DD2459 Software reliability

Lab 2

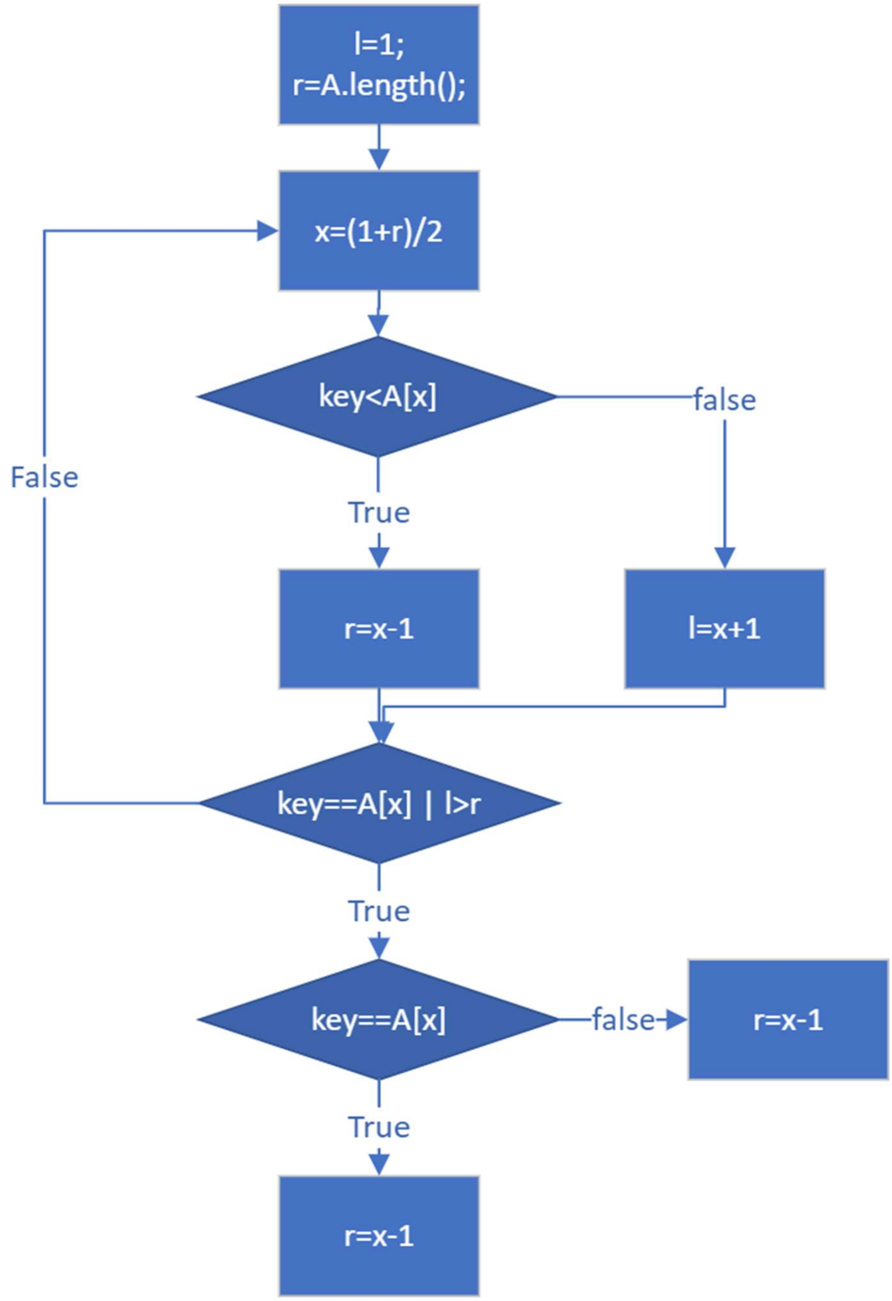
Yuze Cui

yuzec@kth.se

Rui Shi

srui@kth.se

Question 1


l=1;
r=A.length();
x=(1+r)/2
key<A[x]
false
False
True
r=x-1
l=x+1
key==A[x] | l>r
True
key==A[x]
false
r=x-1
True
r=x-1

