**Федеральное государственное автономное образовательное учреждение
высшего образования
"Национальный исследовательский университет "Высшая школа экономики"**

Московский институт электроники и математики им. А.Н. Тихонова НИУ ВШЭ
Департамент компьютерной инженерии

Курс: Алгоритмизация и программирование

Раздел	**Макс. оценка**	**Итоговая оценка**
Работа программы	1	
Тесты	1	
Правильность алгоритма	3	
Ответы на вопросы	2	
Дополнительное задание	3	
Итого	10	

Отчет по лабораторной работе № 11

Студент: Чапайкин Арсений Георгиевич

Группа: БИВ242

Вариант: № 171 (7, 12)

Руководитель: Елисеенко А.М.

Оценка:

Дата сдачи:

МОСКВА 2024

Листинг программы

Задание 1

```cpp
#include <iostream>
#include <vector>
#include <chrono>

template<typename Func, typename... Args>
auto measure_time(Func&& func, Args&&... args)
requires requires(Func function) { function(std::forward<Args>(args)...); }
{
    auto start = std::chrono::high_resolution_clock::now();
    auto result = std::forward<Func>(func)(std::forward<Args>(args)...);
    auto end = std::chrono::high_resolution_clock::now();

    std::chrono::duration<double> duration = end - start;
    return std::make_pair(result, duration.count());
}

long long recursive(int n) {
    switch (n) {
        case 0:
            return 0ll;
        case 1:
            return 1ll;
        default:
            long long result = 0;
            for (int i = 2; i <= n; i++) {
                result += recursive(n / i);
            }
            return result;
    }
}

long long iterative(int n) {
    std::vector<long long> cache(n + 1);
    cache[1] = 1;

    for (int i = 2; i <= n; i++) {
        for (int j = 2; j <= n; j++) {
            cache[i] += cache[i / j];
        }
    }

    return cache[n];
}

int main() {
    int n;
    std::cin >> n;

    auto [res_rec, time_rec] = measure_time(recursive, n);
    auto [res_it, time_it] = measure_time(iterative, n);

    std::cout << std::fixed << std::setprecision(8);
    std::cout << res_rec << ' ' << time_rec << '\n';
    std::cout << res_it << ' ' << time_it << '\n';
}
```

Задание 2

```cpp
#include <iostream>
#include <cassert>
#include <limits>

template <
    typename T,
    typename Allocator = std::allocator<T>
> class BinaryTree {
    struct BaseNode {
        BaseNode* left = nullptr;
        BaseNode* right = nullptr;
        BaseNode* parent = nullptr;
    };

    struct Node : BaseNode {
        T value;

        template <typename... Args>
        Node(BaseNode* left = nullptr, BaseNode* right = nullptr,
             BaseNode* parent = nullptr, Args&&... args)
                : BaseNode(left, right, parent), value(std::forward<Args>(args)...)
        {}
    };

    template <bool IsConst>
    class BaseIterator {
    public:
        using value_type = T;
        using pointer = std::conditional_t<IsConst, const T*, T*>;
        using reference = std::conditional_t<IsConst, const T&, T&>;

        BaseIterator() = default;
        BaseIterator(const BaseIterator&) = default;
        BaseIterator& operator=(const BaseIterator&) = default;

        explicit BaseIterator(BaseNode* node, BaseNode* sentinel)
                : current(node), sentinel(sentinel)
        {}

        bool operator==(const BaseIterator& other) const {
            return current == other.current;
        }

        bool operator!=(const BaseIterator& other) const {
            return current != other.current;
        }

        BaseIterator operator++(int) {
            auto temp = *this;
            ++(*this);
            return temp;
        }

        BaseIterator& operator++() {
            current = find_next(current);
            return *this;
        }

