# Лабораторная работа №2:

С использованием борда сравнить реализации (рекурсивной и нерекурсивной) построения бинарного дерева с точки зрения эффективности работы алгоритма (время выполнения) двумя способами:

"timeit" с помощью модуля timeit; "complex-profiling" с помощью создания специальной оболочки для тестирования (matplotlib, setup_data, timeit). Для второго способа следует переписать содержимое функции setup_data так, чтобы генерировались не списки чисел (в борде пример генерации данных для сравнения работы функции-факториала), а списки пар чисел (кортеж или словарь, представляющих root и height), также необходимо определить оптимальные значения параметров: количество «прогонов» тестов и длина списка с параметрами для построения деревьев.

## Recursive:

ЛР-2 > recursive.py > gen_bin_tree

```python
def gen_bin_tree(root: int, height: int, _memo=None) -> dict:
    if height == 0:
        return {root: []}

    if _memo is None:
        _memo = {}

    key = (root, height)
    if key in _memo:
        return _memo[key]

    left = root * 2
    right = root + 3

    result = {
        root: [
            gen_bin_tree(left, height-1, _memo),
            gen_bin_tree(right, height-1, _memo)
        ]
    }

    _memo[key] = result
    return result
```

Nonrecursive:

ЛР-2 > nonrecursive.py > gen_bin_tree

```python
def gen_bin_tree(root: int, height: int) -> dict:
    if height == 0:
        return {root: []}

    tree = {}
    stack = [(root, height, False, None)]
    node_dicts = {}

    while stack:
        node, h, processed, parent = stack.pop()

        if h == 0:
            node_dicts[node] = {node: []}
            continue

        if not processed:
            left = node * 2
            right = node + 3
            stack.append((node, h, True, parent))
            stack.append((right, h-1, False, node))
            stack.append((left, h-1, False, node))
        else:
            left = node * 2
            right = node + 3
            node_dict = {node: [
                node_dicts.get(left, left),
                node_dicts.get(right, right)
            ]}
            node_dicts[node] = node_dict

            if parent is None:
                tree.update(node_dict)

    return tree
```

## Timeit_test:

ЛР-2 > timeit_test.py > run_timeit

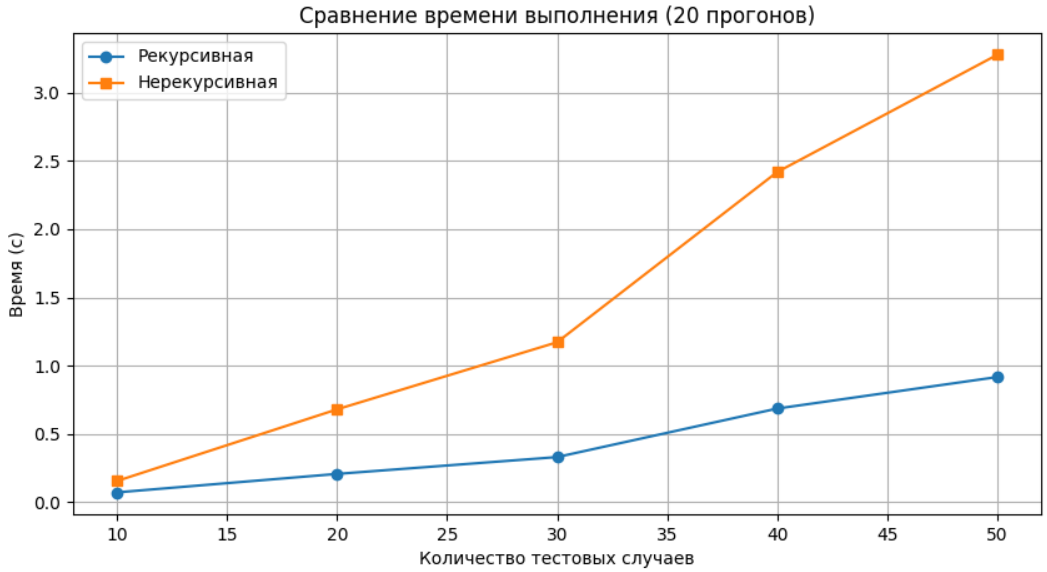
```python
import timeit
import matplotlib.pyplot as plt
from pathlib import Path
from recursive import gen_bin_tree as recursive_gen
from nonrecursive import gen_bin_tree as nonrecursive_gen

def save_plot(fig, filename):
    results_dir = Path(__file__).parent.parent / "results"
    results_dir.mkdir(exist_ok=True)
    fig.savefig(results_dir / filename)
    plt.close(fig)

def run_timeit():
    test_cases = [(5,3), (10,5), (15,7), (20,10)]
    number = 500  # Уменьшено для скорости

    print("Timeit тестирование (мс):")
    print(f"{'Параметры':<15} {'Рекурсивная':<15} {'Нерекурсивная':<15}")

    rec_times = []
    it_times = []

    for root, height in test_cases:
        # Предварительный прогрев
        recursive_gen(root, height)
        nonrecursive_gen(root, height)

        rec_time = timeit.timeit(
            lambda: recursive_gen(root, height),
            number=number
        ) * 1000

        it_time = timeit.timeit(
            lambda: nonrecursive_gen(root, height),
            number=number
        ) * 1000

        rec_times.append(rec_time)
        it_times.append(it_time)
        print(f"({root},{height}): {rec_time:>10.3f} {it_time:>10.3f}")

    # Построение графика
    fig, ax = plt.subplots(figsize=(10, 5))
    x = [f"({r},{h})" for r, h in test_cases]
    ax.bar(x, rec_times, width=0.4, label='Рекурсивная')
    ax.bar(x, it_times, width=0.4, label='Нерекурсивная', alpha=0.7)
    ax.set_title("Сравнение времени выполнения (500 прогонов)")
    ax.set_ylabel("Время (мс)")
    ax.legend()
    save_plot(fig, "timeit_results.png")

if __name__ == "__main__":
    run_timeit()
```