Question 2

(i)

```
requires A!=null;
ensures (\result.length==A.length) &
    (\forall int j; 0<=j<\result.length;
        (\num_of int i; 0<=i<A.length;\result[i]==\result[j]) ==
            (\num_of int i; 0<=i<.length; A[i]==\result[j])) &
    (forall int k;0<=k<\result.length-1; result[k]<=\result[k+1])
```

(ii)

```
requires (A!=null) & (typeof(A)==typeof(int))
ensures ((\result==-1)|(A[\result]==key))& (\not_modified(A)==1)
```

(iii)

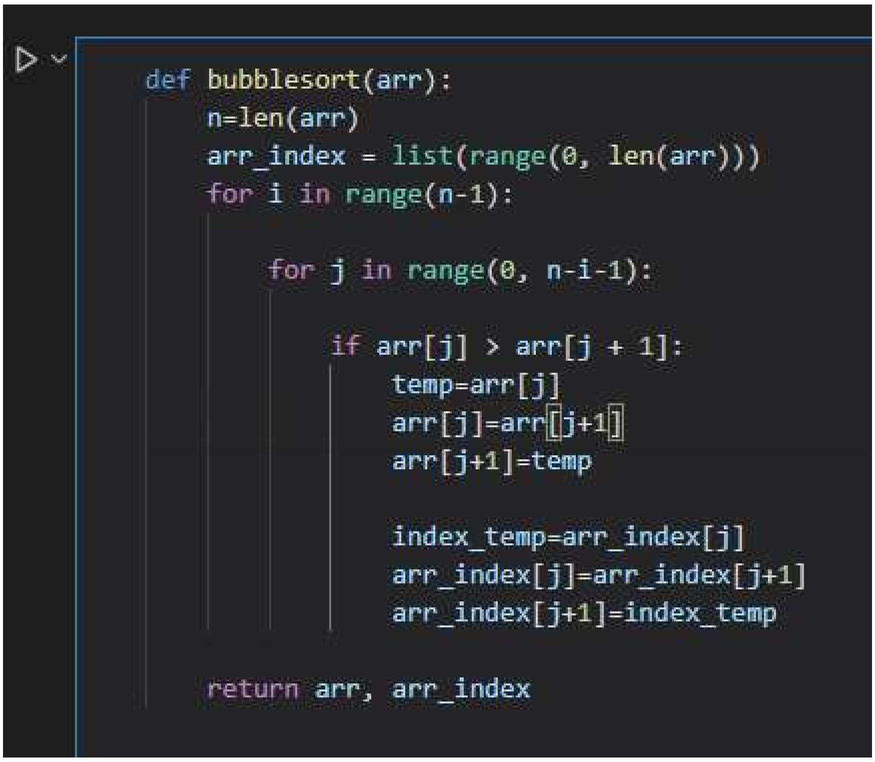
```
requires (A!=null) & (typeof(A)==typeof(int))
ensures( ((\result==key)&(\exists int i; 0<=i<A.length; A[i]==key)) |
            ((\result==-1)&!(\exists int i; 0<=i<A.length; A[i]==key)) )&
        (\not_modified(A)==1)
```

(iv)

```
requires (A!=null) & (typeof(A)==typeof(int)) &
            (forall int i;0<=i<A.length-1; A[k]<=A[k+1])
ensures ((\result==-1)|(A[\result]==key))& (\not_modified(A)==1)
```

