Numeri[] contiene almeno 1 numero il cui cubo/quadrato supera il limite

```java
//T17 Nomi [] contiene stringhe di lunghezza 1 e Numeri[] contiene almeno 1 numero il cui cubo/quadrato supera il limite
@ParameterizedTest
@MethodSource("String1and1numbercubeover")
void String1and1numbercubeover(int n, int m,String[] nomi,int[] numeri){

   Main.effettuaOperazioni(n,m,nomi,numeri);

    assertFalse(Arrays.stream(nomi).anyMatch(nome -> nome.length() != 1 ),
            message: "nomi contiene almeno 1 stringa di lunghezza 1: " + Arrays.toString(nomi));

    assertTrue(Arrays.stream(numeri).anyMatch(numero -> (numero % 2 == 0) && numero >= 10 || numero > 5 ),
            message: "numri contiene un numero il cui cubo/quadrato supera il limite: " + Arrays.toString(numeri));

}
private static Stream<Arguments> String1and1numbercubeover() {
    return Stream.of(
            Arguments.of( ...arguments: 1, 1,new String[]{"s", "c"},new int[]{1,10}),
            Arguments.of( ...arguments: 2, 2,new String[]{"a", "s","d"},new int[]{3,12,0})
    );
}
```

T18:Nomi [] contiene stringhe di lunghezza 1 e Numeri[] contiene almeno un numero uguale a 1

```java
//T18:Nomi [] contiene stringhe di lunghezza 1 e Numeri[] contiene almeno un numero uguale a 1
@ParameterizedTest
@MethodSource("String1and1numberone")
void String1and1numberone(int n, int m,String[] nomi,int[] numeri){

    Main.effettuaOperazioni(n,m,nomi,numeri);

    assertFalse(Arrays.stream(nomi).anyMatch(nome -> nome.length() != 1 ),
            message: "nomi contiene almeno 1 stringa di lunghezza 1: " + Arrays.toString(nomi));

    assertTrue(Arrays.stream(numeri).anyMatch(numero -> numero == 1 ),
            message: "numeri contiene un numero 1: " + Arrays.toString(numeri));

}
private static Stream<Arguments> String1and1numberone() {
    return Stream.of(
            Arguments.of( ...arguments: 1, 1,new String[]{"s", "c"},new int[]{1,10}),
            Arguments.of( ...arguments: 2, 2,new String[]{"a", "s","d"},new int[]{3,1,0})
    );
}
```

T19: Nomi [] contiene almeno 1 stringa vuota e Numeri[] contiene n numeri positivi

```java
//T19: Nomi [] contiene almeno 1 stringa vuota e Numeri[] contiene n numeri positivi
@ParameterizedTest
@MethodSource("Stringnullandnpositive")
void Stringnullandnpositive(int n, int m,String[] nomi,int[] numeri){
    Main.effettuaOperazioni(n,m,nomi,numeri);

    assertTrue(Arrays.stream(nomi).anyMatch(nome -> nome == ""),
            message: "il nome e' vuoto: " + Arrays.toString(nomi));

    assertFalse(Arrays.stream(numeri).anyMatch(numero -> numero <= 0),
            message: "i numeri sono tutti positivi: " + Arrays.toString(numeri));

}
private static Stream<Arguments> Stringnullandnpositive() {
    return Stream.of(
            Arguments.of( ...arguments: 1, 1,new String[]{"a", ""},new int[]{1,100}),
            Arguments.of( ...arguments: 2, 2,new String[]{"a", "s",""},new int[]{2,12,3})
    );
}
```

T20: Nomi [] contiene almeno 1 stringa vuota e Numeri[] contiene almeno un numero uguale a 0

```java
//T20: Nomi [] contiene almeno 1 stringa vuota e Numeri[] contiene almeno un numero uguale a 0
@ParameterizedTest
@MethodSource("testNameNullNumber0InArray")
void testNameNullNumber0InArray(int n, int m,String[] nomi,int[] numeri) {
    Main.effettuaOperazioni(n,m,nomi,numeri);

    assertTrue(Arrays.stream(nomi).anyMatch(nome -> nome == ""),
            message: "il nome e' vuoto: " + Arrays.toString(nomi));

    assertTrue(Arrays.stream(numeri).anyMatch(numero -> numero == 0),
            message: "Lo 0 non è presente nell'array numeri: " + Arrays.toString(numeri));
}

private static Stream<Arguments> testNameNullNumber0InArray() {
    return Stream.of(
            Arguments.of( ...arguments: 1, 1,new String[]{"ciao", ""},new int[]{1,0}),
            Arguments.of( ...arguments: 2, 2,new String[]{"ciao", "","hello"},new int[]{1, 2,0})
    );
}
```

T21: Nomi [] contiene almeno 1 stringa vuota e Numeri[] contiene almeno 1 numero il cui cubo/quadrato supera il limite

```java
//T21: Nomi [] contiene almeno 1 stringa vuota e Numeri[] contiene almeno 1 numero il cui cubo/quadrato supera il limite
@ParameterizedTest
@MethodSource("testNameNullStringandonenumberthatexeedlimit")
void testNameNullStringandonenumberthatexeedlimit(int n, int m,String[] nomi,int[] numeri) {
    Main.effettuaOperazioni(n,m,nomi,numeri);

    assertTrue(Arrays.stream(nomi).anyMatch(nome -> nome == ""),
            message: "il nome e' vuoto: " + Arrays.toString(nomi));

    assertTrue(Arrays.stream(numeri).anyMatch(numero -> (numero % 2 == 0) && numero >= 10 || numero > 5 ),
            message: "numri contiene un numero il cui cubo/quadrato supera il limite: " + Arrays.toString(numeri));
}

private static Stream<Arguments> testNameNullStringandonenumberthatexeedlimit() {
    return Stream.of(
            Arguments.of( ...arguments: 1, 1,new String[]{"ciao", ""},new int[]{7,0}),
            Arguments.of( ...arguments: 2, 2,new String[]{"ciao", "","hello"},new int[]{11, 2,0})
    );
}
```

