

UNIVERSITÀ DEGLI STUDI DI BARI ALDO MORO

Dipartimento di Informatica

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

# DOCUMENTAZIONE PROGETTO

Corso di Integrazione e Test a.a. 2022/2023

Prof.ssa **Azzurra Ragone**

[f.lacarbonara3@studenti.uniba.it](mailto:f.lacarbonara3@studenti.uniba.it)

f.lisco7@studenti.uniba.it

Team DoubleLF

* **Fabio Lacarbonara 737518**
* **Francesco Lisco 735640**

Sommario

[DOCUMENTAZIONE PROGETTO 1](#_Toc138867419)

[Homework n°1 3](#_Toc138867420)

[Testing Workflow For Specification-Based Testing (Merge Sort) 3](#_Toc138867421)

[**1)** Comprensione dei requisiti: 3](#_Toc138867422)

[**2)** Esplora cosa fa il programma, dati vari input: 3](#_Toc138867423)

[**3)** Esplora input, output e identifica le partizioni 5](#_Toc138867424)

[**4)** Identifica i casi limite (boundary cases) 5](#_Toc138867425)

[**5)** Elaborare casi di test 6](#_Toc138867426)

[**6)** Automatizza i casi di test 7](#_Toc138867427)

[**7)** Aumenta i casi di test della suite con creatività ed esperienza 9](#_Toc138867428)

[Homework n°2 10](#_Toc138867429)

[Structural Testing and Code Coverage (Task 1) 10](#_Toc138867430)

[Structural Testing and Code Coverage (Task 2) 15](#_Toc138867431)

# Homework n°1

## Testing Workflow For Specification-Based Testing (Merge Sort)

### Comprensione dei requisiti:

* 1. L’obiettivo del metodo è quello di riordinare una lista di numeri interi fornita in input, in ordine crescente.
  2. Il programma riceve 3 parametri in input:
     1. **arr**, che rappresenta la lista di numeri da riordinare.
     2. **l**, che rappresenta la lista di numeri dal primo elemento fino al punto medio di “arr”.
     3. **r**, che rappresenta la lista di numeri dal punto medio fino all’ultimo elemento di “arr”.
  3. Il programma restituisce in output la lista di numeri “arr” ordinata.

### Esplora cosa fa il programma, dati vari input:

* 1. Immagine che contiene testo

     Descrizione generata automaticamente
  2. Immagine che contiene testo, dispositivo, metro, chiusura

     Descrizione generata automaticamente
  3. Immagine che contiene testo

     Descrizione generata automaticamente
  4. Immagine che contiene testo

     Descrizione generata automaticamente
  5. Immagine che contiene testo

     Descrizione generata automaticamente
  6. Immagine che contiene testo

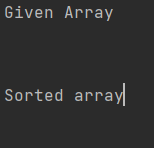
     Descrizione generata automaticamente
  7. Immagine che contiene testo

     Descrizione generata automaticamente
  8. Immagine che contiene testo

     Descrizione generata automaticamente
  9. Immagine che contiene testo

     Descrizione generata automaticamente
  10. Immagine che contiene testo

      Descrizione generata automaticamente
  11. Immagine che contiene testo, Carattere, schermata, design

      Descrizione generata automaticamente
  12. 

### Esplora input, output e identifica le partizioni

* 1. Input individuali:

Parametro r:

1. Null
2. Array vuoto
3. Array di lunghezza = 1
4. Array di lunghezza > 1

Parametro l:

1. Null
2. Array vuoto
3. Array di lunghezza = 1
4. Array di lunghezza > 1

Parametro arr:

1. Null
2. Array vuoto
3. Array di lunghezza = 1
4. Array di lunghezza > 1
   1. Combinazioni input:
5. arr, l e r sono null
6. arr, l e r sono vuoti
7. arr è di lunghezza 1, l è vuoto ed r ha lunghezza 1
8. arr è di lunghezza > 1 e pari, l ed r hanno la stessa lunghezza
9. arr è di lunghezza > 1 e dispari, l è di lunghezza < di r
   1. Classi di Output:

Array di numeri interi (Output)

1. Null
2. Array vuoto
3. Array di lunghezza = 1
4. Array di lunghezza > 1
5. Identifica i casi limite (boundary cases)Analizzando il codice abbiamo individuato la presenza di boundary cases in corrispondenza  
   della seguente condizione: if (arr.length <= 1)  
   1. **1** è l’on point: il punto che si trova al limite (presente all’interno della condizione)
   2. **2** è l’off point: il punto più vicino al limite che appartiene alla partizione successiva (arr.length > 1) e poiché l’on point rende la condizione vera, l’off point sarà il primo punto che la rende falsa.
   3. **1** è l’in point: il punto che rende la condizione vera
   4. **2** è l’out point: il punto che rende la condizione falsa.
6. Elaborare casi di test  
   Abbiamo deciso di testare, come casi i seguenti:  
     
   ***Casi di test eccezionali:***
7. arr è null
8. arr è vuoto
9. l è null
10. l è vuoto
11. r è null
12. r è vuoto
13. l è maggiore di r

***Casi di test per arr di lunghezza = 1 (Boundary test)***

1. arr ha un singolo elemento, l è vuoto ed r è uguale ad arr
2. arr ha un singolo elemento, l non è vuoto
3. arr ha un singolo elemento, r non è uguale ad arr
4. arr ha un singolo elemento, l non è vuoto ed r non è uguale ad arr

***Casi di test per arr di lunghezza > 1:***

1. arr ha dimensioni pari maggiori di 1, l è la prima metà ed r è la seconda metà
2. arr ha dimensioni dispari maggiori di 1, l contiene tutti i numeri prima del punto medio ed r contiene tutti i numeri dal punto medio in poi.

### Automatizza i casi di test

***T1, T2, T3, T4, T5, T6, T7:***

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

***T8, T9, T10, T11:***

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

Durante l’elaborazione del test T9 in Junit , ci siamo resi conto della presenza di un controllo che non rispettava i requisiti: 

poiché per “riordinare” un array composto da un solo elemento dovrebbe essere necessario restituire in output l’array di partenza, di conseguenza non basta che r sia diverso da zero ma sia uguale ad arr. Di seguito la modifica nel codice:



***T12, T13:***

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

1. Aumenta i casi di test della suite con creatività ed esperienza

T14. arr ha tutti gli elementi (più di uno) in ordine crescente

T15. arr ha più di un elemento, un elemento non è posizionato in ordine crescente

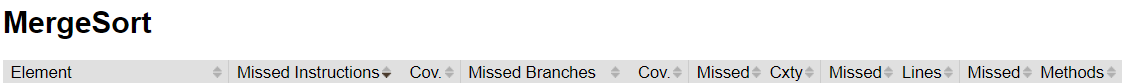
Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamenteT16. arr sia con numeri positivi che negativi  
T17. arr con soli valori negativi  
T18. arr con elementi duplicati  
T19. arr con elementi in ordine decrescente  
T20. arr con elemento “maxvalue”  
T21. arr con elemento “minvalue”  
T22. arr sia con elemento “maxvalue” che “minvalue”

# Homework n°2

## Structural Testing and Code Coverage (Task 1)

Immagine che contiene testo, schermata, Carattere, design

Descrizione generata automaticamentePer completare la suite di test black-box dobbiamo effettuare gli structural testing e l’analisi di copertura del codice. Per quando riguarda il code coverage, ci siamo serviti del tool Jacoco per verificare quanto il nostro codice fosse coperto dai test.

Come si può evincere dall’immagine, **la line coverage** equivale al **100%** in quanto ogni riga del codice è stata testata almeno una volta, comprese le istruzioni come: if, while, for, ecc...

Per iniziare gli structural testing abbiamo deciso di utilizzare la **Rule of Thumb**:

1. Iniziare con il **Branch Coverage**
2. Per espressioni più complicate decidere se utilizzare la **Condition + Branch** Coverage oppure **l’MC/DC**.