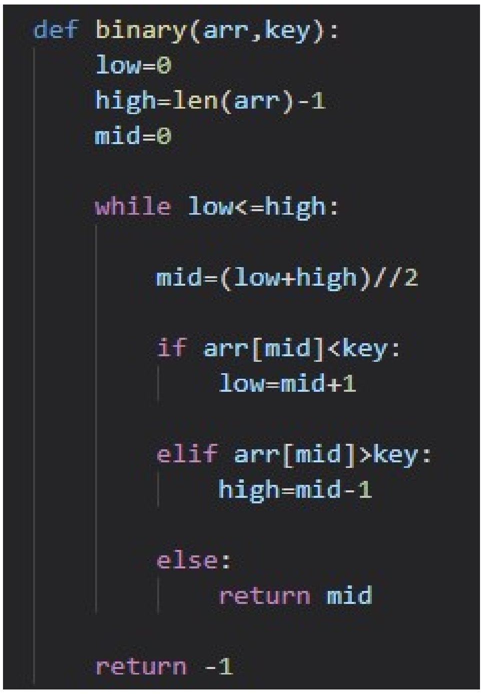
Question 3

(a) sorting of integer arrays of arbitrary length



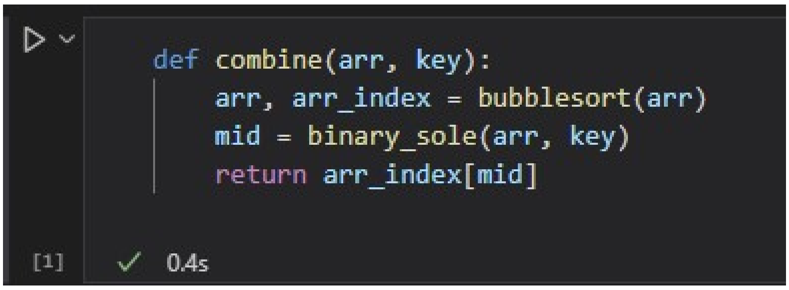
```
def bubblesort(arr):
    n=len(arr)
    arr_index = list(range(0, len(arr)))
    for i in range(n-1):

        for j in range(0, n-i-1):

            if arr[j] > arr[j + 1]:
                temp=arr[j]
                arr[j]=arr[j+1]
                arr[j+1]=temp

                index_temp=arr_index[j]
                arr_index[j]=arr_index[j+1]
                arr_index[j+1]=index_temp

    return arr, arr_index
```

(b) membership queries on sorted arrays of arbitrary length using binary search



```
def binary(arr,key):
    low=0
    high=len(arr)-1
    mid=0

    while low<=high:

        mid=(low+high)//2

        if arr[mid]<key:
            low=mid+1

        elif arr[mid]>key:
            high=mid-1

        else:
            return mid

    return -1
```

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



```
def combine(arr, key):
    arr, arr_index = bubblesort(arr)
    mid = binary_sole(arr, key)
    return arr_index[mid]
```

Question 4

(i) 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.

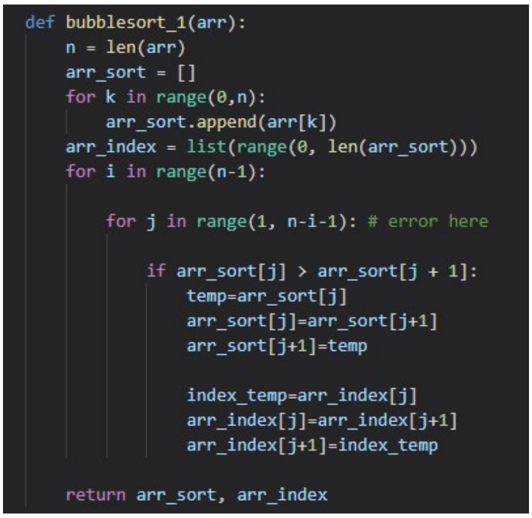
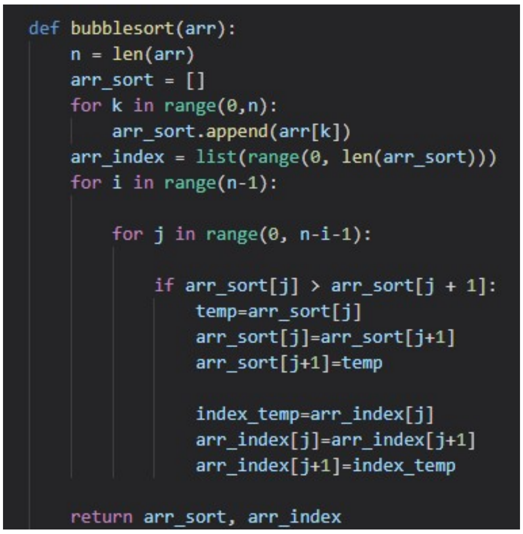
Mutation type	Random testing	Pairwise
Mutation 1	Average: 22.51625	Average: 21.90818 Min: 7
Mutation 2	Average: 4.46703	Average: 4.36102 Min: 3
Mutation 3	Average: 4.47193	Average: 4.33235 Min: 5
Mutation 4	Average: 4.46337	Average: 4.38716 Min: 6
Mutation 5	Average: 4.46354	Average: 4.33827 Min: 5
Mutation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