T22: Nomi [] contiene almeno 1 stringa vuota e Numeri[] contiene almeno un numero uguale a 1

```java
//T22: Nomi [] contiene almeno 1 stringa vuota e Numeri[] contiene almeno un numero uguale a 1
@ParameterizedTest
@MethodSource("testContainsEmptyStringandNumberOne")
void testContainsEmptyStringandNumberOne(int n, int m,String[] nomi,int[] numeri){
    Main.effettuaOperazioni(n,m,nomi,numeri);

    assertTrue(Arrays.stream(nomi).anyMatch(nome -> nome == ""),
            message: "nomi contiene almeno una stringa vuota: " + Arrays.toString(nomi));

    assertTrue(Arrays.stream(numeri).anyMatch(numero -> numero == 1),
            message: "numeri contiene almeno un numero 1 : " + Arrays.toString(numeri));

}
private static Stream<Arguments> testContainsEmptyStringandNumberOne() {
    return Stream.of(
            Arguments.of( ...arguments: 1, 1,new String[]{"ciao", ""},new int[]{1,3}),
            Arguments.of( ...arguments: 2, 2,new String[]{"ciao", "","hello"},new int[]{1,2,3})
    );
}
```

T23: Nomi [] contiene numeri e Numeri[] contiene n numeri positivi

```java
//T23: Nomi [] contiene numeri e Numeri[] contiene n numeri positivi
@ParameterizedTest
@MethodSource("testContainsNumberinString")
void testContainsNumberinString(int n, int m,String[] nomi,int[] numeri){
    Main.effettuaOperazioni(n,m,nomi,numeri);

    assertTrue(Arrays.stream(nomi).anyMatch(nome -> nome.matches( regex: ".*\\d.*")),
            message: "nomi contiene numeri: " + Arrays.toString(nomi));

    assertFalse(Arrays.stream(numeri).anyMatch(numero -> numero <= 0),
            message: "i numeri sono tutti positivi: " + Arrays.toString(numeri));

}


private static Stream<Arguments> testContainsNumberinString() {
    return Stream.of(
            Arguments.of( ...arguments: 1, 1,new String[]{"cia3o", "jl"},new int[]{1,3}),
            Arguments.of( ...arguments: 2, 2,new String[]{"cia3o", "halo","hello"},new int[]{1,2,3})
    );
}
```

T24: Nomi [] contiene numeri e Numeri[] contiene almeno un numero uguale a 0

```java
//T24: Nomi [] contiene numeri e Numeri[] contiene almeno un numero uguale a 0
@ParameterizedTest
@MethodSource("testContainsNumberinStringAndNumber0")
void testContainsNumberinStringAndNumber0(int n, int m,String[] nomi,int[] numeri) {
    Main.effettuaOperazioni(n,m,nomi,numeri);

    assertTrue(Arrays.stream(nomi).anyMatch(nome -> nome.matches( regex: ".*\\d.*")),
            message: "nomi contiene numeri: " + Arrays.toString(nomi));

    assertTrue(Arrays.stream(numeri).anyMatch(numero -> numero == 0),
            message: "Lo 0 non è presente nell'array numeri: " + Arrays.toString(numeri));

}

private static Stream<Arguments> testContainsNumberinStringAndNumber0() {
    return Stream.of(
            Arguments.of( ...arguments: 1, 1,new String[]{"hhh1h23", "khh8g"},new int[]{1,0}),
            Arguments.of( ...arguments: 2, 2,new String[]{"ci434ao", "l3434ola","3434miao"},new int[]{0,6,3})
    );
}
```

T25: Nomi [] contiene numeri e Numeri[] contiene almeno 1 numero il cui cubo/quadrato supera il limite

```
//T25: Nomi [] contiene numeri e Numeri[] contiene almeno 1 numero il cui cubo/quadrato supera il limite
@ParameterizedTest
@MethodSource("testContainsNumberinStringAndNumberThatexceedLimit")
void testContainsNumberinStringAndNumberThatexceedLimit(int n, int m,String[] nomi,int[] numeri) {
    Main.effettuaOperazioni(n,m,nomi,numeri);

    assertTrue(Arrays.stream(nomi).anyMatch(nome -> nome.matches( regex: ".*\\d.*")),
            message: "nomi contiene numeri: " + Arrays.toString(nomi));

    assertTrue(Arrays.stream(numeri).anyMatch(numero -> (numero % 2 == 0) && numero >= 10 || numero > 5 ),
            message: "numeri contiene un numero il cui cubo/quadrato supera il limite: " + Arrays.toString(numeri));

}

private static Stream<Arguments> testContainsNumberinStringAndNumberThatexceedLimit() {
    return Stream.of(
            Arguments.of( ...arguments: 1, 1,new String[]{"ciao4", "lola"},new int[]{7,1}),
            Arguments.of( ...arguments: 1, 1,new String[]{"ciao", "lola2"},new int[]{10,8}),
            Arguments.of( ...arguments: 2, 2,new String[]{"ciao3", "lola","lollo"},new int[]{11,1,1})
    );
}
```

