# Second assignment

**Sylvain Gabry (901215-T255) – sjygabry@kth.se**

**Gabor Andai (880619-4512) – andai@kth.se**

### Algorithm

method BinarySearch( A : array of integer, key : integer ) : integer

begin

var x, l, r : integer;

l=1; r = A.length();

repeat

x = (l+r) div 2;

if key < A[x] then r=x-1 else l=x+1

until ( key==A[x]) or (l>r)

if key==A[x] then return x else return -1

end

# Question 1

**Condensation graph**

# Question 2

### 2.1 Sorting - Pre and Post conditions

/\*@ requires !Empty();

@ ensures

@ element[i] <= element[i+1]

@ also

@ ensures

@ element\_num == \old(element\_num)

\*/

### 2.2 Searching (also applies for binary searching) - Pre and Post conditions

/\*@ requires !Empty();

@ also

@ requires validateArraySorted() == true;

@ ensures

@ (\forall int i; (0<=i && i<array.length()); array[i]!=key)

@ ==> \result == -1

@ also

@ ensures

  @ (\exist int i; (0<=i && i<array.length()); array[i]==key)

@ ==> (\result >= 0 ) && (\result < array.length())

@ && (array[\result] == key)

  \*/

### 2.3 Membership - Pre and Post conditions

/\*@ requires !Empty();

@ also

@ requires validateArraySorted() == true;

@ ensures

@ (\forall int i; ( 0<=i && i<array.length()); array[i]!=key)

@ ==> \result == false

@ also

@ ensures

  @ (\exist int i; ( 0<=i && i<array.length()); array[i]==key)

@ ==> \result == true

  \*/

# Question 3

See annex for the code.

# Question 4

See annex for the code.

## 4.1 Results

**Injections**

See code below. Injections are described in comments.

Size of the array was 20.

|  |  |  |
| --- | --- | --- |
| Injection | Random testing | Pairwise testing |
| 1. | Infinite loop | Infinite loop |
| 2. | Fail | Fail |
| 3. | Infinite loop | Infinite loop |
| 4. | **Fail** | **OK (most of the time)** |
| 5. | Fail | Fail |
| 6. | Fail | Fail |

The Injection-4 was the only case where there was a difference, between pair-wise and random testing. We injected left=2, which means that the binary search algorithm doesn’t check the first two elements of the array.

The reason why the pair-wise testing passed is because the key that we choose is usually not placed in the first two elements of the array.

However, we were running the random test much more times, therefore it happened that the key was in one of the first two elements after sorting, thus the test had failed.

## 4.2 Injections

public boolean membershipQueryOnSortedArr(int[] array, int key) {

//sanity check on input arguments

if(array == null || array.length == 0){

return false;

}

//sanity check on the array

if (!validateArraySorted(array)) {

return false;

}

int x,left,right;

x=0;

/\* INJECTION-4: left = 2 \*/

left=0;

/\* INJECTION-5: right = array.length \*/

right = array.length -1 ;

/\* INJECTION-3: key != array[x] || left <=right \*/

/\* INJECTION-6: key != array[x] && left < right \*/

while ( key != array[x] && left <= right ){

x = (left+right) >> 1;

if( key < array[x]){

/\* INJECTION-1: right = x + 2 \*/

right = x - 1;

}

else{

left = x + 1;

}

}

/\* INJECTION-2: key != array[x] \*/

if (key == array[x]){

return true;

}

else{

return false;

}

}

## 4.3 Increasing test effort

Before running the binary search on the array, we checking that the array is sorted correctly, so we did not make injections in the first program.

Regarding the binary search, we observed that running tests more times is increasing the chance to detect the error, that Injection-4 caused. The explanation is in 4.4.

## 4.4 Increasing array size

All the injections work in the same way, except Injection-4. For this injection, not only the size of the array, but also the maximum value of the random numbers has an impact on the test results.

Increasing the size and the maximum value, we have a better chance to discover the error.

# Annex

**quicksort.java**

/\*

\* To change this template, choose Tools | Templates

\* and open the template in the editor.