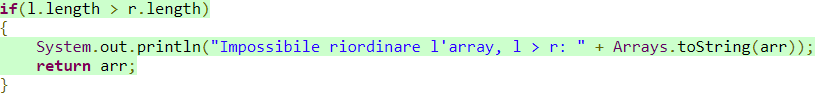
Per quanto riguarda il Branch Coverage, abbiamo verificato che per ogni istruzione ogni branch sia stato coperto dai test:

Immagine che contiene testo, Carattere, schermata, linea

Descrizione generata automaticamente

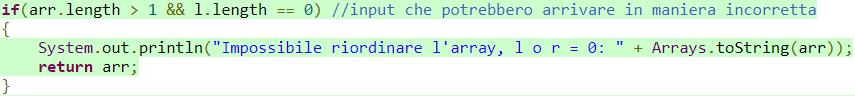
Branch Coverage =

* T1: arr true, l false, r false
* T2: arr false, l false, r false



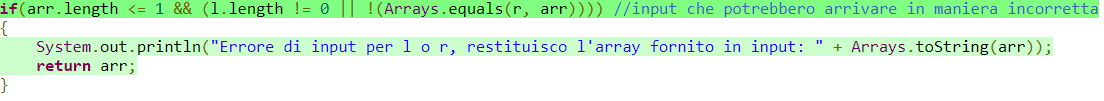
Branch Coverage =

* T7: true
* T4: false



Branch Coverage =

* T6: arr true, l true
* T8: arr false, l true



Branch Coverage =

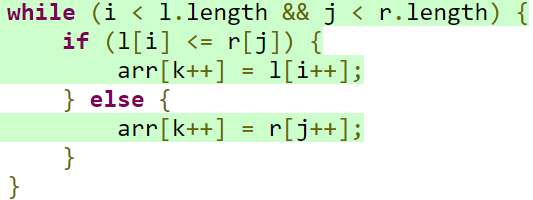
* T9: arr true, l true, r true
* T8: arr true, l false, r true

Immagine che contiene testo, Carattere, schermata, linea

Descrizione generata automaticamente

Branch Coverage =

* T8: arr true
* T7: arr false



Valutiamo il while:

Branch Coverage =

* T12: i true, j true
* T12: i false, j false (Utilizzando un test nel quale viene effettuato un ordinamento, senza casi particolari, la condizione nel while sarà vera finchè non avrà confrontato tutti gli elementi di l con quelli di r, perciò in quest’unico caso di test entrambi i branch vengono testati).

Valutiamo l’if:

- T12: a true

- T12: a false

Immagine che contiene testo, Carattere, numero, linea

Descrizione generata automaticamente

Branch Coverage =

* T12: false
* T12: true

Immagine che contiene testo, Carattere, schermata, linea

Descrizione generata automaticamente

Branch Coverage =

* T12: true
* T12: false

In conclusione, possiamo affermare che abbiamo raggiunto il 100% di **Branch Coverage** per ogni istruzione.

Proseguiamo con la Rule of Thumb adottando il principio di **code coverage MC/DC** per le espressioni più complesse e il **Condition + Branch** per quelle meno complesse:

**MC/DC**

Immagine che contiene testo, Carattere, schermata, linea

Descrizione generata automaticamente

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test case | arr == null | l == null | r == null | Decision |
| T1 | true | false | false | true |
| T2 | false | false | false | false |
| T3 | false | true | false | true |
| T5 | false | false | true | true |