T26 Nomi [] contiene numeri e Numeri[] contiene almeno un numero uguale a 1

```
//T26: Nomi [] contiene numeri e Numeri[] contiene almeno un numero uguale a 1
@ParameterizedTest
@MethodSource("testnumbersinstringandnumberone")
void testnumbersinstringandnumberone(int n, int m,String[] nomi,int[] numeri) {
    Main.effettuaOperazioni(n,m,nomi,numeri);

    assertTrue(Arrays.stream(nomi).anyMatch(nome -> nome.matches( regex: ".*\\d.*")),
            message: "nomi contiene numeri: " + Arrays.toString(nomi));

    assertTrue(Arrays.stream(numeri).anyMatch(numero -> numero == 1),
            message: "numeri contiene un numero il cui cubo/quadrato supera il limite: " + Arrays.toString(numeri));

}

private static Stream<Arguments> testnumbersinstringandnumberone() {
    return Stream.of(
            Arguments.of( ...arguments: 1, 1,new String[]{"ciao4", "lola"},new int[]{7,1}),
            Arguments.of( ...arguments: 1, 1,new String[]{"ciao", "lola3"},new int[]{10,1}),
            Arguments.of( ...arguments: 2, 2,new String[]{"ciao3", "lola","lollo"},new int[]{11,1,1})
    );
}
```

T27 Il numero di elementi contenuti negli array numeri e nomi non corrisponde alle variabili m ed n

```java
//T27 Il numero di elementi contenuti negli array numeri e nomi non corrisponde alle variabili m ed n
@ParameterizedTest
@MethodSource("testListLengthMismatch")
void testListLengthMismatch(int n, int m, String[] nomi, int[] numeri) {
    String result = Main.effettuaOperazioni(n, m, nomi, numeri);

    assertEquals( expected: "La lunghezza di uno dei due o entrambe le liste non corrisponde alla dimensione fornita in input", result);
}

private static Stream<Arguments> testListLengthMismatch() {
    return Stream.of(
            Arguments.of( ...arguments: 2, 2, new String[]{"ciao", "lola"}, new int[]{7, 2, 4}),
            Arguments.of( ...arguments: 1, 1, new String[]{"ciao", "lola", "lollo"}, new int[]{7, 2})
    );

}
```

# 7. Aumentare la suite di test con creatività ed esperienza

Per aumentare la suite di test abbiamo pensato alla realizzazione di nuovi test:
T28: Nomi[] contiene nomi uguali, stampa solo una volta il nome anche con CAPS differente.

```java
//T28 Nomi[] contiene nomi uguali, stampa solo una volta il nome anche con CAPS differente
    @ParameterizedTest
    @MethodSource("testnameContainsEqualNameAlthoughTheyStartWithDifferentCaps")
    void nameContainsEqualNameAlthoughTheyStartWithDifferentCaps(int n, int m,String[] nomi,int[] numeri) {
        Main.effettuaOperazioni(n,m,nomi,numeri);

        boolean containsSameNamesWithDifferentCaps = Arrays.stream(nomi)
                .anyMatch(nome -> Arrays.stream(nomi)
                        .anyMatch(otherNome -> !nome.equals(otherNome) && nome.equalsIgnoreCase(otherNome)));

        assertTrue(containsSameNamesWithDifferentCaps,
                message: "nomi contiene nomi uguali ma con caps differente: " + Arrays.toString(nomi));

    }
    1 usage
    private static Stream<Arguments> testnameContainsEqualNameAlthoughTheyStartWithDifferentCaps() {
        return Stream.of(
                Arguments.of( ...arguments: 4, 4,new String[]{"Federico", "federico"},new int[]{7,1})
        );
    }
```

T29: Numeri[] contiene numeri uguali, stampa solo una volta il numero.

```java
//T29 Numeri[] contiene numeri uguali, stampa solo una volta il numero.
@ParameterizedTest
@MethodSource("testNumbersHaveDuplicatedNumbers")
void NumbersHaveDuplicatedNumbers(int n, int m,String[] nomi,int[] numeri) {
    Main.effettuaOperazioni(n,m,nomi,numeri);

    boolean hasDuplicateNumbers = Arrays.stream(numeri)
            .anyMatch(numero -> Arrays.stream(numeri)
                    .filter(num -> num == numero)
                    .count() >= 2);

    assertTrue(hasDuplicateNumbers,  message: "numeri contiene almeno due numeri uguali: " + Arrays.toString(numeri));

}
1 usage
private static Stream<Arguments> testNumbersHaveDuplicatedNumbers() {
    return Stream.of(
            Arguments.of( ...arguments: 4, 4,new String[]{"Federico", "federico"},new int[]{7,7})
    );
}
```

# HOMEWORK 2

Al termine del primo Homework, abbiamo effettuato i test con coverage utilizzando il tool jacoco, e abbiamo riscontrato che la classe interessata dai test, ovvero effettuaOperazioni(), ha raggiunto una percentuale di line coverage pari al 100%. Ecco qui sotto un resoconto generale

## Esercitazione1

| Element | Missed Instructions | Cov. | Missed Branches | Cov. | Missed | Cxty | Missed | Lines | Missed | Methods | Missed | Classes |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| org.example | | 83% | | 100% | 2 | 17 | 8 | 43 | 2 | 3 | 0 | 1 |
| Total | 31 of 190 | 83% | 0 of 28 | 100% | 2 | 17 | 8 | 43 | 2 | 3 | 0 | 1 |

