EcoLab1
Создание компонента, реализующего алгоритм сортировки Tree Sort

Выполнила
Карапетян Элина
21ПИ-2

1. Алгоритм

Tree Sort является алгоритмом сортировки на основе деревьев. Он строит бинарное дерево поступательного обхода (in-order traversal) из входных данных, а затем проводит обход этого дерева, чтобы получить отсортированные данные.

Вот шаги алгоритма Tree Sort:

1. Создать пустое бинарное дерево поиска (Binary Search Tree, BST).
2. Пройтись по входному массиву элементов слева направо.
3. Вставить каждый элемент в бинарное дерево поиска, сохраняя свойство BST (т.е. левый узел меньше родительского узла, а правый узел больше родительского узла).
4. После вставки всех элементов в BST, провести поступательный обход дерева (in-order traversal) и заполните результирующий массив.
5. Возвратить отсортированный массив.

Псевдокод:

```
function TreeSort(array)
    if array is empty
        return array

    BST = empty binary search tree

    for each element in array
        Insert element into BST

    result = empty array

    Traverse BST in-order and fill the result array

    return result

function Insert(element, node)
    if node is null
        create a new node with element value
        return new node

    if element < node.value
        node.left = Insert(element, node.left)
    else
        node.right = Insert(element, node.right)

    return node

function InOrderTraversal(node, result)
    if node is not null
        InOrderTraversal(node.left, result)
        append node.value to result
        InOrderTraversal(node.right, result)
```

2. Сложность работы

Алгоритм Tree Sort имеет следующую сложность:

1. Средний случай: В среднем случае алгоритм Tree Sort имеет сложность O(n * log(n)), где n - это количество элементов в входном массиве. Это связано с тем, что каждое вставление элемента в бинарное дерево поиска требует log(n) операций, а обход дерева также требует log(n) операций на каждый элемент.

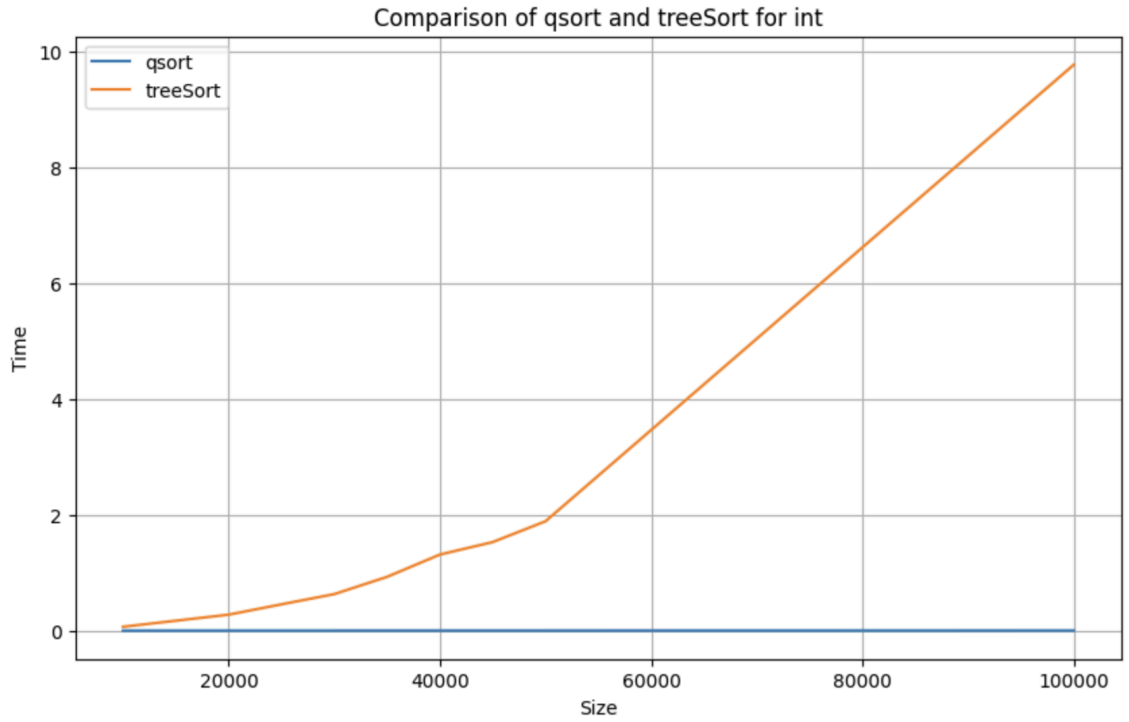
2. Худший случай: В худшем случае алгоритм Tree Sort имеет сложность O(n^2). Это происходит, когда бинарное дерево поиска становится вырожденным, т.е. превращается в список. Это может случиться, например, если входной массив уже отсортирован. В этом случае каждое вставление элемента в дерево требует n операций, а обход дерева также требует n операций на каждый элемент.

