DD2459 Software reliability

Lab 2

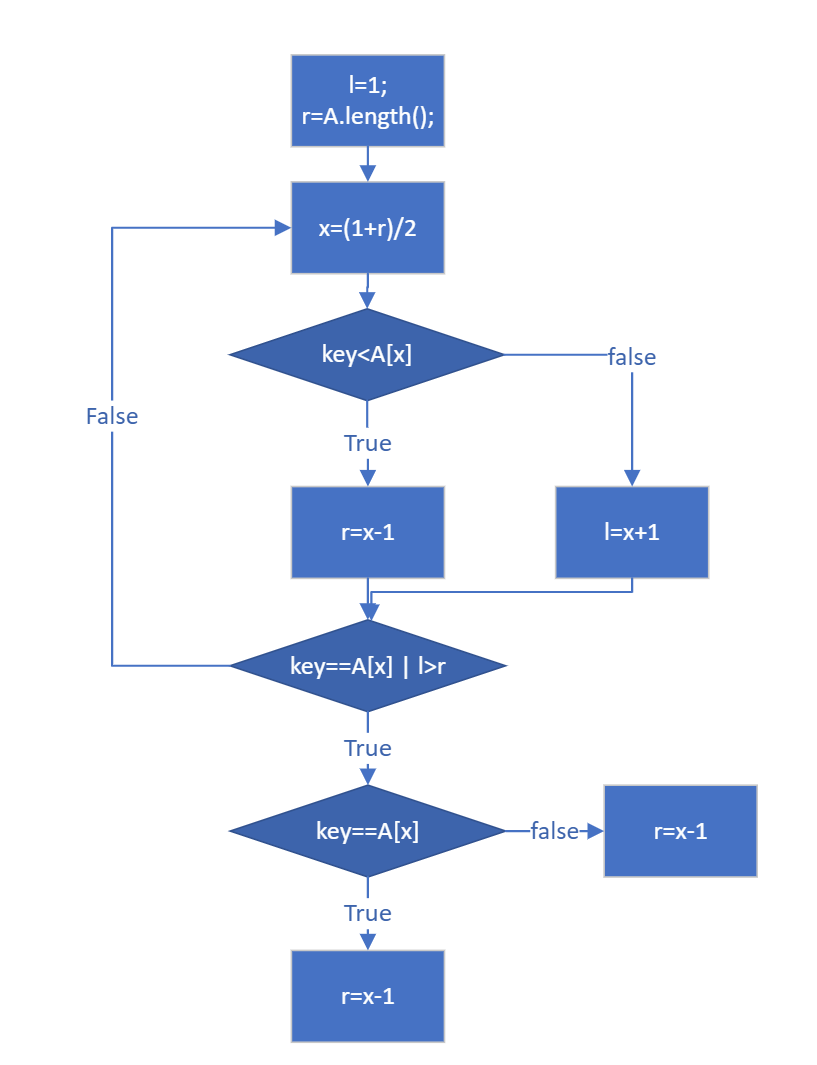
Yuze Cui

[yuzec@kth.se](mailto:yuzec@kth.se)

Rui Shi

[srui@kth.se](mailto:srui@kth.se)

Question 1



Question 2

//@ requires (\* x is positive \*);

/\*@ ensures (\* \result is an

@ approximation to

@ the square root of x \*)

@ && \result >= 0;

@\*/

public static double sqrt(double x) {

return Math.sqrt(x);

}

Pre and postcondition in JML language:

(i) //@ requires(\*all elements in array A is the same format\*);

/\*@ ensures(\*\the elements of array A are

@ reordered to a new array where the

@ elements are in ascending order\*)

@\*/

(ii) /\*@ requires(\*all elements in array A are in the same format\*) && (\*Key is in the same

@ format of elements in array A\*);\*/

/\*@ ensures(\*\If founded, result is the index of the element equals searching target,

@ if not founded, returns -1\*)

@\*/

(iii) //@ require(\*all elements in array A is in the same format\*);

/\*@ ensures (\*If there exists an element equals Key, result is 1,

@ if not founded, returns -1\*)

@\*/

(iv) //@ requires(\*all elements in array A are integer\*) && (\*Key is integer\*);

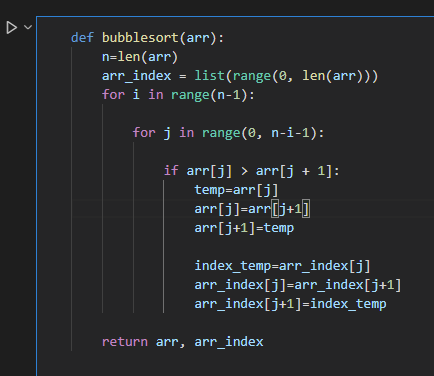
/\*@ ensures(\*\If founded, result is the index of the element equals searching target,

@ if not founded, returns -1\*)

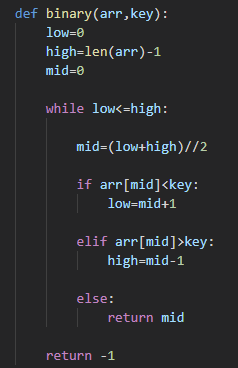
@\*/

Question 3

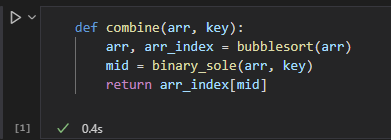
1. sorting of integer arrays of arbitrary length



1. membership queries on sorted arrays of arbitrary length using binary search



1. membership queries on unsorted arrays of arbitrary length, by combining program (i) with program (ii).