## Main

| Element | Missed Instructions | Cov. | Missed Branches | Cov. | Missed | Cxty | Missed | Lines | Missed | Methods |
|---|---|---|---|---|---|---|---|---|---|---|
| main(String[]) | | 0% | | n/a | 1 | 1 | 7 | 7 | 1 | 1 |
| Main() | | 0% | | n/a | 1 | 1 | 1 | 1 | 1 | 1 |
| effettuaOperazioni(int, int, String[], int[]) | | 100% | | 100% | 0 | 15 | 0 | 35 | 0 | 1 |
| Total | 31 of 190 | 83% | 0 of 28 | 100% | 2 | 17 | 8 | 43 | 2 | 3 |

```java
public static String effettuaOperazioni(int n, int m, String[] nomi, int[] numeri) throws RuntimeException {

        if (n != m)
        {
            return "Il numero di elementi inseriti non corrisponde";
        }
        else
        {
            if (n<=0)
            {
                return("Non e' possibile inserire una grandezza pari o minore di 0");
            }
            else {

                if(nomi.length != m || numeri.length != n)
                {
                    return "La lunghezza di uno dei due o entrambe le liste non corrisponde alla dimensione fornita in input";
                }

                for (int i = 0; i <= n-1; i++)
                {
                    if (numeri[i] < 0)
                    {
                        return ("Non e' possibile inserire un numero negativo");
                    }
                    if (numeri[i] > 99 )
                    {
                        return ("Il numero inserito e' troppo grande (offpoint) ");
                    }
                    if (nomi[i] == null) {
                        return ("Non e' possibile inserire stringhe nulle");
                    }
                }
            }

            // Calcola il quadrato dei numeri pari e il cubo dei numeri dispari
            List<Integer> quadrati = new ArrayList<>();
            List<Integer> cubi = new ArrayList<>();

            for (int i = 0; i <= n-1; i++) {
                if (numeri[i] % 2 == 0)
                {
                    int quadrato = numeri[i] * numeri[i];
                    if (quadrato < 100)
                    {
                        quadrati.add(quadrato);
```

```
                    }
                }
                else
                {
                    int cubo = numeri[i] * numeri[i] * numeri[i];
                    if (cubo < 200)
                    {
                        cubi.add(cubo);
                    }
                }
            }

            // Unisci i quadrati e i cubi in un unico array e ordina il tutto
            List<Integer> numeriOrdinati = new ArrayList<>();
            numeriOrdinati.addAll(quadrati);
            numeriOrdinati.addAll(cubi);
            Collections.sort(numeriOrdinati);


            Arrays.sort(nomi);

            // Stampa i numeri ordinati
            for (int numero : numeriOrdinati) {
                System.out.println(numero);
            }

            // Stampa i nomi ordinati
            for (String nome : nomi) {
                System.out.println(nome);
            }

        }
        return "Codice terminato con successo";
    }

}
```

# Task 1

Per completare i test Black Box , abbiamo proceduto con lo svolgimento degli structural test , seguendo il criterio del **Rule of Thumb**:

Non essendo in possesso di condizioni particolarmente complesse, abbiamo effettuato la branch coverage su ogni istruzione presente nella classe interessata:

```
if (n != m) {
    return "Il numero di numeri e nomi inseriti non corrisponde";
}


else {
```

- T4 n True, m True
- T1 n False, m False

```
if (n==0) {
    return("Non e' possibile creare array di dimensione 0");
}
```

- T1 n True
- T11 n False

```
if (numeri[i] < 0)
{
    return ("Non e' possibile inserire un numero negativo");
}
```

- T7 numeri[] True
- T11 numeri[] False

  i test T11 , T12…T26 appartengono all stessa partizione, per comoditá abbiamo deciso di utilizzare semplicemente T11 nelle espressioni

```
if (numeri[i] > 99 )
{
    return ("Il numero inserito e' troppo grande (offpoint) ");
}
```

- T9 numeri[] True
- T11 numeri[] False

Per questa particolare condizione abbiamo invece utilizzato lla Condition + Branch Coverage , in quanto ci troviamo d'avanti ad un espressione piú complessa:

```
if(nomi.length != m || numeri.length != n)
{
    return "La lunghezza di uno dei due o entrambe le liste non corrisponde alla dimensione fornita in input";
}
```

- T27: nomi.lenght **False** , numeri.lenght **False**. 2/4 condizioni coperte su 1/2 branch totali.
- T11: nomi.lenght **True** , numeri.lenght **True**. 2/4 condizioni coperte su 1/2 branch totali.

$$\text{c+b coverage} = \frac{\text{branches covered} + \text{conditions covered}}{\text{number of branches} + \text{number of conditions}} \times 100\% \quad \Rightarrow \frac{2+4}{2+4} = 100\%$$

```
if (nomi[i] == null) {
    return ("Non e' possibile inserire stringhe nulle");
}
```

- T10 nomi[] True
- T11 nomi[] False

```
for (int i = 0; i <= n; i++) {
    if (numeri[i] % 2 == 0) {
        int quadrato = numeri[i] * numeri[i];
        if (quadrato < 100) {
            quadrati.add(quadrato);
        }
    } else {
        int cubo = numeri[i] * numeri[i] * numeri[i];
        if (cubo < 200) {
            cubi.add(cubo);
        }
    }
}
```

- T11 condizione ciclo for True

```
if (numeri[i] % 2 == 0) {
    int quadrato = numeri[i] * numeri[i];
    if (quadrato < 100) {
        quadrati.add(quadrato);
    }
} else {
```

- T9 numeri[] False
- T11 numeri[] True

```
    if (quadrato < 100) {
        quadrati.add(quadrato);
    }
```

- T11 numeri[] True
- T17 numeri[] False

```
} else {
    int cubo = numeri[i] * numeri[i] * numeri[i];
    if (cubo < 200) {
        cubi.add(cubo);
    }
}
```

- T9 else False
- T11 else True

# Task 2

Per il secondo Homework abbiamo utilizzato il codice dei nostri colleghi del team “DoubleLF” eseguendo lo Specification Based Testing secondo i 7 punti:

## 1. Comprendere i requisiti

Il codice fornito implementa l'algoritmo di ordinamento "Merge Sort" per un array di interi. L'obiettivo dell'algoritmo è ordinare l'array in modo crescente.