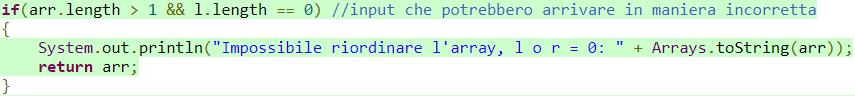
Arr: {T1,T2}

L: {T2,T3}

R:{T2,T5}

I test che quindi prendiamo sono: **{T1, T2, T3, T5}.**

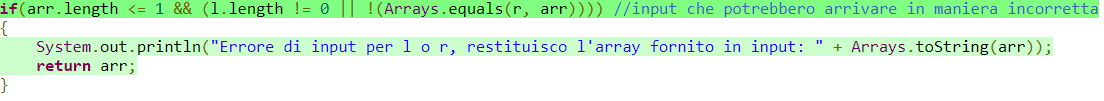
**Condition + Branch:**



C + B coverage =

* T6: arr true, l true
* T9: arr false, l false

**MC/DC**



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test case | arr.length ==1 | l.length != 0 | R != arr | decision |
| T1 | false | true | true | false |
| T2 | false | false | true | false |
| T4 | false | False | false | false |
| T5 | false | true | false | false |
| T8 | true | false | false | false |
| T9 | true | true | false | true |
| T10 | true | false | true | true |
| T11 | true | true | true | true |

Arr:{T1,T11}{T2,T10}**{T5,T9}**

L:{T8,T9}

R:{T8,T10}

I test che quindi prendiamo sono **{T5,T8,T9,T10}.**

**Condition + Branch (Condizione del while):**

Immagine che contiene testo, Carattere, calligrafia, linea

Descrizione generata automaticamente

C + B coverage =

* T12: i true, j true
* T12: i false, j false

I test che avevamo selezionato per la Branch Coverage, in questo caso, si sono rilevati sufficienti per coprire anche la Condition + Branch.

In conclusione, in seguito allo specification-based testing, abbiamo raggiunto il 100% di line coverage; perciò, abbiamo effettuato l’analisi di copertura del codice seguendo la Rule of Thumb, e abbiamo notato che anche per le espressioni più complesse non è stato necessario aggiungere alcun test white box.

## Structural Testing and Code Coverage (Task 2)

Il codice che abbiamo selezionato è quello del Team GAF. Per il task 2 abbiamo seguito i seguenti passi:

### Comprensione dei requisiti:

* 1. L’obiettivo del programma è quello di calcolare il quadrato dei numeri pari e il cubo dei numeri dispari, presenti in un array di numeri interi positivi, forniti in input e poi

ordinati in modo crescente. In maniera parallela, il programma ordina una lista di nomi, anch’essi forniti in input.

* 1. Il programma riceve 4 parametri:
* numeri che rappresenta la lista di numeri interi positivi
* nomi che rappresenta la lista di stringhe da riordinare
* m, rappresenta la dimensione della lista dei nomi
* n, rappresenta la dimensione della lista dei numeri
  1. 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

### Esplora cosa fa il programma per vari input:

****

**Immagine che contiene testo, Carattere, schermata, nero

Descrizione generata automaticamente**

****

**Immagine che contiene testo, Carattere, schermata, design

Descrizione generata automaticamente**

****

****

****

**Immagine che contiene testo, Carattere, schermata, design

Descrizione generata automaticamente**

**Immagine che contiene testo, schermata, Carattere, design

Descrizione generata automaticamente**

### 3) Esplora gli input, output e identifica le partizioni:

**a.** Input individuali

Parametro m:

1. > 0
2. < = 0

Parametro n:

1. > 0
2. <= 0

Parametro nomi:

1. Null
2. Array vuoto
3. Array di lunghezza > 1

Parametro numeri:

1. Null
2. Array vuoto
3. Array di lunghezza > 1
4. Combinazioni input:
5. m,n sono diversi
6. numeri è null e m,n sono uguali
7. nomi è null e m,n sono uguali
8. numeri vuoto e m,n sono uguali
9. nomi vuoto e m,n sono uguali
10. numeri e nomi di lunghezza > 1 e m,n sono uguali
11. Classi di Output:

Stringa

### 4) Identifica i casi limite (boundary cases):

Analizzando il codice abbiamo individuato la presenza di boundary cases in corrispondenza  
delle seguenti condizioni:

* if (n<=0) (questa condizione serve per verificare che n e di conseguenza m, poiché devono essere uguali, siano > 0)
  + - * **0** è l’on point: il punto che si trova al limite (presente all’interno della condizione)
      * **1** è l’off point: il punto più vicino al limite che appartiene alla partizione successiva (n > 0) e poiché l’on point rende la condizione vera, l’off point sarà il primo punto che la rende falsa.
      * **0** è l’in point: il punto che rende la condizione vera
      * **1** è l’out point: il punto che rende la condizione falsa.
* If (numeri[i] <0)
  + - * **0** è l’on point: il punto che si trova al limite (presente all’interno della condizione)
      * **-1** è l’off point: il punto più vicino al limite che appartiene alla partizione successiva (numeri[i] < 0) e poiché l’on point rende la condizione falsa, l’off point sarà il primo punto che la rende vera.
      * **-1** è l’in point: il punto che rende la condizione vera
      * **0** è l’out point: il punto che rende la condizione falsa.
* if (numeri[i] > 99)
  + 1. **99** è l’on point: il punto che si trova al limite (presente all’interno della condizione)
    2. **100** è l’off point: il punto più vicino al limite che appartiene alla partizione successiva (numeri[i] > 99) e poiché l’on point rende la condizione falsa, l’off point sarà il primo punto che la rende vera.
    3. **100** è l’in point: il punto che rende la condizione vera
    4. **99** è l’out point: il punto che rende la condizione falsa.
* if (quadrato < 100) (questa condizione serve per verificare che il quadrato dei numeri presenti nell’array “numeri” sia < 100 e di conseguenza possa essere accettato come valore)
  + - * **100** è l’on point: il punto che si trova al limite (presente all’interno della condizione)
      * **99** è l’off point: il punto più vicino al limite che appartiene alla partizione successiva (quadrato < 100) e poiché l’on point rende la condizione falsa, l’off point sarà il primo punto che la rende vera.
      * **99** è l’in point: il punto che rende la condizione vera
      * **100** è l’out point: il punto che rende la condizione falsa.
* if (cubo < 200) (questa condizione serve per verificare che il cubo dei numeri presenti nell’array “numeri” sia < 200 e di conseguenza possa essere accettato come valore)
  + - * **200** è l’on point: il punto che si trova al limite (presente all’interno della condizione)
      * **199** è l’off point: il punto più vicino al limite che appartiene alla partizione successiva (cubo < 200) e poiché l’on point rende la condizione falsa, l’off point sarà il primo punto che la rende vera.
      * **199** è l’in point: il punto che rende la condizione vera
      * **200** è l’out point: il punto che rende la condizione falsa.

### 5) Elaborare i casi di test

**a. *Casi di test eccezionali:***

T1. n è diverso da m

T2. numeri è null

T3. nomi è null

T4. numeri e nomi hanno lunghezze diverse

T5. Almeno un elemento di nomi è null

***b. Casi di test per soli numeri pari il cui quadrato è minore di 100:***

T6. numeri ha n elementi pari aventi i quadrati minori di 100, nomi ha m elementi, n e m sono uguali  
***c. Casi di test per soli numeri dispari il cui cubo è minore di 200:***

T7. numeri ha n elementi dispari aventi i cubi minori di 200, nomi ha m elementi, n e m sono uguali

1. **Boundary test**

T8. n ed m sono uguali a 0

T9. Almeno un elemento di numeri è negativo

T10. Almeno un elemento di numeri è maggiore di 99

T11. Almeno un quadrato di un elemento di numeri pari è maggiore di 100

T12. Almeno un cubo di un elemento di numeri dispari è maggiore di 200

### 6) Automatizza i casi di test

**T1, T2, T3, T4, T5:**

Immagine che contiene testo, schermata, software

Descrizione generata automaticamente**Immagine che contiene testo, schermata, software

Descrizione generata automaticamente**

**Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente T6:**

**Immagine che contiene testo, schermata, Carattere, software

Descrizione generata automaticamenteT7:**

**Immagine che contiene testo, schermata, software

Descrizione generata automaticamenteT8, T9, T10, T11, T12:**

### Immagine che contiene testo, schermata Descrizione generata automaticamente7) Aumenta i casi di test della suite con creatività ed esperienza

In seguito al completamento del 1.6 abbiamo raggiunto il 100% di Code Coverage effettuando 12 test (minor numero di test), rendendoci conto di avere la necessità di aggiungere due test nella prospettiva di analisi black box.