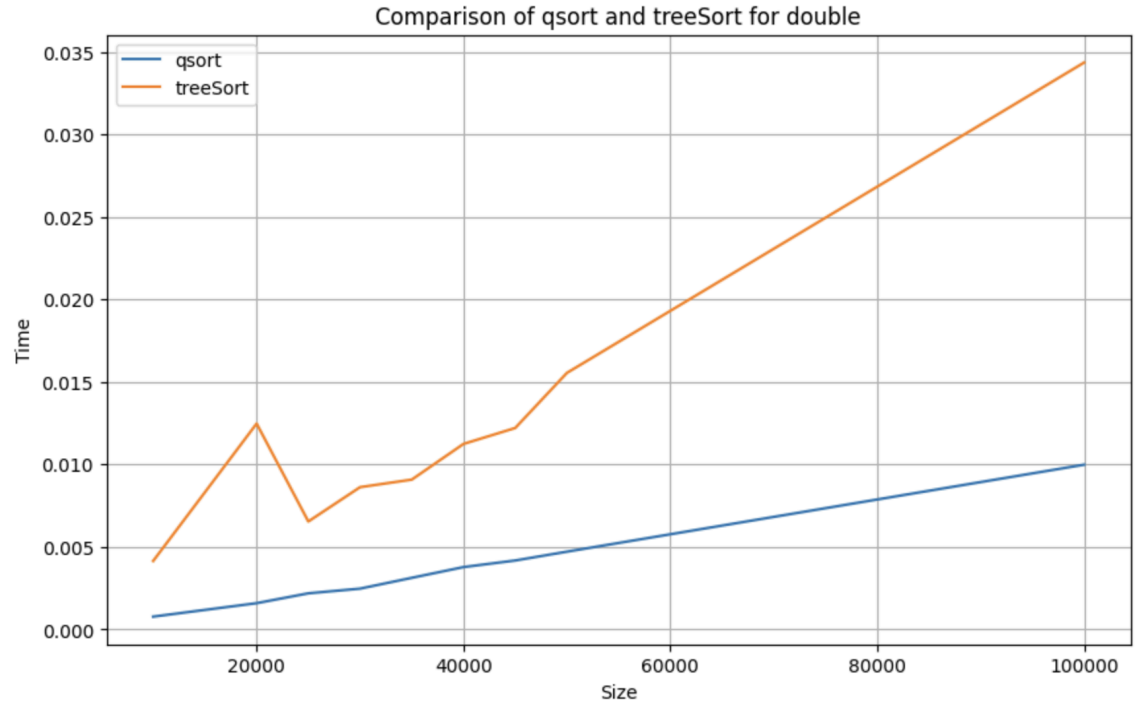
Пространственная сложность алгоритма составляет O(n), так как требуется дополнительная память для хранения дерева.

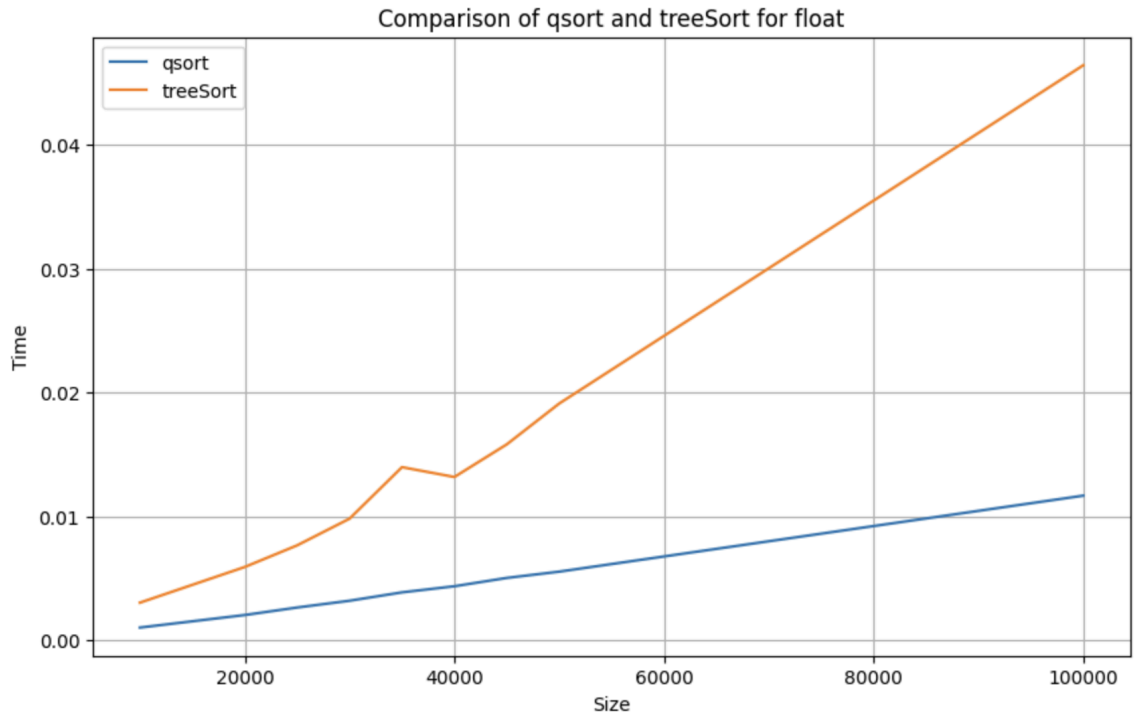
Таким образом, алгоритм Tree Sort является эффективным в среднем случае, но может быть неэффективным в худшем случае.