Il codice è composto principalmente dalla classe MergeSort, che contiene un metodo statico chiamato mergeSort. Questo metodo prende in input tre array di interi: arr, l e r. L'array arr rappresenta l'array da ordinare, mentre l e r rappresentano due sottoarray che verranno uniti durante il processo di ordinamento.

Il codice gestisce anche alcuni casi speciali, come gli array nulli o vuoti, fornendo un messaggio di errore appropriato. Inoltre, fornisce metodi di utilità per stampare gli array e calcolare le suddivisioni necessarie durante il processo di ordinamento.

## 2. Esplora cosa fa il programma per vari input

```
> Task :Homework2.main()
Given Array
32 3 6 7

Sorted array
3 6 7 32

BUILD SUCCESSFUL in 167ms
```

```
> Task :Homework2.main()
Given Array
1 1

Sorted array
1 1

BUILD SUCCESSFUL in 164ms
```

```
> Task :Homework2.main()
Given Array
1

Sorted array
1

BUILD SUCCESSFUL in 175ms
```

```
> Task :Homework2.main()
Given Array


Sorted array
```

```
> Task :Homework2.main()
Given Array
0

Sorted array
0

BUILD SUCCESSFUL in 162ms
```

```
> Task :Homework2.main()
Given Array
Null Array

Sorted array
Null Array

BUILD SUCCESSFUL in 149ms
```

```
> Task :Homework2.main()
Given Array
1 2 3

Sorted array
1 2 3
```

```
> Task :Homework2.main()
Given Array
3 2 1

Sorted array
1 2 3

BUILD SUCCESSFUL in 158ms
```

```
> Task :Homework2.main()
Given Array
-3 2 -1

Sorted array
-3 -1 2

BUILD SUCCESSFUL in 143ms
```

```
> Task :Homework2.main()
Given Array
-1 -2 -3

Sorted array
-3 -2 -1

BUILD SUCCESSFUL in 151ms
```

```
C:\Users\giuli\Desktop\esameIntegrazioneTest\src\main\java\org\example\Homework2.java:107: error: incompatible types:
        int[] arr = {0.2,2,1};
                    ^
```

```
> Task :Homework2.main()
Given Array
1 2 4 12 11 9

Sorted array
1 2 4 9 11 12

BUILD SUCCESSFUL in 163ms
```

```
> Task :Homework2.main()
Given Array
12 11 9 1 2 4

Sorted array
1 2 4 9 11 12

BUILD SUCCESSFUL in 158ms
```

```
Given Array
2 3 1 2 5 6 6 7

Sorted array
1 2 2 3 5 6 6 7

BUILD SUCCESSFUL in 154ms
```

# 3. Esplora input, output e identifica le partizioni

## a) Ingressi Individuali:

- **Array arr:**
  - **arr nullo**
  - **arr di lunghezza = 1**
  - **arr di lunghezza > 1**
  - **arr di lunghezza = 0**

- **sottoArray l:**
  - **arr nullo**
  - **arr di lunghezza = 1**
  - **arr di lunghezza > 1**

**arr di lunghezza = 0**

- **sottoArray r:**
  **arr nullo**
  **arr di lunghezza = 1**
  **arr di lunghezza > 1**
  **arr di lunghezza = 0**

### b) Combinazioni di input:

**arr nullo, r nullo, l nullo**
**arr di dimensione = 1 , r di dimensione =1 , l di dimensione 0**
**arr di dimensione >1 dispari , r piú grande di l**
**arr di dimensione >1 pari , r della stessa dimensione di l**
**arr di dimensione 0, r di dimensione 0, l di dimensione 0**

### c) Classi di output:

**arr nullo**
**arr con 0 elementi**
**arr con 1 elemento**
**arr con n elementi di dimensione > 1 uguale a quello di partenza**
**arr con n elementi di dimensione > 1 diverso da quello di partenza**

# 4. Identifica i casi limite

I casi limite che abbiamo individuato , si concentrano nella partizione della condizione arr.length < 2 , in quanto:

- ***off point*** è il valore piú vicino al limite che rende la condizione falsa, ed in questo caso è proprio il 2.
- ***on point*** è il valore piú vicino al limite che rende la condizione vera, ed in questo caso è proprio 1
- ***in point*** è il valore che rende la condizione vera , ovvero 1.
- ***out point*** è il valore che rende la condizione false, ovvero 2.