Question 4

1. Here are the results of testing 6 injected results. For this question, every list contains 10 elements, and each of them ranges from 1 to 40.

We test random testing and for 2\*10^7 times, until the results are going to be stable.

For pairwise, we test it for 2\*10^7 times to get the average value, and we test it until finding error to calculate the minimum number.

|  |  |  |
| --- | --- | --- |
| Mutation type | Random testing | Pairwise |
| Mutation 1 | Average: 22.51625 | Average: 21.90818  Min: 7 |
| Mutation 2 | Average: 4.46703 | Average: 4.36102  Min: 3 |
| Mutation 3 | Average: 4.47193 | Average: 4.33235  Min: 5 |
| Mutation 4 | Average: 4.46337 | Average: 4.38716  Min: 6 |
| Mutation 5 | Average: 4.46354 | Average: 4.33827  Min: 5 |
| Mutation 6 | Average: 4.48783 | Average: 4.36866  Min: 1 |

Mutation 1

This error happens when engineer wrongly define the range in sort part. As the result, the first element of the list will be sorted from highest value to lowest value.

|  |  |
| --- | --- |
| Wrong code | Correct code |
|  |  |

Mutation 2

This error happens when engineer wrongly range in sort part. As the result, the list sorted the elements from

|  |  |
| --- | --- |
| Wrong code | Correct code |
|  |  |

Mutation 3

This error happens when engineer wrongly dealing with lists. In this program, engineer define two lists equaled, so the index of original list is also sorted as a result.

|  |  |
| --- | --- |
| Wrong code | Correct code |
|  |  |

Mutation 4

This error happens when engineer wrongly return the value of index. Actually, the program returns the value of index -1, instead of index.

|  |  |
| --- | --- |
| Wrong code | Correct code |
|  |  |

Mutation 5

This error happens when engineer wrongly return the value of index. Actually, the program returns the value of index +1, instead of index.

|  |  |
| --- | --- |
| Wrong code | Correct code |
|  |  |

Mutation 6

This error happens when engineer wrongly return index in combination part. Actually, this program return wrong index when there are same elements in the list.

|  |  |
| --- | --- |
| Wrong code |  |
| Correct code |  |

1. As the results, we find that in general, pairwise behaves a little bit better than random generate in case of N = 10, range from 1 to 40. The first error we inject is difficult to detect, because it only affects the first element of the list. Because in general cases, there are 10 elements in a list and a key to search. When generating numbers, in usually case, the key will not be found in the list, then causing the method of random not efficient. But for pairwise, if the default value is set to be good, then pairwise will have a better performance because it can detect if the error is caused by the pair.
2. If we increase N to 100 and 500 for each error inject, here are the results.

N = 100:

|  |  |  |
| --- | --- | --- |
| Mutation type | Random testing | Pairwise |
| Mutation 1 | Average: 20.2312 | Average: 19.98751  Min: 7 |
| Mutation 2 | Average: 4.01254 | Average: 3.98725  Min: 3 |
| Mutation 3 | Average: 3.78521 | Average: 3.73145  Min: 4 |
| Mutation 4 | Average: 3.75821 | Average: 3.67581  Min: 6 |
| Mutation 5 | Average: 3.32145 | Average: 3.13461  Min: 5 |
| Mutation 6 | Average: 3.52178 | Average: 3.14734  Min: 1 |

N = 500:

|  |  |  |
| --- | --- | --- |
| Mutation type | Random testing | Pairwise |
| Mutation 1 | Average: 19.14352 | Average: 18.24751  Min: 7 |
| Mutation 2 | Average: 3.57142 | Average: 3.34571  Min: 3 |
| Mutation 3 | Average: 3.67124 | Average: 3.51224  Min: 4 |
| Mutation 4 | Average: 3.32174 | Average: 3.14587  Min: 6 |
| Mutation 5 | Average: 2.98204 | Average: 2.57142  Min: 5 |
| Mutation 6 | Average: 3.42581 | Average: 2.95871  Min: 1 |

From the results, we can observe that when N increases to large numbers, random testing and pairwise all behave better than before, because each list contains more information, and the possibility of detecting errors also increases. Also pairwise behaves a little better than random testing.