(ii)

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

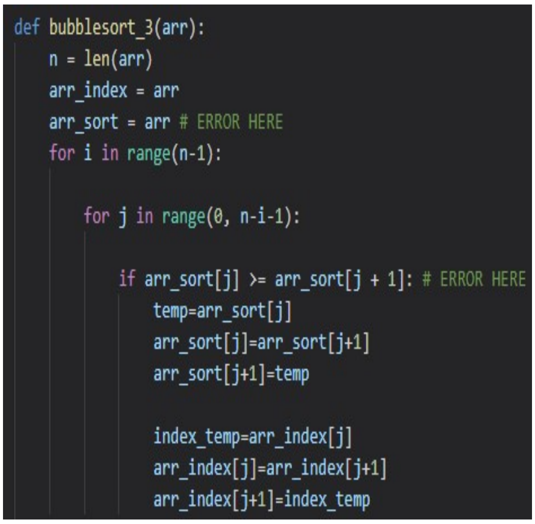
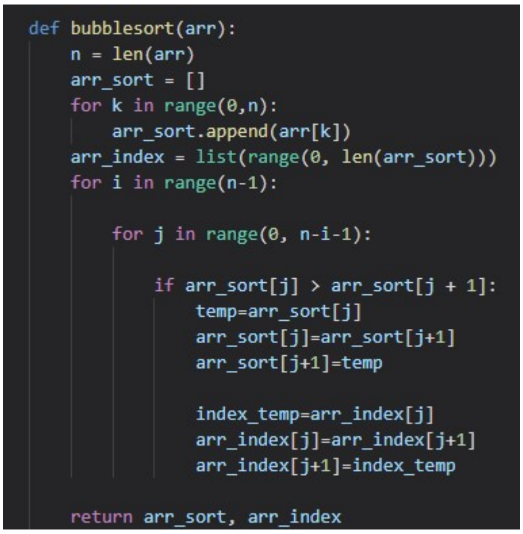
	

```
def bubblesort_1(arr):
    n = len(arr)
    arr_sort = []
    for k in range(0,n):
        arr_sort.append(arr[k])
    arr_index = list(range(0, len(arr_sort)))
    for i in range(n-1):

        for j in range(1, n-i-1): # error here

            if arr_sort[j] > arr_sort[j + 1]:
                temp=arr_sort[j]
                arr_sort[j]=arr_sort[j+1]
                arr_sort[j+1]=temp

                index_temp=arr_index[j]
                arr_index[j]=arr_index[j+1]
                arr_index[j+1]=index_temp

    return arr_sort, arr_index
```

```
def bubblesort(arr):
    n = len(arr)
    arr_sort = []
    for k in range(0,n):
        arr_sort.append(arr[k])
    arr_index = list(range(0, len(arr_sort)))
    for i in range(n-1):

        for j in range(0, n-i-1):

            if arr_sort[j] > arr_sort[j + 1]:
                temp=arr_sort[j]
                arr_sort[j]=arr_sort[j+1]
                arr_sort[j+1]=temp

                index_temp=arr_index[j]
                arr_index[j]=arr_index[j+1]
                arr_index[j+1]=index_temp

    return arr_sort, arr_index
```

