CSP2348-5243 Data Structures
Tutorial 03: Array Algorithm Analysis

Related Objectives from Unit Outline:

The material in this module provides a brief introduction to:

- Describe (arrays, linked lists, binary trees, and hash tables) data structures and analyse the complexity and performance of their associated algorithms.

Objectives:

- To become familiar with the general properties of arrays;
- To become familiar with the specific properties of Java arrays;
- To demonstrate the awareness of the principles of algorithms in array insertion, deletion, searching, and merging.

Requirements:

From Blackboard download and unzip the following file *csp2348_5243_workshop03.zip*, which contains source code required for this worksheet.

Workshop Activity:

Complete the following.

Exercise 1:

Using the BigO notation, analyse the array ascending algorithm outlined below. Recall, that the ascending method determines whether the array's elements are in the ascending order with/without uniqueness. Use the following tabular format to help you calculate the algorithm time complexity.

code	##	cost

int ascending(int[] array, boolean unique) {
  if (array==null) return -1;
  int n=array.length;
  for (i=1; i<n; i++){
  if (array[i-1]>array[i]) return i;
  if (unique && array[i-1]==array[i]) return i;
  }
  return n;
}
	01
	02
	03
	04
	05
	06
	07
	08
	09
		- O(1)
		O(1)
		O(1)
		O(1)×#e
		O(1)×#e
		O(1)×#e
		-
		O(1)
		-

loop-control=1,2,3,…,n-1 #e = number of executions ≈ O(?) maximum cost = O(?)		

Exercise 2:

Using the BigO notation, analyse the array *searchLinear()* algorithm outlined below. Recall, that the *searchLinear()* method determines whether the item is present in the array, which is **assumed to be in ascending order**. The *int* result is the item's position with respect to the array's values. Use the following tabular format to help you calculate the algorithm time complexity.

code	##	cost

int searchLinear(int[] array, int item) {
if (array==null) return -1;
for (int i=0; i<array.length; i++)
if (array[i]>=item)
return i;
return array.length;
}
	01
	02
	03
	04
	05
	06
		-
		O(?)
		O(?)
		O(?)
		O(?)
		O(?)

loop-control = 1, 2, 3,…, n-1 #e = number of executions ≈ O(?) maximum cost = O(?)		

Exercise 3:

Using the Big-O notation, analyse the array merging algorithm (also see ppt slides Module 03). Hint: To help you calculate the time complexity of each expression (line of code) adopt the tabular format you have used for exercise 1 and 2.

```
public void merge(int[] a1, int l1, int r1, int[] a2, int l2, int r2, int[] a3, int l3) {
    //merge existing a1[l1,...,r1] and existing a2[l2,...,r2]
    //into existing a3[l3], where a1 and a2 are sorted
    int i = l1, j = l2, k = l3;
    while (i <= r1 && j <= r2) {
        if (a1[i] <= a2[j]) {
            a3[k++] = a1[i++];
        } else {
            a3[k++] = a2[j++];
        }
    }
    while (i <= r1) {
        a3[k++] = a1[i++];
    }
    while (j <= r2) {
        a3[k++] = a2[j++];
    }
}
```

Exercise 4:

Follow the steps outlined below to create a new project called *workshop03*:

- Start NetBeans IDE and create a new project called *workshop03*.
- Download from Blackboard and unzip the file called *csp2345_243_workshop03.zip*. This contains the source code of the java classes required for this workshop.
- In the *workshop03* project create 4 classes with the source code contained in *csp2345_243_workshop03.zip*.
- Study the code, compile and execute the *ArrayMerge* class.

Exercise 5:

- Follow similar steps as in Exercise 4 to study the code, compile and execute the *ArraySearch* class.
- Note how to construct array objects;
- Observe the general properties of arrays showed by the executed results.

Exercise 6:

- Follow similar steps as in Exercise 4 to study the code, compile and execute the `LinearSearch` class.
- Run this program to test pre-set targets;
- Explain the executed results according to the principles of the linear search algorithm;
- Modify the code to test different ***targets***.

Exercise 7:

- Follow similar steps as in Exercise 4 to study the code, compile and execute the `BinarySearch` class.
- Run this program to test pre-set targets;
- Explain the executed results according to the principles of the binary search algorithm;
- Modify the code to test different ***targets***.