# 5. Elaborare casi di test

**Testiamo innanzitutto i casi limite:**

## Casi Eccezionali:

- T1 Arr`[]` nullo
- T2 `l[]` nullo
- T3 r`[]` nullo
- T4 l`[]` vuoto
- T5 r`[]` vuoto
- T6 Arr`[]` vuoto
- T7 Dimensione di l  maggiore a r
- T8 Arr giá ordinato in senso crescente

## Con Arr`[]` di dimensione 1  (Boundary case)

- T9 Arr`[]` con solo 1 numero, r vuoto l vuoto (l ed r non vengono riempiti)
- T10 Arr`[]` con solo 1 numero, r uguale ad arr, l vuoto  (caso normale)
- T11 Arr`[]` con solo 1 numero, r non vuoto l non vuoto (l ed r vengono riempiti male)

## Caso Arr.lenght >1 pari:

- T12 Arr`[]` con lunghezza pari, l della stessa dimensione di r  (caso normale)
- T13 Arr`[]` con lunghezza pari, l vuoto

## Caso Arr.lenght >1 dispari:

- T14 Arr`[]` con lunghezza dispari, l piú piccolo di r (caso normale)

# 6. Automatizzare i casi di test

- T1 Arr`[]` nullo

```
//T1
@Test
@DisplayName("arrIsNullorEmpty")
void arrIsNull() { assertNull(Homework2.mergeSort( arr: null, new int[]{1}, new int[]{2,3})); }
```

- T2 `l[]` nullo

```
//T2
@Test
@DisplayName("l is null")
void lIsNull() { assertNull(Homework2.mergeSort(new int[]{3,5,7},  l: null, new int[]{5,7})); }
```

- T3 r`[]` nullo

```
//T3
@Test
@DisplayName("r is null")
void rIsNull() { assertNull(Homework2.mergeSort(new int[]{3,5,7},new int[]{5,7}, r: null)); }
```

- T4 l`[]` vuoto

```
//T4
@Test
@DisplayName("lisEmpty")
void lIsEmpty()
{
    assertArrayEquals(new int[]{3,5,7},Homework2.mergeSort(new int[]{3,5,7}, new int[]{}, new int[]{5,7}));
}
```

- T5 r`[]` vuoto

```
//T5
@Test
@DisplayName("lisEmpty")
void rIsEmpty()
{
    assertArrayEquals(new int[]{3,5,7},Homework2.mergeSort(new int[]{3,5,7},new int[]{},new int[]{5,7}));
}
```

- T6 Arr`[]` vuoto

```
//T6
@Test
@DisplayName("ArrisEmpty")
void ArrIsEmpty()
{
    assertArrayEquals(new int[]{},Homework2.mergeSort(new int[]{}, new int[]{1}, new int[]{2, 3}));
}
```

- T7 Dimensione di l  maggiore a r

```
//T7
@Test
@MethodSource("Test7")
void LGreaterThanR()
{
    assertArrayEquals(new int[]{3,5,7},Homework2.mergeSort(new int[]{3,5,7},new int[]{3},new int[]{}));
}
```

- T8 Arr giá ordinato in senso crescente

```
//T8
@Test
@DisplayName("ArrayOrderAsc")
void arrOrdAsc()
{
    assertArrayEquals(new int[]{3,5,7},Homework2.mergeSort(new int[]{3,5,7},new int[]{3},new int[]{5,7}));
}
```

- T9 Arr`[]` con solo 1 numero, r vuoto l vuoto (l ed r non vengono riempiti)

```java
//T9
@Test
@DisplayName("ArrayDim1LEmptyREmpty")
void ArrayDim1LEmptyREmpty()
{
    assertArrayEquals(new int[]{3},Homework2.mergeSort(new int[]{3},new int[]{},new int[]{}));
}
```

- T10 Arr`[]` con solo 1 numero, r uguale ad arr l vuoto  (caso normale)

```java
//T10
@Test
@DisplayName("ArrayDim1RDimArrLEmpty")
void ArrayDim1RDimArrLEmpty()
{
    assertArrayEquals(new int[]{3},Homework2.mergeSort(new int[]{3},new int[]{},new int[]{3}));
}
```

- T11 Arr`[]` con solo 1 numero, r non vuoto l non vuoto (l ed r vengono riempiti male)

```java
//T11
@Test
@DisplayName("ArrayDim1RNotEmptyLNotEmpty")
void ArrayDim1RNotEmptyLNotEmpty()
{
    assertArrayEquals(new int[]{3},Homework2.mergeSort(new int[]{3},new int[]{4},new int[]{3}));
}
```

- T12 Arr`[]` con lunghezza pari, l della stessa dimensione di r  (caso normale)

```java
//T12
@Test
@DisplayName("ArrayEvenLEqualsR")
void ArrayEvenLEqualsR()
{
    assertArrayEquals(new int[]{2,3},Homework2.mergeSort(new int[]{3,2},new int[]{3},new int[]{2}));
}
```

- T13 Arr`[]` con lunghezza pari, l vuoto

```java
//T13
@Test
@DisplayName("ArrayEvenLEmpty")
void ArrayEvenLEmpty()
{
    assertArrayEquals(new int[]{3,5,7,2},Homework2.mergeSort(new int[]{3,5,7,2}, new int[]{}, new int[]{4,7,2}));
}
```

- T14 Arr`[]` con lunghezza dispari, l piú piccolo di r (caso normale)

```java
//T14
@Test
@DisplayName("ArrayOddLMinorThenL")
void ArrayOddLMinorThenL()
{
    assertArrayEquals(new int[]{3,5,7},Homework2.mergeSort(new int[]{3,5,7}, new int[]{3}, new int[]{5,7}));
}
```

# 7. Aumentare la suite di test con creatività ed esperienza

In ottica Black Box , abbiamo incrementato la suit di test , con creatività ed esperienza , in modo da garantire ulteriormente il 100% di coverage.