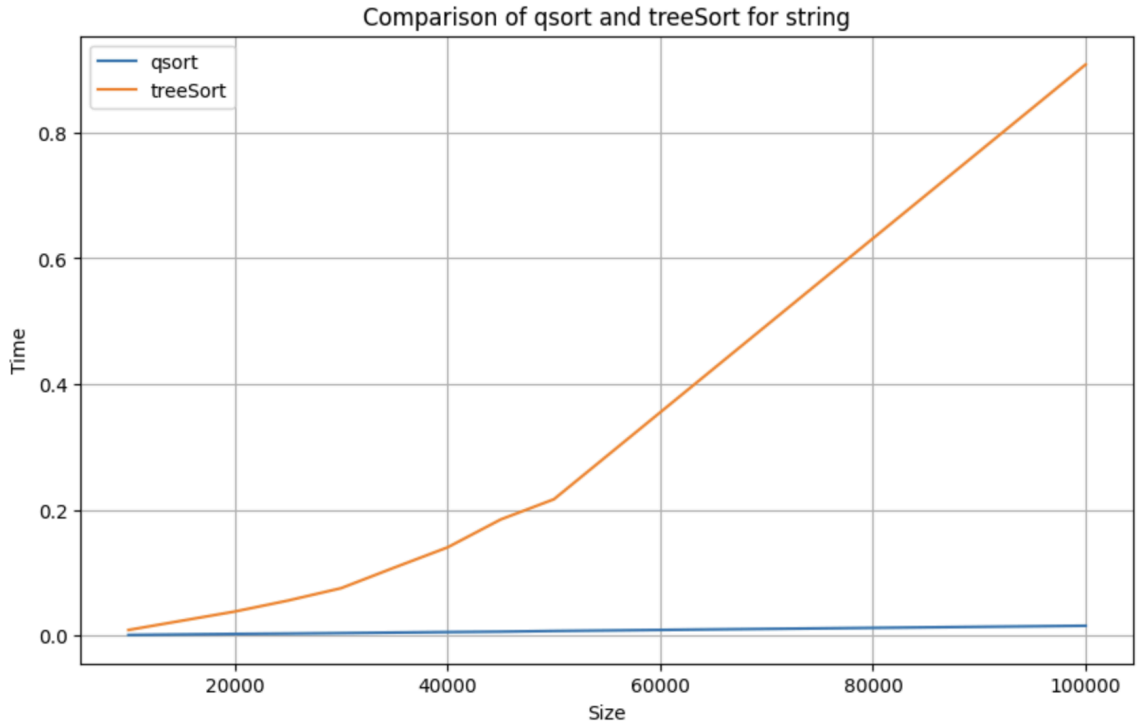
3. Результаты тестирования

Графики представлены в линейных осях









На основе предоставленной информации можно сделать следующие выводы:

1. Для int:

Для небольших размеров данных (10000 элементов) treeSort показывает лучшее время работы, чем qsort. Однако, при увеличении размера данных,

время работы treeSort начинает расти значительно быстрее, чем qsort. Это связано с тем, что при отсутствии балансировки, дерево может вырождаться в список, и операции поиска, вставки и удаления будут выполняться за линейное время O(n), а не за логарифмическое время O(log n), как в случае сбалансированного дерева.

2. Для double:

Для небольших размеров данных (10000 элементов) treeSort также показывает лучшее время работы, чем qsort. Однако, при увеличении размера данных, время работы treeSort начинает расти значительно быстрее, чем qsort. Кроме того, для double в начале графика наблюдается резкий скачок времени работы treeSort, что может быть связано с тем, что при малом количестве элементов дерево еще остается сбалансированным, но при дальнейшем увеличении количества элементов, оно начинает вырождаться в список, и время работы алгоритма резко увеличивается.

3. Для float:

Для float график похож на график для double, с той лишь разницей, что время работы treeSort растет немного медленнее, чем для double. Это может быть связано с тем, что операции сравнения для float выполняются быстрее, чем для double, и поэтому время работы алгоритма меньше.

4. Для string:

Для string график также похож на график для int, с той лишь разницей, что время работы treeSort растет немного медленнее, чем для int. Это может быть связано с тем, что операции сравнения для string выполняются медленнее, чем для int, и поэтому время работы алгоритма больше.

В целом, на основе предоставленной информации можно сделать вывод, что qsort показывает лучшее время работы, чем treeSort, за исключением небольших размеров данных. При отсутствии балансировки, treeSort может быть эффективным только для небольших данных, где время работы алгоритма не будет значительно отличаться от времени работы qsort. Однако, при работе с большими данными, отсутствие балансировки может привести к значительному ухудшению производительности treeSort по сравнению с qsort.