Практика 3

1. Написать функцию, осуществлющую проверку, является ли заданное целое число n простым; оценка сложности должна быть $O\sqrt(n)$.

```julia
# № 1 ----------------------------------

function isprime(n::IntType) where IntType <: Integer # является ли заданное число простым
    for d in 2:IntType(ceil(sqrt(n)))
        if n % d == 0
            return false
        end
    end
    return true
end
```

2. Написать функцию, реализующую "решето" Эратосфена, т.е. возвращающую ветор всех простых чисел, не превосходящих заданное число n.

```julia
# № 2 ----------------------------------

function eratosphenes_sieve(n::Integer)
    prime_indexes = ones(Bool, n)
    prime_indexes[begin] = false
    i = 2
    prime_indexes[i^2:i:n] .= false # - четные вычеркнуты
    i=3
    #ИНВАРИАНТ: i - простое нечетное
    while i <= n
        prime_indexes[i^2:2i:n] .= false
        # т.к. i^2 - нечетное, то шаг тут можно взять равным 2i, т.к. нечетное+нечетное=четное, а все четные уже вычеркнуты
        i+=1
        while i <= n && prime_indexes[i] == false
            i+=1
        end
        # i - очередное простое (первое не вычеркунутое)
    end
    return findall(prime_indexes)
end
```

3. Написать функцию, осуществляющую разложение заданное целое число на степени его простых делителей.

```
function factorize(n::IntType) where IntType <: Integer
    list = NamedTuple{(:div, :deg), Tuple{IntType, IntType}}[]
    for p in eratosphenes_sieve(Int(ceil(n/2)))
        k = degree(n, p) # кратность делителя
        if k > 0
            push!(list, (div=p, deg=k))
        end
    end
    return list
end

function degree(n, p) # кратность делителя `p` числа `n`
    k=0
    n, r = divrem(n,p)
    while n > 0 && r == 0
        k += 1
        n, r = divrem(n,p)
    end
    return k
end
```

4. Реализлвать функцию, осуществляющую вычисление сренего квадратического отклонения (от среднего значения) заданного числового массива за один проход этого массива.

```
function meanstd(aaa)
    T = eltype(aaa)
    n = 0; s¹ = zero(T); s² = zero(T)
    for a ∈ aaa
        n += 1; s¹ .+= a; s² += a*a
    end
    mean = s¹ ./ n
    return mean, sqrt(s²/n - mean*mean)
end
```

5. Написать функции, позволяющие взаимное преобразование различных способов представления корневых деревьев.
   Рассмотреть следующие способы представления корневого дерева

- с помощью вложенных векторов;

- с помощью списка смежностей, представленного словарём (Dict{Int, Vector{Union{Int, Nothing}})
- с помощью связанных структур.

6. Для дерева, представленного вложенными векторами, реализовать следующие функции

- функцию, возвращающую высоту дерева
- функцию, возвращающую, число листьв дерева
- функцию, возвращающую число всех вершин дерева
- функцию, возвращающую наибольшую валентность по выходу вершин дерева
- функцию, возвращающую среднюю длину пути к вершинам дерева

Все эти функции следует реализовать рекурсивно на основе соответсвующего рекурреного соотношения для дерева. А именно, если $H_1, \dots, H_n$ - это высоты смежных с корнем поддеревьев, то высота самого дерева $H = \max H_1, \dots, H_n + 1$ (высотта тривиального дерева, состоящего только из корня, по определению равна 1).

Если $N_1, \dots, N_n$ - это числа листьев в смежных с корнем поддеревьях, то число листьев всего дерева $N = N_1 + \dots + N_n$, если дерево не тривиальное, или $N = 1$, если дерево тривиальное.

Если $N_1, \dots, N_n$ - это числа вершин в смежных с корнем поддеревьях, то число вершин всего дерева $N = N_1 + \dots + N_n + 1$.

Если $P_1, \dots, P_n$ - это наибольшие валентности вершин в смежных с корнем поддеревьях, то наибольшая валентность всего всего дерева $P = \max P_1, \dots, P_n, n$.

Если $S_1, \dots, S_n$ - это суммарные длины путей от корня каждого поддерева до всех эго вершин, то суммарная длина путей от коня дерева, до всех его вершин равна $S = S_1 + \dots + S_n + N$, где N-число вершин в дереве. Поэтому искомая средняядлина путей к вершинам дерева есть $L = S/N$.