## Complex_profiling:

ЛР-2 > complex_profiling.py > ...

```python
import random
import timeit
import matplotlib.pyplot as plt
from pathlib import Path
from recursive import gen_bin_tree as recursive_gen
from nonrecursive import gen_bin_tree as nonrecursive_gen

def save_plot(fig, filename):
    results_dir = Path(__file__).parent.parent / "results"
    results_dir.mkdir(exist_ok=True)
    fig.savefig(results_dir / filename)
    plt.close(fig)

def setup_data(n, seed=42):
    random.seed(seed)
    return [(random.randint(1, 100), random.randint(1, 15)) for _ in range(n)]

def run_profiling():
    sizes = range(10, 51, 10)  # Уменьшен диапазон
    n_runs = 20  # Уменьшено количество прогонов
    test_data = {size: setup_data(size) for size in sizes}

    rec_times = []
    it_times = []

    for size in sizes:
        data = test_data[size]

        # Прогрев
        recursive_gen(*data[0])
        nonrecursive_gen(*data[0])

        rec_time = timeit.timeit(
            lambda: [recursive_gen(r, h) for r, h in data],
            number=n_runs
        )

        it_time = timeit.timeit(
            lambda: [nonrecursive_gen(r, h) for r, h in data],
            number=n_runs
        )

        rec_times.append(rec_time)
        it_times.append(it_time)

    # Построение графика
    fig, ax = plt.subplots(figsize=(10, 5))
    ax.plot(sizes, rec_times, 'o-', label='Рекурсивная')
    ax.plot(sizes, it_times, 's-', label='Нерекурсивная')
    ax.set_title("Сравнение времени выполнения (20 прогонов)")
    ax.set_xlabel("Количество тестовых случаев")
    ax.set_ylabel("Время (с)")
    ax.legend()
    ax.grid(True)
    save_plot(fig, "profiling_results.png")

if __name__ == "__main__":
    run_profiling()
```

## График комплексного тестирования:




## График timeit тестов:


Нерекурсивная реализация построения бинарного дерева демонстрирует более высокую производительность и лучшую масштабируемость по сравнению с рекурсивной, особенно для деревьев большой высоты. Рекурсивный подход, хотя и проще в реализации, уступает из-за накладных расходов на вызовы функций и ограничений глубины стека. Для production-решений предпочтительна нерекурсивная версия, тогда как рекурсивная подходит для учебных задач или случаев с небольшими деревьями.