\*/

package quicksort;

/\*\*

\*

\* @author

\*/

public class quicksort {

private int[] elements;

private int no\_of\_elements;

public void sort(int[] values){

//sanity check

if(values == null || values.length == 0){

return;

}

//set members

this.elements = values;

no\_of\_elements = values.length;

//call sort algorithm

qs(0, no\_of\_elements - 1);

}

private void qs(int low, int high){

int i = low;

int j = high;

//select middle element as a pivot

int pivot = elements[ low + ((high - low)/2) ];

// split elements into two list

while (i <= j){

// iterate through the left side, searching for elements

// that are bigger than pivot

while (elements[i] < pivot) {

i++;

}

// iterate through the right side, searching for elements

// that are smaller than pivot

while (elements[j] > pivot) {

j--;

}

/\*if we have found an element in the left side which is smaller than pivot,

\* or we have found an element in the right side which is larger than pivot

\* then swap them

\*/

if (i <= j){

swap(i, j);

i++;

j--;

}

}

//call recursively

if (low < j){

qs(low, j);

}

if (i < high){

qs(i, high);

}

}

private void swap(int i, int j) {

int tmp = elements[i];

elements[i] = elements[j];

elements[j] = tmp;

}

}

**membershipqry.java**

/\*

\* To change this template, choose Tools | Templates

\* and open the template in the editor.

\*/

package membershipqry;

import quicksort.quicksort;

/\*\*

\*

\* @author

\*/

public class membershipqry {

public boolean membershipQueryOnUnSortedArr(int[] array, int key, quicksort sortObj) {

//sanity check

if(array == null || array.length == 0){

return false;

}

//sorting the array

sortObj.sort(array);

//membership search

return membershipQueryOnSortedArr(array,key);

}

public boolean membershipQueryOnSortedArr(int[] array, int key) {

//sanity check on input arguments

if(array == null || array.length == 0){

return false;

}

//sanity check on the array

if (!validateArraySorted(array)) {

return false;

}

int x,left,right;

x=0;

/\* INJECTION-4: left = 2 \*/

left=2;

/\* INJECTION-5: right = array.length \*/

right = array.length -1 ;

/\* INJECTION-3: key != array[x] || left <=right \*/

/\* INJECTION-6: key != array[x] && left < right \*/

while ( key != array[x] && left <= right ){

x = (left+right) >> 1;

if( key < array[x]){

/\* INJECTION-1: right = x + 2 \*/

right = x - 1;

}

else{

left = x + 1;

}

}

/\* INJECTION-2: key != array[x] \*/

if (key == array[x]){

return true;

}

else{

return false;

}

}

private boolean validateArraySorted(int[] testArray) {

for (int i = 0; i < testArray.length - 1; i++) {

if (testArray[i] > testArray[i + 1]) {

return false;

}

}

return true;

}

}

**randomTest.java**

/\*

\* To change this template, choose Tools | Templates

\* and open the template in the editor.

\*/

package randomTest;

import org.junit.After;

import org.junit.AfterClass;

import org.junit.Before;

import org.junit.BeforeClass;

import org.junit.Test;

import static org.junit.Assert.\*;

import java.util.Arrays;

import java.util.Random;

import membershipqry.membershipqry;

import quicksort.quicksort;

/\*\*

\*

\* @author

\*/

public class randomTest {

private int[] testArray;

private int key;

private final static int SIZE = 20;

private final static int MAX\_VALUE = 20;

private quicksort sorter = new quicksort();

private membershipqry query = new membershipqry();

public randomTest() {

}

@BeforeClass

public static void setUpClass() {

}

@AfterClass

public static void tearDownClass() {

}

@Before

public void setUp() throws Exception{

testArray = new int[SIZE];

Random generator = new Random();

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

testArray[i] = generator.nextInt(MAX\_VALUE);

}

key = testArray[0];

}

@After

public void tearDown() {

}

@Test

public void testNullPointer() {

boolean ret = query.membershipQueryOnUnSortedArr(null,key,sorter);

if(ret){

fail("wrong return value");

}

assertTrue(true);

}

@Test

public void testEmpty() {

boolean ret = query.membershipQueryOnUnSortedArr(new int[0],key,sorter);

if(ret){

fail("wrong return value");

}

assertTrue(true);

}

@Test

public void testMembershipQuery() {

//run test 10x times

for(int i=0; i<40000; i++){

//create random vector for every time

Random generator = new Random();

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

testArray[j] = generator.nextInt(MAX\_VALUE);

}

//set key for the first element of the unsorted array,

//make sure we can find it

key = testArray[0];

boolean ret = query.membershipQueryOnUnSortedArr(testArray,key,sorter);

if (!validateArraySorted(testArray)) {

fail("Sort failed");

}

if(!ret){

fail("Couldn't find member");

