პროგრამირების აბსტრაქციები
სემინარის ამოცანები #13

1 ჭეშმარიტი წინადადებების დათვლა

ჩვენი მიზანია დავითვალოთ რამდენი ლოგიკური გამოსახულება შეიძლება შევადგინოთ მუდმივებით "True" და "False" და n ცალი ლოგიკური ოპერატორით (თითო ოპერატორის არის ან "And" ან "Or"), ფრჩხილების გამოყენების გარეშე, ისე რომ გამოსახულების გამოთვლისას პასუხი იყოს "True". გაითვალისწინეთ რომ "And" ოპერატორს უფრო მაღალი პრიორიტეტი აქვს. რეკურსიისას ერთიდაიგივე შეკითხვაზე პასუხის გამოთვლა ბევრჯერ რომ არ დაგვჭირდეს, ერთხელ გამოთვლილი პასუხი დაიმახსოვრეთ ხოლმე და მეორედ თუ იგივეს გამოთვლა მოგიწევთ, დამახსოვრებული პასუხი გამოიყენეთ.

```
#include "map.h"

int countTrueSentences(int n, Map<int, int>& storedValues) {
    if (storedValues.containsKey(n))
        return storedValues[n];

    // case when there is no OR operator
    int result = 1;

    // case with at least one OR operator
    // loop over the last appearance of OR operator
    for (int i = 1; i <= n; i++) {
        result += (1 << (2*n - 2*i + 1));
        result += ((1 << i) - 1)*countTrueSentences(n - i, storedValues);
    }

    storedValues[n] = result;
    return result;
}
```

2 დალაგების ალგორითმების შედარება

ამ ამოცანაში ერთმანეთს უნდა შევადაროთ დალაგების ალგორითმები "selection sort" და "insertion sort" რამდენიმე რეალურ მონაცემზე. ამისთვის, ორივე ალგორითმისთვის დაწერეთ ფუნქცია, რომელიც (მისამართით) გადაცემულ მთელი რიცხვების ვექტორს დაალაგებს და თან დაითვლის ამ დალაგებისას შესრულებული ოპერაციების რაოდენობას, რომელსაც ის პასუხად დააბრუნებს.

```
#include "vector.h"

void swap(int& a, int& b) {
    int tmp = a;
    a = b;
    b = tmp;
}

int selectionSortWithCompareCount(Vector<int>& vec) {
    int result = 0;
    for (int i = 0; i < vec.size(); i++) {
        int minIndex = i;
        for (int j = i + 1; j < vec.size(); j++) {
            if (vec[j] < vec[minIndex]) minIndex = j;
            result++;
        }
        swap(vec[i], vec[minIndex]);
    }
    return result;
}

int insertionSortWithCompareCount(Vector<int>& vec) {
    int result = 0;
    for (int i = 1; i < vec.size(); i++) {
        int currentPosition = i;
        while (currentPosition > 0) {
            result++;
            if (vec[currentPosition] < vec[currentPosition - 1]) {
                swap(vec[currentPosition], vec[currentPosition - 1]);
                currentPosition--;
            } else {
                break;
            }
        }
    }
    return result;
}
```