# Homework 2

*Date: 2023-06-25 17:30*
*type:* #Homework
*Class*: Com s 311 - Algorithm Analysis & Design

*Basic Data Structures*

## Question 1

Consider an array $A$ of 0's and 1's such that the 0's appear in the array before all the 1's. The objective is to find the largest index $i$ such that $A[i] = 0$. Write an algorithm that takes as input $A$ and outputs $i$. Derive the runtime of your algorithm.

```
foo(arr[] A){
	int i=0
	for i in range [i...A.length] {
		if A[i] = 1
			break
	}
	return i;
}
```

The algorithm does a single loop through the array with a constant number of operations done inside the loop:

$$T(n) = \sum_{i=0}^{n} c_1 = c_1 * n \implies O(n)$$

## Question 2

Consider an array $A$ of integers. Write an algorithm that outputs the length of the longest subarray when the sum of the elements in the subarray is equal to 0. For instance, for the array $A = \{12, 11, -2, 1, 7, -15, -2, 3, -3, 10\}$ the output of your algorithm should be 8, for the array $A = \{12, 1, 0, -2\}$ the output should be 1, and for the array $A = \{13, 1, -2\}$ the output should be 0. Derive the runtime of your algorithm.

Using a brute force algorithm, as can do the following:

```
foo(arr[] A){
	subLength = 0
	for i=0 in range [0...A.length]
```

```
                sum = 0
                for j=0 in range [i...A.length]
                        sum += arr[j]
                        if sum == 0
                                subLength = max(subLength, j-i+1)
        return subLength;
}
```

We are starting at the beginning of the array, then checking with each iteration if the numbers afterwords can form a sum of 0, and then checking to see if that length is larger than the one we already found.
We have a loop within a loop, where each loop depends on the input size of the array, and where the 2nd loop shrinks in accordance to the first:

$$T(n) = \sum_{i=0}^{n}\sum_{j=i}^{n} c_1 = \underbrace{\sum_{i=0}^{n} c*(n-i)}_{\text{sum of constant} = c*\text{upper bound}} = c*(n-i)*n = cn - ci*n = cn^2 - cin \implies O(n^2)$$

# Question 3

Given an arary $A$ of integers. Write an algorithm that outputs the k-th largest element. Derive the runtime of your algorithm.

sort the list, then index the k-th element:

```language-psuedo
foo(arr[] A, int k){
        for i=0 to n-1
                find the smallest element in A[i..n-1] //loop
                exchange it with A[i]
        return A[-k]
}
```

This algorithm performs a basic *selection sort* by doing almost exactly what we did in the previous question except instead of adding to a sum, we are checking a conditional if it is less than the current element at $i$. Selection sort is known to run in $O(n^2)$, and we just perform a random access of the $k$-th largest element, except it is negative to start from the back (it is sorted in ascending order -> 1st largest number would be at the end of the list, negative indexing starts from the back):

$$T(n) = \sum_{i=0}^{n-1}\sum_{j=i+1}^{n-1} c_1 = \underbrace{\sum_{i=0}^{n-1} c_1*(n-i)}_{c_1*(n-1-i+1)=(n-i)} = c_1*(n-i)*(n-1) \to c*(n^2 - ni - n + i) \implies O(n^2)$$

# Question 4

In the lectures, we studied binary heaps. A min-Heap can be visualized as a binary tree of height with each node having at most two children with the property that value of a node is at most the value of its children. Such heap containing $n$ elements can be represented (stored) as an array with the property:

$$\forall i[(2i \leq n \implies a[i] \leq a[2i]) \cap (2i+1 \leq n \implies a[i] \leq a[2i+1])]$$

Suppose that would like to construct a $l$ min Heap: each node has at most $l$ children and the value of a node is at most the value of its children.

1. How do you represent $l$ min heap as an array? Write the property of the array.

an $l$ min heap would have at most $l$ children, where every child is less than its parent:

$$\forall i[(li \leq n \implies a[i] \leq a[li]) \cap (li+1 \leq n \implies a[i] \leq a[li+1]) \cap (li+2 \leq n \implies a[i] \leq a[li+2]) \cap \ldots$$
$$\cap (li+(l-1) \leq n \implies a[i] \leq a[li+(l-1)])]$$

1. What is the height of the $l$ heap (when visualized as a tree) consisting of $n$ elements in terms of $l$ and $n$? Assume that the height of a heap-tree with just 1 element is 1.

- The first level would just have 1 parent node
- The second level consists of $l$ nodes, so the first+second levels = $l+1$
- The third level consists of $l^2$ nodes, so adding to the first 2 level = $l^2+l+1$
- The amount will keep increasing exponentially, but the amount of levels is proportional to log of the amount of nodes, thus

The height = $\log_l n$

1. Describe an algorithm to remove the smallest element from an $l$ heap containing $n$ elements. What is the asymptotic run time of the algorithm (in terms of $n$ and $l$ ). Part of the grade depends on the runtime.

Because it is a min heap, the smallest element is at the root, so:

1. delete the root and swap with the most recently added element.
2. We can then continue to percolate down the node until the smallest node is back to the root of the tree:
    1. Compare the root to its children by taking the Min of all the children: Min = min(children(currentNode))
3. set current node to the one we just swapped
4. repeat steps 2-3 until the tree is back to maintaining the heap property
5. end

Because each level needs to be processed each time to ensure the whole tree maintains the property, the time would be the height of the tree, which is $O(\log_l n)$