}

}

assertTrue(true);

}

@Test

public void testMembershipQueryFalse() {

//run test 10x times

for(int i=0; i< 40000; i++){

//create random vector for every time

Random generator = new Random();

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

testArray[j] = generator.nextInt(MAX\_VALUE);

}

//set key outside the maxvalue, make sure we can't find it

key = ( MAX\_VALUE + 1 + generator.nextInt(MAX\_VALUE));

boolean ret = query.membershipQueryOnUnSortedArr(testArray,key,sorter);

if (!validateArraySorted(testArray)) {

fail("Sort failed");

}

if(ret){

fail("Found member? impossible...");

}

}

assertTrue(true);

}

private boolean validateArraySorted(int[] testArray) {

for (int i = 0; i < testArray.length - 1; i++) {

if (testArray[i] > testArray[i + 1]) {

return false;

}

}

return true;

}

}

**pairwiseTest.java**

/\*

\* To change this template, choose Tools | Templates

\* and open the template in the editor.

\*/

package pairwiseTest;

import java.util.Random;

import membershipqry.membershipqry;

import quicksort.quicksort;

import org.junit.After;

import org.junit.AfterClass;

import org.junit.Before;

import org.junit.BeforeClass;

import org.junit.Test;

import static org.junit.Assert.\*;

/\*\*

\*

\* @author jbs

\*/

public class pairwiseTest {

private int[] testArray;

private int[] defaultArray;

private int key;

private final static int SIZE = 20;

private final static int MAX\_VALUE = 200000;

private quicksort sorter = new quicksort();

private membershipqry query = new membershipqry();

public pairwiseTest() {

//create reference defaultArray, it never change during the exectuion

defaultArray = new int[SIZE];

//create testArray where we change the values

testArray = new int[SIZE];

//fill up testArray

Random generator = new Random();

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

defaultArray[i] = generator.nextInt(MAX\_VALUE);

}

}

@BeforeClass

public static void setUpClass() {

}

@AfterClass

public static void tearDownClass() {

}

@Before

public void setUp() throws Exception{

}

@After

public void tearDown() {

}

@Test

public void testNullPointer() {

boolean ret = query.membershipQueryOnUnSortedArr(null,0,sorter);

if(ret){

fail("wrong return value");

}

assertTrue(true);

}

@Test

public void testEmpty() {

boolean ret = query.membershipQueryOnUnSortedArr(new int[0],0,sorter);

if(ret){

fail("wrong return value");

}

assertTrue(true);

}

@Test

public void testMembershipQuery() {

Random generator = new Random();

//change one of the input variables

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

//change one more input variables

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

//copy the reference testArray to the keyArray

for(int k=0; k<defaultArray.length; k++){

testArray[k] = defaultArray[k];

}

//pairwise test

//change two input variables

testArray[i] = generator.nextInt(MAX\_VALUE);

testArray[j] = generator.nextInt(MAX\_VALUE);

//set key, we make sure that key is in the table

key = testArray[0];

//run the membership query with the testArray and key

//expecting for a ret == true

boolean ret = query.membershipQueryOnUnSortedArr(testArray,key,sorter);

//validate if the array is sorted

if (!validateArraySorted(testArray)) {

fail("Sort failed");

}

if(!ret){

fail("Couldn't find member");

}

}

}

assertTrue(true);

}

@Test

public void testMembershipQueryFalse() {

Random generator = new Random();

//change one of the input variables

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

//change one more input variables

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

//copy the reference testArray to the keyArray

for(int k=0; k<defaultArray.length; k++){

testArray[k] = defaultArray[k];

}

//pairwise test

//change two input variables

testArray[i] = generator.nextInt(MAX\_VALUE);

testArray[j] = generator.nextInt(MAX\_VALUE);

//set key, we make sure that key is NOT in the table

key = MAX\_VALUE + 1 + generator.nextInt(MAX\_VALUE);

//run the membership query with the testArray and key

//expecting for a ret == true

boolean ret = query.membershipQueryOnUnSortedArr(testArray,key,sorter);

//validate if the array is sorted

if (!validateArraySorted(testArray)) {

fail("Sort failed");

}

if(ret){

fail("Find a member? impossible...");

}

}

}

assertTrue(true);

}

private boolean validateArraySorted(int[] testArray) {

for (int i = 0; i < testArray.length - 1; i++) {

if (testArray[i] > testArray[i + 1]) {

return false;

}

}

return true;

}

}