МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
**«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»**

АНАЛИЗ СЛОЖНОСТИ БЫСТРОЙ И ПИРАМИДАЛЬНОЙ СОРТИРОВОК

ОТЧЁТ

студента 2 курса 251 группы
направления 09.03.04 — Программная инженерия
факультета КНиИТ
Карасева Вадима Дмитриевича

Проверено:
доцент, к. ф.-м. н. ______________ М. И. Сафрончик

Саратов 2025

СОДЕРЖАНИЕ

1 Сортировки

1.1 Быстрая сортировка

```cpp
void quickSort(std::vector<int>& arr, int left, int right) {
    // Базовый случай: если подмассив содержит 0 или 1 элемент
    if (left >= right) return; // O(1)

    // 1. Выбор опорного элемента (pivot)
    int pivot = arr[(left + right) / 2]; // O(1) - средний элемент
    // Можно выбрать первый, последний или случайный элемент

    // 2. Разделение (partition)
    int i = left; // O(1)
    int j = right; // O(1)

    while (i <= j) { // O(kn), n - количество элементов, k - количество
        // Ищем элемент больше опорного слева
        while (arr[i] < pivot) i++; // O(t), t - количество элементов до

        // Ищем элемент меньше опорного справа
        while (arr[j] > pivot) j--; // O(m)

        // Если индексы не пересеклись, меняем элементы местами
        if (i <= j) { // O(t + m) = O(n)
            std::swap(arr[i], arr[j]); // O(1)
            i++; // O(1)
            j--; // O(1)
        }
    }

    // 3. Рекурсивные вызовы для подмассивов
    quickSort(arr, left, j); // O(log n)
    quickSort(arr, i, right); // O(log n)
}
```

1.2 Пирамедальная сортировка

```
void heap(vector<int>& c, int i, int n) { // сложность o(log n)
    // предполагаем, что текущий элемент - максимальный
    int ind_mx = i;
    // проверяем левого потомка
    if (i * 2 + 1 < n) {
        // если левый потомок больше текущего максимума
        if (c[ind_mx] < c[i * 2 + 1])
            ind_mx = i * 2 + 1;
    }
    // проверяем правого потомка
    if (i * 2 + 2 < n) {
        // если правый потомок больше текущего максимума
        if (c[ind_mx] < c[i * 2 + 2])
            ind_mx = i * 2 + 2;
    }
    // если максимум изменился
    if (ind_mx != i) {
        // меняем местами текущий элемент с максимальным потомком
        swap(c[i], c[ind_mx]);
        // рекурсивно вызываем heap для поддерева
        heap(c, ind_mx, n);
    }
}

// функция пирамидальной сортировки (heapsort)
void pyramid_sort(vector<int>& c, int n) {
    // построение max-heap (начиная с последнего нелистового узла)
    for (int i = n / 2 - 1; i >= 0; i--) // сложность o(n)
        heap(c, i, n); // сложность o(log n)

    // извлечение элементов из кучи по одному
    for (int i = n - 1; i >= 0; i--) { // сложность o(n)
        // перемещаем текущий корень в конец
```

```
        swap(c[0], c[i]); // сложность o(1)
        // восстанавливаем max-heap для уменьшенной кучи
        heap(c, 0, i); // сложность o(log n)
    }
}
```

2 Анализ сложности сортировок

2.1 Быстрая сортировка

Общая сложность:

Для 2-х вложенных циклов общая сложность равна: $(t + m)$ = O(n), Для внешнего $while$ сложность: O(k_n), где n – общее кол-во элементов, k – кол-во повторений внешнего цикла. Тогда общая сложность по правилу суммы: O(max(k_n,1,1)) = O(k_n).

Если массив отсортирован, то в лучшем случае мы просто пройдёмся по его n элементам. Сложность равна O($n \log n$), так как мы делим каждый раз при рекурсивном вызове массив пополам, и всего таких случаев $log_2 n$.

Согласно первой теореме:

$$T(n) = \begin{cases} c, & \text{если } n = 1, \\ 1\alpha t\left(\dfrac{n}{k}\right) + bn^{\tau}, & \text{если } n > 1. \end{cases}$$

Построим реккурентное соотношение:

$$T(n) = \begin{cases} c, & \text{если } n = 1, \\ 2T\left(\frac{n}{2}\right) + bn, & \text{если } n > 1. \end{cases}$$

Общий случай (лучший/средний)

Параметры рекурсии:

— $a = 2$ — количество подзадач,

— n/k — размер подзадачи ($k = 2$ — постоянная),

— Трудоёмкость рекурсивного перехода: $O(n)$, $\tau = 1$.

По следствию основной теоремы для лучшего случая:

$$t(n) = O(n^{\tau} \log_k n) = O(n \log n)$$

Худший случай

Если опорный элемент каждый раз оказывается минимальным или максимальным, происходит неравномерное разделение:

— Один подмассив пуст,

— Другой содержит $n - 1$ элементов.

Для отсортированного массива (прямо или обратно):

— Глубина рекурсии: n уровней,

— На каждом уровне: $O(n)$ операций.

Итоговая сложность:

$$t(n) = O(n^2)$$

2.2 Пирамедальная сортировка

Алгоритм основан на построении бинарной кучи, которая удовлетворяет следующим свойствам:

1. Каждый узел больше своих потомков.
2. Элементы на последнем уровне располагаются слева направо.

Высота бинарного дерева не превышает $\log_2 N$, поэтому временная сложность функции построения кучи:

$$T(n) = O(\log n)$$

Цикл сортировки проходит по всем n элементам, вызывая функцию построения кучи. Таким образом, общая сложность алгоритма:

$$T(n) = O(n \log n)$$

Эффективность алгоритма может снижаться, если большие значения находятся в одной части пирамиды.