DD2459: Software Reliability

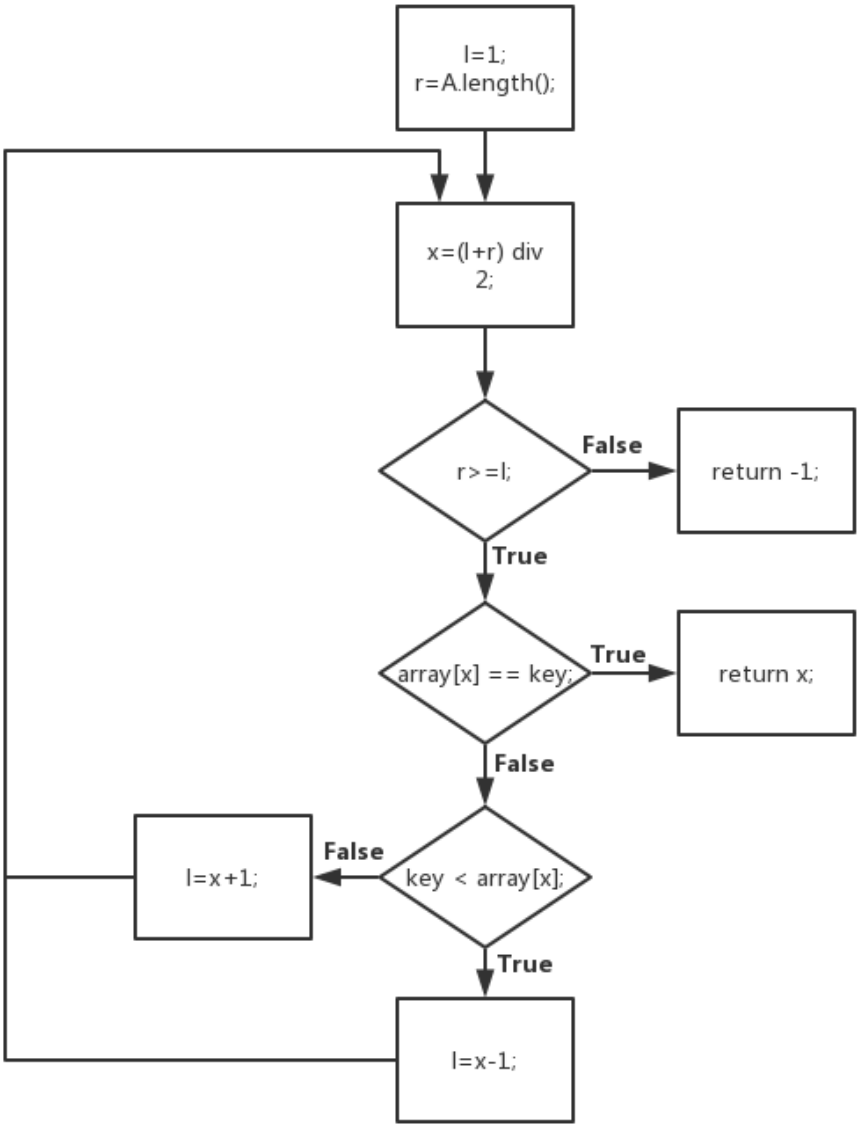
Lab 2 Report: Black-box and Requirements-Based Testing: Sorting and Searching

Yuxiang Liu
yuxliu@kth.se
9505152455

Guanghao Guo
gguo@kth.se
9508099372

1. Question 1

The image of the condensation graph is as follow:



2. Question 2

2.1 Sorting

```
/*@      requires
         inputArray != null &&
         inputArray.length > 0

         ensures
          ( \forall int i; 0 <= i && i < inputArray.length - 1 ; inputArray [ i ] <= inputArray [ i+1 ] )
          && inputArray.length == \old ( inputArray.length )

         @*/
```

2.2 Searching

```
/*@
          requires
          key != null && inputArray != null &&
          inputArray.length > 0

         ensures
         \result == -1 ==> (\forall int i; 0 <= i && i < inputArray.length ; inputArray [ i ] != key ) ||
         \result == i ==> (\exists int i; 0 <= i && i < inputArray.length ; inputArray [ i ] == key )

         @*/
```

2.3 Membership

```
/*@      requires
         inputArray != null && inputArray.length > 0 ;

         ensures
         \result == 0 ==> (\forall int i; 0 <= i && i < inputArray.length ; inputArray [ i ] != key ) ||
         \result == 1 ==> (\exists int i; 0 <= i && i < inputArray.length ; inputArray [ i ] == key )

         @*/
```

2.4 Binary searching

```
/*@      requires
         key != null && inputArray != null && inputArray.length > 0  &&
         \forall int i; 0 <= i && i < inputArray.length - 1 ; inputArray [ i ] <= inputArray [ i+1 ]
         ensures
         \result == -1 ==> (\forall int i; 0 <= i && i < inputArray.length ; inputArray [ i ] != key ) ||
          \result == i ==> (\exists int i; 0 <= i && i < inputArray.length ; inputArray [ i ] == key ) ;
         @*/
```