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
	

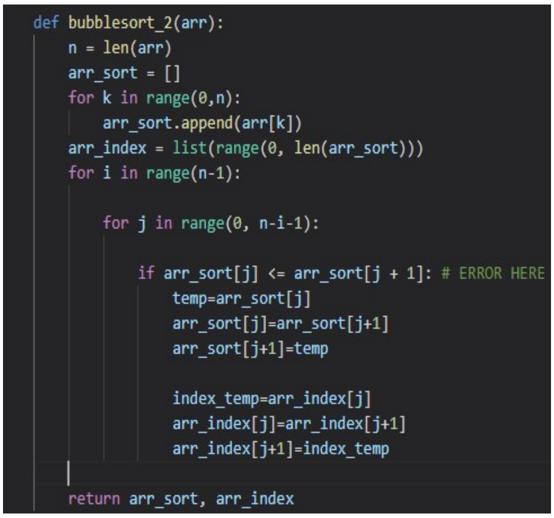
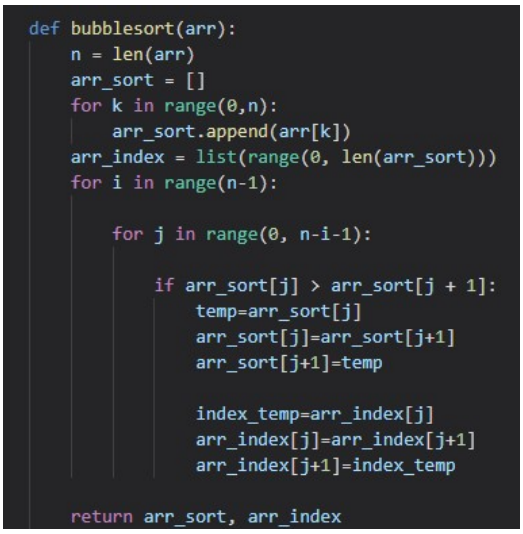
```
def bubblesort_3(arr):
    n = len(arr)
    arr_index = arr
    arr_sort = arr # ERROR HERE
    for i in range(n-1):

        for j in range(0, n-i-1):

            if arr_sort[j] >= arr_sort[j + 1]: # ERROR HERE
                temp=arr_sort[j]
                arr_sort[j]=arr_sort[j+1]
                arr_sort[j+1]=temp

                index_temp=arr_index[j]
                arr_index[j]=arr_index[j+1]
                arr_index[j+1]=index_temp
```

```
def bubblesort(arr):
    n = len(arr)
    arr_sort = []
    for k in range(0,n):
        arr_sort.append(arr[k])
    arr_index = list(range(0, len(arr_sort)))
    for i in range(n-1):

        for j in range(0, n-i-1):

            if arr_sort[j] > arr_sort[j + 1]:
                temp=arr_sort[j]
                arr_sort[j]=arr_sort[j+1]
                arr_sort[j+1]=temp

                index_temp=arr_index[j]
                arr_index[j]=arr_index[j+1]
                arr_index[j+1]=index_temp

    return arr_sort, arr_index
```