- T15 Arr`[]` già ordinato in ordine decrescente

```
//T15 Arr[] già ordinato in ordine decrescente
@Test
@DisplayName("ArraySortedInDescendingOrder")
@MethodSource("ArraySortedInDescendingOrderArguments")
void ArraySortedInDescendingOrder() {
    assertArrayEquals(new int[] {1,3,5,7,9}, Homework2.mergeSort(new int[]{9, 7, 5, 3, 1}, new int[]{9, 7}, new int[]{5, 3, 1}));
}
```

- T16 l`[]` ed `r[]` già ordinati in ordine crescente , ma complessivamente non disposti in ordine crescente.

```
//T16 l[] ed r[] già ordinati in ordine crescente , ma complessivamente non disposti in ordine crescente.
@Test
@DisplayName("LandRalreadyOrderedButArrIsNotOrdered")
void LandRalreadyOrderedButArrIsNotOrdered()
{
    assertArrayEquals(new int[]{5,6,8,9},Homework2.mergeSort(new int[]{8,9,5,6}, new int[]{8,9}, new int[]{5,6}));
}
```

# HOMEWORK 3

Per la realizzazione del terzo Homework , abbiamo scelto come traccia un codice che permette di effettuare la validazione di una password, tenendo conto dei seguenti vincoli per definire una password valida:

1. La password deve essere compresa tra minimo 8 e massimo 20 caratteri.
2. La password deve contenere almeno un carattere speciale e un numero
3. La password deve contenere almeno un carattere Maiuscolo ed uno minuscola.

Il codice restituirá 1 nel caso in cui la password sia valida, -1 nel caso in cui la lunghezza della password non venga rispettata e 0 nel caso in cui la password inserita non rispetta le regole sopra definite.

## Utilizzo della libreria JQWIK

Per i test di tipo property-based abbiamo utilizzato la libreria Jqwik. Essa fornisce un framework per la scrittura e l'esecuzione di test basati sulle proprietà delle funzioni o dei metodi che si desidera testare.

I test property-based sono un approccio per testare il software in cui si definiscono proprietà generali che il sistema o le funzioni devono soddisfare, invece di specificare casi di test

specifici. Queste proprietà generali vengono poi verificate utilizzando un insieme di valori generati casualmente.
Jqwik semplifica la scrittura dei test property-based in Java fornendo un'API intuitiva e potente. Offre funzionalità come la generazione automatica dei dati di test, la gestione dei parametri e la generazione dei report delle statistiche.

## Distinzione delle Partizioni

Per iniziare i property based test, abbiamo individuato le partizioni presenti in questo codice , giungendo alla seguente suddivisione:

- **Successo**: Password accettata.
- **Fallimento**: Password non accettata.
- **Fuori dal Range** :Password di dimensioni non valide.

## Spiegazione metodi di test

La firma dei seguenti metodi presenta le seguenti annotazioni:

- **@Property** : suggerisce che il metodo è utilizzato per testare una proprietà specifica o un comportamento atteso di un sistema o di una funzione.
- **@Report(Reporting.GENERATED)**: indica che il metodo genererà un report sul risultato del test.
- **@StatisticsReport:(format = Histogram.class)**: indica che verrà generato un report statistico utilizzando il formato di visualizzazione di un istogramma.

Il primo test, *testAcceptedPassword ,* é un metodo che testa che tutte le password generate rispettino i requisiti stabiliti.
Inoltre il test sfrutta un'oggetto Arbitrary<String> stringheAccettate() per generare password che soddisfino i requisiti del programma.

```
//genera e testa situazioni accettate

@Property(tries = 10)
@Report(Reporting.GENERATED)
@StatisticsReport(format = Histogram.class)
void testAcceptedPassword(@ForAll("stringheAccettate") String Password){
    Integer result = Homework3.validatePassword(Password);
    Statistics.collect( ...values: "Password Accettate : Cod." ,result);
    assertEquals(result , actual: 1);
}
```

```
no usages
@Provide
Arbitrary<String> stringheAccettate() {
    return Arbitraries.strings().withCharRange('!', '~').filter(s -> s.length() >= 8 && s.length() <= 20 &&
            s.matches( regex: ".*[A-Z].*") && s.matches( regex: ".*[a-z].*") && s.matches( regex: ".*\\d.*")
            && s.matches( regex: ".*[!@#$%^&*()].*") && s.matches( regex: ".*[0-9].*"));
}
```

Report e password generate:

```
timestamp = 2023-06-26T10:02:45.295837800, [TestHomework3:testAcceptedPassword] (10) statistics =
       # |                        label | count |
    -----|------------------------------|-------|-----------------------------------------------
       0 | Password Accettate : Cod. 1 |    10 | ##########

timestamp = 2023-06-26T10:02:45.300336800, TestHomework3:testAcceptedPassword =
                              |----------------------jqwik----------------------
tries = 10                    | # of calls to property
checks = 10                   | # of not rejected calls
generation = RANDOMIZED       | parameters are randomly generated
after-failure = SAMPLE_FIRST  | try previously failed sample, then previous seed
when-fixed-seed = ALLOW       | fixing the random seed is allowed
edge-cases#mode = MIXIN       | edge cases are mixed in
edge-cases#total = 0          | # of all combined edge cases
edge-cases#tried = 0          | # of edge cases tried in current run
seed = -1770822539576280728   | random seed to reproduce generated values


BUILD SUCCESSFUL in 1s
```

```
timestamp = 2023-06-26T09:53:49.103855500, generated =
  arg0: "0w}@ZVP-"


timestamp = 2023-06-26T09:53:49.103855500, generated =
  arg0: ":6bZ!Hb>/1"


timestamp = 2023-06-26T09:53:49.105362100, generated =
  arg0: "(QG1E9Fp"


timestamp = 2023-06-26T09:53:49.107786500, generated =
  arg0: "}_91dVtj!"


timestamp = 2023-06-26T09:53:49.108786900, generated =
  arg0: "&lrxzq;C7"


timestamp = 2023-06-26T09:53:49.108786900, generated =
  arg0: "-C/4z)Ji"


timestamp = 2023-06-26T09:53:49.109783500, generated =
  arg0: "Fs=}H9!^V@"
```