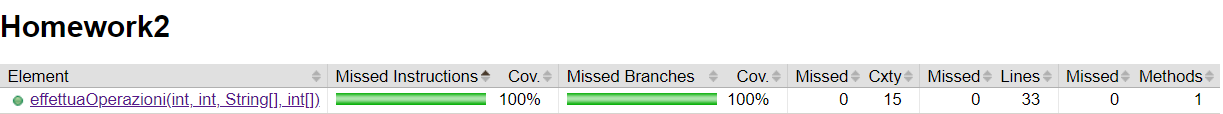
T13: nomi ha m elementi di cui almeno una stringa è vuota, numeri ha n elementi, n e m sono uguali

T14: numeri ha n elementi pari aventi i quadrati minori di 100 e dispari aventi i cubi minori di 200, nomi ha m elementi, n e m sono uguali

Immagine che contiene testo, schermata, software

Descrizione generata automaticamente

**EffettuaOperazioni**

****

**Immagine che contiene testo, schermata, Carattere, numero

Descrizione generata automaticamenteImmagine che contiene testo, schermata, Carattere, numero

Descrizione generata automaticamente**

# Homework n°3

Per la realizzazione del terzo Homework, abbiamo realizzato un codice che simula il funzionamento del gioco del “Superenalotto”. Il codice presenta i seguenti requisiti:

* + - 1. I numeri che compongono una scheda sono compresi tra 1 e 90.
      2. Una scheda vince quando tutti i suoi numeri sono presenti all’interno della scheda vincente (non importa l’ordine).
      3. Se almeno un numero nella scheda non è presente nella scheda vincente, allora la scheda perde.

Le proprietà sulla base dei requisiti sono:

1. **lose**, se almeno un numero compreso tra 1 (incluso) e 90 (incluso) non è presente nella scheda vincente, il programma restituisce false.
2. **win**, se tutti i numeri compresi tra 1 (incluso) e 90 (incluso) sono presenti nella scheda vincente, il programma restituisce true.
3. **Invalid**, se almeno un numero è minore di 1 (escluso) e maggiore di 90 (escluso), il programma genera un’*InvalidParameterException.*

Scriviamo il PBT relativo alla proprietà **lose:**

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

**Report sulle statistiche:**

Immagine che contiene schermata, testo, Carattere, linea

Descrizione generata automaticamente

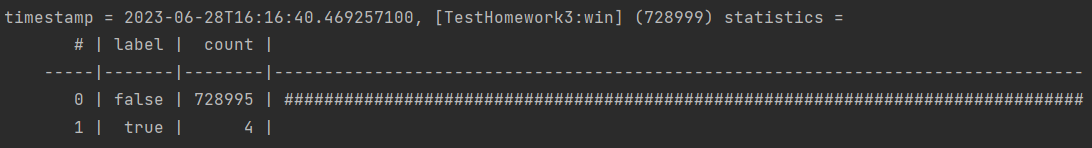
In seguito alla raccolta dei dati statistici, su 728999 tentativi la possibilità di vincere è di circa .

Scriviamo il PBT relativo alla proprietà **win:**

**Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente**

**Report sulle statistiche:**

****

In seguito alla raccolta dei dati statistici, su 728999 tentativi la possibilità di vincere è di circa .

Nota: Per avere un calcolo più accurato sarebbe necessario effettuare un calcolo statistico più approfondito, ricordando che il gioco del Superenalotto è basato sul caso.

Scriviamo il PBT relativo alla proprietà **invalid:**

**Immagine che contiene testo, schermata, software, Carattere

Descrizione generata automaticamente**

**Report sulle statistiche:**

**Immagine che contiene schermata, testo, Carattere, linea

Descrizione generata automaticamente**

Il **false** rappresenta i numeri maggiori di 90, mentre il **true** rappresenta i numeri minori di 1.