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

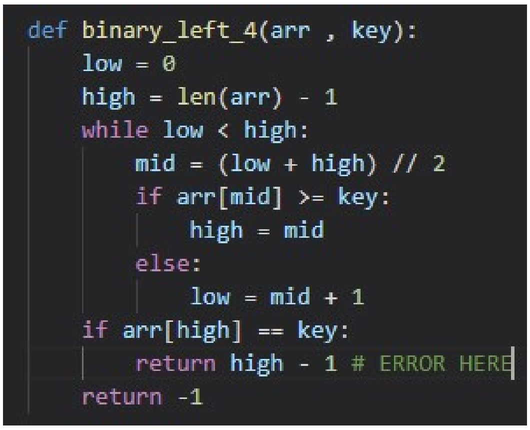
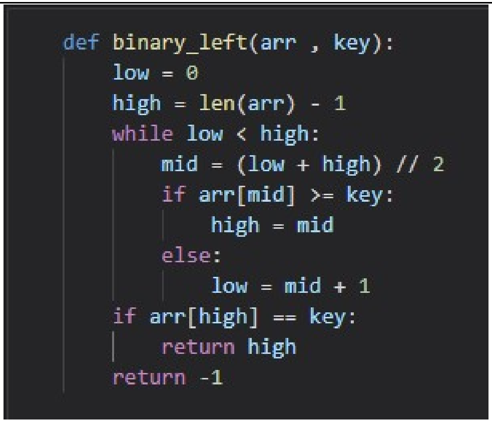
	

```python
def bubblesort_2(arr):
    n = len(arr)
    arr_sort = []
    for k in range(0,n):
        arr_sort.append(arr[k])
    arr_index = list(range(0, len(arr_sort)))
    for i in range(n-1):

        for j in range(0, n-i-1):

            if arr_sort[j] <= arr_sort[j + 1]: # ERROR HERE
                temp=arr_sort[j]
                arr_sort[j]=arr_sort[j+1]
                arr_sort[j+1]=temp

                index_temp=arr_index[j]
                arr_index[j]=arr_index[j+1]
                arr_index[j+1]=index_temp

    return arr_sort, arr_index
```

```python
def bubblesort(arr):
    n = len(arr)
    arr_sort = []
    for k in range(0,n):
        arr_sort.append(arr[k])
    arr_index = list(range(0, len(arr_sort)))
    for i in range(n-1):

        for j in range(0, n-i-1):

            if arr_sort[j] > arr_sort[j + 1]:
                temp=arr_sort[j]
                arr_sort[j]=arr_sort[j+1]
                arr_sort[j+1]=temp

                index_temp=arr_index[j]
                arr_index[j]=arr_index[j+1]
                arr_index[j+1]=index_temp

    return arr_sort, arr_index
```

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.

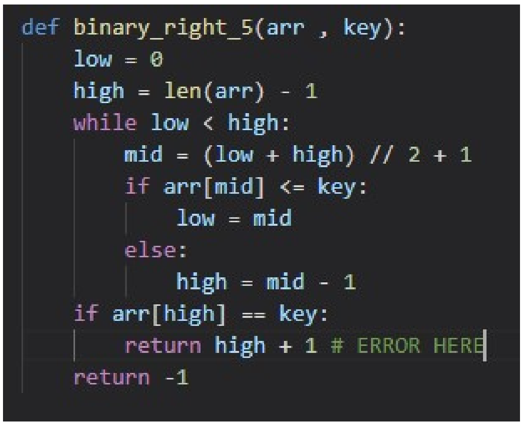
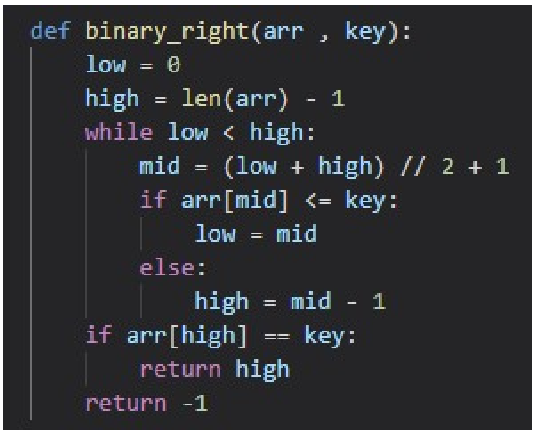
Wrong code	Correct code
	

```python
def binary_left_4(arr , key):
    low = 0
    high = len(arr) - 1
    while low < high:
        mid = (low + high) // 2
        if arr[mid] >= key:
            high = mid
        else:
            low = mid + 1
    if arr[high] == key:
        return high - 1 # ERROR HERE
    return -1
```

```python
def binary_left(arr , key):
    low = 0
    high = len(arr) - 1
    while low < high:
        mid = (low + high) // 2
        if arr[mid] >= key:
            high = mid
        else:
            low = mid + 1
    if arr[high] == key:
        return high
    return -1
```