Il secondo test, *testNotAcceptedPassword* , é un metodo che testa che tutte le password generate non rispettino i requisiti stabiliti.
Il test sfrutta un'oggetto Arbitrary<String> stringheNonAccettate() per generare password che non soddisfino i requisiti del programma.

```java
//genera e testa situazioni non accettate

@Property(tries = 10)
@Report(Reporting.GENERATED)
@StatisticsReport(format = Histogram.class)
void testNotAcceptedPassword(@ForAll("stringheNonAccettate") String Password){
    Integer result = Homework3.validatePassword(Password);
    Statistics.collect( ...values: "Password Non accettate : Cod." ,result);
    assertEquals(result , actual: 0);
}
```

```java
@Provide
Arbitrary<String> stringheNonAccettate() {
    return Arbitraries.strings().filter(s -> s.length() >= 8 && s.length() <= 20 && s.matches( regex: ".*[a-z].*"));
}
```

Report e password generate:

```
timestamp = 2023-06-26T10:05:14.303497600, [TestHomework3:testNotAcceptedPassword] (10) statistics =
       # |                          label | count |
    -----|--------------------------------|-------|-----------------------------------------------
       0 | Password Non accettate : Cod. 0 |    10 | ##########

timestamp = 2023-06-26T10:05:14.308327600, TestHomework3:testNotAcceptedPassword =
                              |----------------------jqwik----------------------
tries = 10                    | # of calls to property
checks = 10                   | # of not rejected calls
generation = RANDOMIZED       | parameters are randomly generated
after-failure = SAMPLE_FIRST  | try previously failed sample, then previous seed
when-fixed-seed = ALLOW       | fixing the random seed is allowed
edge-cases#mode = MIXIN       | edge cases are mixed in
edge-cases#total = 0          | # of all combined edge cases
edge-cases#tried = 0          | # of edge cases tried in current run
seed = -7337083466420989743   | random seed to reproduce generated values


BUILD SUCCESSFUL in 1s
```

```
timestamp = 2023-06-26T09:54:46.614051200, generated =
  arg0: "?????h??"


timestamp = 2023-06-26T09:54:46.690948100, generated =
  arg0: "d?????????"


timestamp = 2023-06-26T09:54:46.709242900, generated =
  arg0: "?????a???"


timestamp = 2023-06-26T09:54:46.717419800, generated =
  arg0: "??j?????"


timestamp = 2023-06-26T09:54:46.730647, generated =
  arg0: "?????x??"


timestamp = 2023-06-26T09:54:46.732642100, generated =
  arg0: "????w????"


timestamp = 2023-06-26T09:54:46.739132700, generated =
  arg0: "???????v"
```

Il terzo test, *testOutOfRangePassword,* é un metodo che testa che tutte le password generate abbiano una lunghezza non conforme con i requisiti.
Infine anche per questo ultimo caso, sfruttiamo un'oggetto Arbitrary<String> stringheOutOfRange() per generare password di lunghezza diversa dai limiti consentiti.

```java
//genera e testa situazioni Out of Range

@Property(tries = 10)
@Report(Reporting.GENERATED)
@StatisticsReport(format = Histogram.class)
void testOutOfRangePassword(@ForAll("stringheOutOfRange") String Password){
    Integer result = Homework3.validatePassword(Password);
    Statistics.collect( ...values: "Password Out of Range : Cod." ,result);
    assertEquals(result,  actual: -1);
}
```

```java
@Provide
Arbitrary<String> stringheOutOfRange() {
    return Arbitraries.strings().withCharRange('!', '~').filter(s -> s.length() > 20 &&
            s.matches( regex: ".*[A-Z].*") && s.matches( regex: ".*[a-z].*") && s.matches( regex: ".*\\d.*")
            && s.matches( regex: ".*[!@#$%^&*()].*") && s.matches( regex: ".*[0-9].*"));
}
```

Report e password generate:

```
timestamp = 2023-06-26T10:06:32.817354600, [TestHomework3:testOutOfRangePassword] (10) statistics =
       # |                           label | count |
    -----|---------------------------------|-------|---------------------------------------------
       0 | Password Out of Range : Cod. -1 |    10 | ##########

timestamp = 2023-06-26T10:06:32.823339900, TestHomework3:testOutOfRangePassword =
                              |----------------------jqwik----------------------
tries = 10                    | # of calls to property
checks = 10                   | # of not rejected calls
generation = RANDOMIZED       | parameters are randomly generated
after-failure = SAMPLE_FIRST  | try previously failed sample, then previous seed
when-fixed-seed = ALLOW       | fixing the random seed is allowed
edge-cases#mode = MIXIN       | edge cases are mixed in
edge-cases#total = 0          | # of all combined edge cases
edge-cases#tried = 0          | # of edge cases tried in current run
seed = -3648536655765359252   | random seed to reproduce generated values


BUILD SUCCESSFUL in 1s
```

```
timestamp = 2023-06-26T09:55:55.747510700, generated =
  arg0: "`7g%<J}"


timestamp = 2023-06-26T09:55:55.749507, generated =
  arg0: "J66rX&"


timestamp = 2023-06-26T09:55:55.750504200, generated =
  arg0: "1*Kq"


timestamp = 2023-06-26T09:55:55.752500100, generated =
  arg0: "k!9{bE"


timestamp = 2023-06-26T09:55:55.755492300, generated =
  arg0: "CY3eI!"


timestamp = 2023-06-26T09:55:55.757917500, generated =
  arg0: "b5^0Of""
```