3. Question 3

We attach the code in C of the basic algorithms implemented:

```
#include <stdio.h>
void sorting(unsigned char* array,int r)
{
        for (int i = 0; i < r; i++)
    {
        for (int j = 0; j < (r - i - 1); j++)
        {
            if (array[j] > array[j + 1])
            {
                int temp = array[j];
                array[j] = array[j + 1];
                array[j + 1] = temp;
            }
        }
    }
    for (int k = 0; k < r;k++){
        printf("%d ",array[k]);
    }
    //return array[r];
}
int BinarySearch(unsigned char* array,unsigned char key,int r){
    // for (int k = 0; k < r;k++){
    //      printf("%d ",array[k]);
    //  }
    int l=0;
    int x =(l+r)/2;
    while(r>=l){
        if(array[x]==key){
            return x;
        }
        if (key<array[x]){
            r=x-1;
        }
        else{
            l=x+1;
        }
        x=(l+r)/2;
```

```
        //printf("%d\n",x);
    }
    return -1;
};
int oracle(unsigned char* b, int num, unsigned char key, int result)
{
    int i;
    int flag = -1;
    for (i = 0; i < num - 1; i++)
    {
        if (b[i] > b[i + 1])
            return -1; // sort fail;
    }
    for (i = 0; i < num; i++)
    {
        if (b[i] == key)
            flag = 1;
    }
    if (result == -1 && flag == -1)
        return 1; //pass
    else if (result != -1 && flag == 1 && key == b[result])
        return 1;//pass
    else
        return -2; //search fail
}
```

4. Question 4

4.1 Random test

We chose to generate 100 random test cases. The array is set to have length 20 and the range of the elements is chosen at 0 < element < 50 and the key 0 < key < 50.

Attached Code in C

```
#include "binary_search.c"
#include <stdlib.h>
#include <time.h>
int main(void){
    int count =0;
    srand(time(NULL));
    while(count < 100)
```

```
    {
    unsigned char array[20];
    int r = sizeof (array) / sizeof (array[0]);
    for (int i = 0; i < r; i++){
        array[i] = rand() % 50+1;
    }
    for (int j=0; j < r; j++){
        printf("%d ", array[j]);
    }
    unsigned char key=rand() % 50+1;
    printf("key = %d\n",key);
    sorting(&array[0],r);
    int result=BinarySearch(&array[0],key,r);
    if (result==-1){
        printf("The element is not in the array\n");
    }else{
        printf("The element is prensent at index " "%d\n",result);
    }
    int flag = oracle(&array[0], r, key, result);
     if (flag == -1)
     printf("sort fail\n");
     else if (flag == -2)
     printf("search fail\n");
     else
     printf("pass\n");
     printf("\n");
     count++;
    }
}
```

4.2 Pairwise test

We chose to set the number of choices for the array variable at 20 and for the key we set the number of choices at 2. We chose to use specific value for the default value and each variable has another random value. This can finally generate 210 test cases.

Attached Code in C

```
#include "binary_search.c"
#include <stdlib.h>
#include <time.h>
int main(void){
```

```
    int count =0;
    srand(time(NULL));
    for (int i = 0; i < 21; i++){
        for (int j=i+1; j < 21;j++){
            count++;
            unsigned char
NewArray[21]={0,2,4,6,8,10,12,14,16,18,20,22,24,26,28,30,32,34,36,38,10
};//The last one is key
            unsigned char* NewArrayP =& NewArray[0];
            // *NewArrayP++=*DefaultP++;
            NewArray[i]=rand() % 40+1;
            NewArray[j]=rand() % 40+1;
            printf("The input array ");
            for (int m=0;m<20;m++){
                printf("%d ",NewArray[m]);
            }
            unsigned char key= NewArray[20];
            printf(" key = %d\n",key);
            sorting(NewArrayP,20);
            int result=BinarySearch(NewArrayP,key,20);
            //printf("%d",result);
            if (result==-1){
                printf("The element is not in the array\n");
            }else{
                printf("The element is present at index "
"%d\n",result);
            }
            int flag = oracle(NewArrayP,20 , key, result);
            if (flag == -1)
            printf("sort fail"  "%d\n", count);
            else if (flag == -2)
            printf("search fail" "%d\n", count);
            else
            printf("pass\n");
            printf("\n");

        }
    }

printf("%d ",count);
```

}

4.3 Mutations and Performance

Mutations	Random	Pairwise
Binary search initialized with l=1, instead of l=0	14	can't find
the wrong value of array length in sorting function	1	17
the wrong value of array length in binary search function	can't find	119
forget return value in binary search function	1	5
forget to sweep value in sorting function	1	1
wrong return value	3	5

One can notice that the random suite is performing better than the pairwise. Appart that this is due to the choices we made, we also believe that random testing is good to find most common and careless bugs that developers usually do. The pairwise testing is good for finding bugs around the logic of the program, like the third mutation, that a special input is needed. And in most cases, to detect a bug it always need more test cases in terms of pairwise test compared with random test. And the default value chose for pairwise testing is also really significant since in the beginning, I choose sequential value for default, and it turns out that becomes less effective to detect errors or bugs in sorting functions. Therefore, when choosing the default values, we need to be more careful.

4.4 Enlarge Array Size

After scaling the array size we notice that the random test suite needs more test cases to execute in order to find each mutation while surprisingly the pairwise suite performs better, since need less test to find each mutation.