ECS529U Algorithms and Data Structures **Week 4 Lab sheet**

This fourth lab gets you to work with recursive algorithms and also practically compare the efficiency of more sorting algorithms by testing them on randomly generated arrays.

Marks (max 5): Questions 1-3: 1 each | Questions 4-7: 0.5 each

Question 1

Write a Python function

```
def isSubArray(A,B)
```

which takes two arrays and returns `True` if the first array is a (contiguous) subarray of the second array, otherwise it returns `False`. You may solve this problem using recursion or iteration or a mixture of recursion and iteration.

For an array to be a subarray of another, it must occur entirely within the other one without other elements in between. For example:

- [31,7,25] is a subarray of [10,20,26,31,7,25,40,9]
- [26,31,25,40] is not a subarray of [10,20,26,31,7,25,40,9]

Hint: A good way of solving this problem is to make use of an auxiliary function that takes two arrays and returns True if the contents of the first array occur at the front of the second array, otherwise it returns False. Then, `A` is a subarray of `B` if it occurs at the front of `B`, or at the front of `B[1:]`, or at the front of `B[2:]`, etc.

Question 2

Write a Python function

```
def timesOccursIn(k,A)
```

which which takes an integer and an array of integers and returns the number of times the integer occurs in the array. You must use recursion and no loops for this question.

For example, if its arguments are 5 and [1,2,5,3,6,5,3,5,5,4] the function should return 4.

Hint: Suppose `A` is not empty. If the first element of `A` is in fact `k`, the number of times that `k` occurs in `A` is “1 + the number of times it occurs in `A[1:]`”. Otherwise, it is the same as the number of times it occurs in `A[1:]`. On the other hand, if `A` is the empty array `[]` then `k` occurs 0 times in it.

Question 3

Write a Python function

```
def multAll(k,A)
```

which takes an integer `k` and an array `A` of integers and changes `A` by multiplying each of its elements by `k`. You must use recursion and no loops for this question.

For example, if it takes the array [5,12,31,7,25] and the integer 10, it changes the actual array so that it becomes [50,120,310,70,250].

Hint: The following "solution" will not work, as each recursive call creates a new copy of `A` so the original `A` is not changed.

```
def multAllNope(k,A):
    if A == []: return
    A[0] = A[0]*k
    return multAllNope(k,A[1:])
```

Instead, the way (i.e. the trick) to do this is to define an auxiliary function `multAllRec(k,A,i)` which multiplies `A[i],A[i+1],...,A[len(A)-1]` with `k`. This function can then be defined with recursion.

Question 4

Using recursion, write a Python function

```
def multAll2(k,A)
```

which performs the same task as `multAll`, but does it by creating and returning a new array with the multiplied values rather than changing the array passed to it.

Question 5

Using recursion, write a Python function

```
def binSearch2(A,k)
```

which searches for `k` in `A` using binary search (see Lecture 1).

Question 6

Using recursion, write a Python function

```
def filterByMark(A,f)
```

which takes an array `A` of `Script` objects and an integer `f` and returns an array of integers which consists of the `id`'s of each `Script` in `A` whose mark is equal to or greater than `f`.

Question 7

Using your solution to Question 4 from last week's lab (week 3), compare the four sorting functions we saw (selection, insertion, merge and quick sort) using random arrays and fill in the table below. For each array length, produce 3 random arrays to test the sorting functions and fill in the corresponding cell the mean running time (in seconds) for each function. You can copy and paste the sorting code from the lecture slides.

array length	100	1000	1000	10000	100000
selection sort sorting time (s)					
insertion sort sorting time (s)					
merge sort sorting time (s)					
quicksort sorting time (s)					