```

```
        reference operator*() const {
            return static_cast<Node*>(current)->value;
        }

        pointer operator->() const {
            return &(static_cast<Node*>(current)->value);
        }

    private:
        BaseNode* find_next(BaseNode* node) {
            if (node->right != sentinel && node->right != nullptr) {
                node = node->right;
                while (node->left != sentinel && node->left != nullptr) {
                    node = node->left;
                }
                return node;
            } else {
                BaseNode* parent = node->parent;
                while (parent != sentinel && node == parent->right) {
                    node = parent;
                    parent = parent->parent;
                }
                return parent;
            }
        }

        BaseNode* current;
        BaseNode* sentinel;
    };

    BaseNode sentinel_node;

public:
    using iterator = BaseIterator<false>;
    using const_iterator = BaseIterator<true>;
    using node_allocator = typename std::allocator_traits<Allocator>::template
        rebind_alloc<Node>;

    BinaryTree()
            : sentinel_node{&sentinel_node, &sentinel_node, &sentinel_node}
    {}

    ~BinaryTree() {
        clear();
    }

    iterator end() {
        return iterator(&sentinel_node, &sentinel_node);
    }

    iterator begin() {
        return iterator(sentinel_node.left, &sentinel_node);
    }

    const_iterator end() const {
        return const_iterator(&sentinel_node, &sentinel_node);
    }

    const_iterator begin() const {
        return const_iterator(sentinel_node.left, &sentinel_node);
```

```cpp
    }

    bool empty() const {
        return sentinel_node.parent == &sentinel_node;
    }

    iterator find(const T& value) {
        assert(!empty());

        BaseNode* parent_or_self = find_place(value);
        if (static_cast<Node*>(parent_or_self)->value == value) {
            return iterator(parent_or_self, &sentinel_node);
        } else {
            return end();
        }
    }

    template <typename U>
    requires requires(T lhs, U rhs) { lhs < rhs; lhs - rhs; }
    iterator find_closest(const U& value) {
        assert(!empty());

        BaseNode* parent_or_self = find_place(value);
        iterator next = iterator(parent_or_self, &sentinel_node);
        next++;

        if (next != end()
                && *next - value <= value - static_cast<Node*>(parent_or_self)->
                    value) {
            return next;
        } else {
            return iterator(parent_or_self, &sentinel_node);
        }
    }

    std::pair<iterator, bool> insert(const T& value) {
        BaseNode* parent_or_self = find_place(value);
        Node* new_node;

        if (parent_or_self == &sentinel_node) {
            new_node = create_node(&sentinel_node, &sentinel_node, &sentinel_node,
                value);
            sentinel_node.left = sentinel_node.right = sentinel_node.parent =
                new_node;
        } else if (static_cast<Node*>(parent_or_self)->value == value) {
            return {end(), false};
        } else if (static_cast<Node*>(parent_or_self)->value < value) {
            new_node = create_node(nullptr, nullptr, parent_or_self, value);
            parent_or_self->right = new_node;
            if (sentinel_node.right == parent_or_self) {
                sentinel_node.right = new_node;
            }
        } else {
            new_node = create_node(nullptr, nullptr, parent_or_self, value);
            parent_or_self->left = new_node;
            if (sentinel_node.left == parent_or_self) {
                sentinel_node.left = new_node;
            }
        }
        return {iterator(new_node, &sentinel_node), true};
```

```
    }

private:
    node_allocator alloc;

    BaseNode* find_place(const T& value) {
        BaseNode* current = sentinel_node.parent;
        BaseNode* parent = &sentinel_node;

        while (current && current != &sentinel_node) {
            parent = current;
            if (static_cast<Node*>(current)->value < value) {
                current = current->right;
            } else if (value < static_cast<Node*>(current)->value) {
                current = current->left;
            } else {
                return current;
            }
        }

        return parent;
    }

    template <typename... Args>
    Node* create_node(BaseNode* left, BaseNode* right, BaseNode* parent, Args&&...
       args) {
        Node* new_node = std::allocator_traits<node_allocator>::allocate(alloc, 1);
        try {
            std::allocator_traits<node_allocator>::construct(
                alloc, new_node, left, right, parent, std::forward<Args>(args)...);
        } catch (...) {
            std::allocator_traits<node_allocator>::deallocate(alloc, new_node, 1);
            throw std::bad_alloc();
        }
        return new_node;
    }

    void destroy_node(Node* node) {
        std::allocator_traits<node_allocator>::destroy(alloc, node);
        std::allocator_traits<node_allocator>::deallocate(alloc, node, 1);
    }

    void clear() {
        if (sentinel_node.parent != &sentinel_node) {
            delete_subtree(sentinel_node.parent);
        }
        sentinel_node.left = &sentinel_node;
        sentinel_node.right = &sentinel_node;
        sentinel_node.parent = &sentinel_node;
    }

    void delete_subtree(BaseNode* node) {
        if (node == nullptr || node == &sentinel_node) {
            return;
        }
        delete_subtree(node->left);
        delete_subtree(node->right);
        destroy_node(static_cast<Node*>(node));
    }
};
```

```cpp

int main() {
    int n;
    std::cin >> n;

    int min = std::numeric_limits<int>::max();
    int max = std::numeric_limits<int>::min();

    BinaryTree<int> t;
    for (int i = 0; i < n; i++) {
        int x;
        std::cin >> x;

        min = std::min(min, x);
        max = std::max(max, x);

        t.insert(x);
    }

    std::cout << "Binary tree contents:\n";
    for (auto it = t.begin(); it != t.end(); ++it) {
        std::cout << *it << ' ';
    }
    std::cout << '\n';

    if (!t.empty()) {
        auto closest = t.find_closest((min + max) / 2.0);
        std::cout << "Closest to (min + max) / 2:\n";
        std::cout << *closest << '\n';
    } else {
        std::cout << "Tree is empty\n";
    }
}
```

Распечатка тестов к программе и результатов

Задание 1

Номер	Исходные данные	Результат
1	100	658 0.00003535 658 0.00007577
2	1000	33962 0.00063162 33962 0.00707651
3	10000	1817299 0.02140001 1817299 0.68015174
4	100000	97624715 1.13036776 97624715 38.13754973

Задание 2

Номер	Исходные данные	Результат
1	6 1 2 3 4 5 6	Binary tree contents: 1 2 3 4 5 6 Closest to (max + min) / 2: 4
2	8 10 11 6 2 1 7 3 4	Binary tree contents: 1 2 3 4 6 7 10 11 Closest to (max + min) / 2 6
3	4 -1 -2 0 1	Binary tree contents: -2 -1 0 1 Closest to (min + max) / 2:
4	4 -1 -1 -1 -1	Binary tree contents: -1 Closest to (min + max) / 2: -1