Data Structures Completion Points Assignment 1

**Implementation 1.**

public static *int* search(*int* *n*, *int*[] *a*, *int* *x*) {

    if (*n* == 0)

        return -1;

*int* left = 0;

*int* right = *n* - 1;

    while (left < right) {

*int* mid = (left + right) / 2;

        if (*x* > *a*[mid])

            right = mid - 1;

        else

            left = mid;

    }

    if (*a*[left] == *x*)

        return left;

    else

        return -1;

}

This implementation is correct. Unlike in the implementation we discussed during the lecture, in this implementation check, that left == right is omitted in the last if statement. But that check is actually required only when the size of the input array is 0. In this implementation we moved the check to the beginning of the program: if the length of the array is zero, we immediately return -1, making sure that the program doesn’t run further if “there is no array”.

**Implementation 2.**

public static *int* search(*int* *n*, *int*[] *a*, *int* *x*) {

*int* left = 0;

*int* right = *n*;

    while (left + 1 < right) {

*int* mid = (left + right) / 2;

        if (*a*[mid] >= *x*)

            left = mid;

        else

            right = mid;

    }

    if (left + 1 == right && *a*[left] == *x*)

        return left;

    else

        return -1;

}

So it seems that this implementation tries to set new rules “sub array” bound setting. But when it makes the decision to move one of the sub array endpoint pointers it moves the wrong pointer to the midpoint. A case when such behavior is visible would be:

*int* n = 6;

*int*[] arr = new *int*[] { 1, 2, 7, 8, 9, 10 };

*int* x = 9;

During the first “shrinking” of the array, the midpoint is at index 3, the value at which is 8. 8 is smaller than 9, so the algorithm should set the left pointer to mid, but clearly, with given inputs the else block will be executed which leads to the incorrect answer being produced.

**Implementation 3.**

public static *int* search(*int* *n*, *int*[] *a*, *int* *x*) {

    if (*n* == 0)

        return -1;

*int* left = 0;

*int* right = *n* - 1;

    while (right - left > 0) {

*int* mid = (left + right) / 2;

        if (*a*[mid] == *x*) {

            left = mid;

            break;

        }

        if (*a*[mid] > *x*)

            left = mid + 1;

        else

            right = mid - 1;

    }

    if (*a*[left] == *x*)

        return left;

    else

        return -1;

}

**Implementation 4.**

public static *int* search(*int* *n*, *int*[] *a*, *int* *x*) {

    if (*n* == 0)

        return -1;

*int* left = 0;

*int* right = *n* - 1;

    while (right - left > 1) {

*int* mid = (left + right + 1) / 2;

        if (*a*[mid] == *x*) {

            left = mid;

            right = mid;

            break;

        }

        if (*a*[mid] > *x*)

            left = mid + 1;

        else

            right = mid - 1;

    }

    if (*a*[right] == *x*)

        return right;

    else

        return -1;

}