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
	

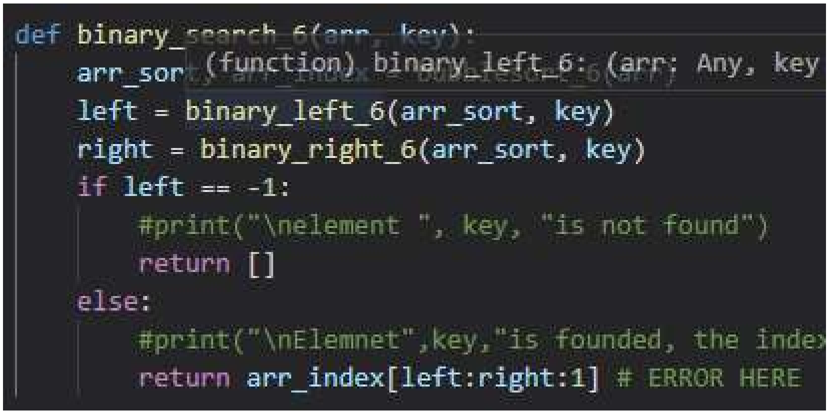
```python
def binary_right_5(arr , key):
    low = 0
    high = len(arr) - 1
    while low < high:
        mid = (low + high) // 2 + 1
        if arr[mid] <= key:
            low = mid
        else:
            high = mid - 1
    if arr[high] == key:
        return high + 1 # ERROR HERE
    return -1
```

```python
def binary_right(arr , key):
    low = 0
    high = len(arr) - 1
    while low < high:
        mid = (low + high) // 2 + 1
        if arr[mid] <= key:
            low = mid
        else:
            high = mid - 1
    if arr[high] == key:
        return high
    return -1
```

Mutation 6

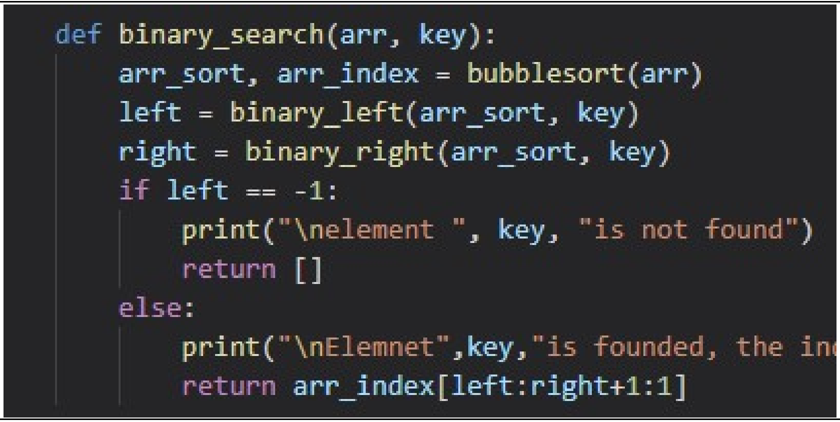
This error happens when engineer wrongly return index in combination part. Actually, this program return wrong index when there are same elements in the list.

Wrong code



```
def binary_search_6(arr, key):
    arr_sor (function) binary_left_6: (arr: Any, key
    left = binary_left_6(arr_sort, key)
    right = binary_right_6(arr_sort, key)
    if left == -1:
        #print("\nelement ", key, "is not found")
        return []
    else:
        #print("\nElemnet",key,"is founded, the index
        return arr_index[left:right:1] # ERROR HERE
```

Correct code



```
def binary_search(arr, key):
    arr_sort, arr_index = bubblesort(arr)
    left = binary_left(arr_sort, key)
    right = binary_right(arr_sort, key)
    if left == -1:
        print("\nelement ", key, "is not found")
        return []
    else:
        print("\nElemnet",key,"is founded, the in
        return arr_index[left:right+1:1]
```

(iii) 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.

(iv) 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.