

МИНОБРНАУКИ РОССИИ

*Федеральное государственное бюджетное образовательное учреждение высшего образования*

***«МИРЭА – Российский технологический университет»***

Отчет

Практическая работа №9

Дисциплина Структуры и алгоритмы обработки данных

Тема. Алгоритмы поиска в таблице (массиве)

|  |  |
| --- | --- |
| Выполнил студент | Смольников А.Б. |
|  | Фамилия И.О. |
| Группа | ИКБО-13-21 |
|  | Номер группы |

**Москва 2022**

**Задание 1**

**Вариант №20**

**Разработать программу поиска записи по ключу в таблице записей с применение двух алгоритмов линейного поиска.**

1. Таблица содержит записи, структура которых определена вариантом. *(шестизначное число – номер участка, строка – название участка)* Ключи уникальны в пределах таблицы.
2. Разработать функцию линейного поиска (метод грубой силы).
3. Разработать функцию поиска с барьером.
4. Провести практическую оценку времени выполнения алгоритмов на таблицах объемом 100, 1000, 10 000 записей.
5. Составить таблицу с указанием: времени выполнения алгоритма, его фактическую и теоретическую вычислительную сложность.
6. Сделать выводы об эффективности алгоритмов.

**Разработка алгоритма и его реализация**

Необходимо разработать два алгоритма линейного поиска в таблице по ключу.

Алгоритм: Для начала создадим таблицу из ячеек записи:

1. struct Kadastr{
2. int id;
3. string SNT\_adress;
4. };
5. struct KadastrTable{
6. Kadastr \*kadastr;
7. int size;
8. int append(Kadastr note);
9. void print();
11. Kadastr LinearSearchBarrier(int id);
12. Kadastr BruteForce(int id);
13. ~KadastrTable();
14. };

Метод грубой силы основан на грубом переборе всех значений ячеек пока не будет найдена нужная запись.

Рассмотрим код функции поиска:

Предусловие – искомый ключ id

Постусловие – искомая запись

1. Kadastr KadastrTable::BruteForce(int id){
2. for(int i = 0; i < size; i++){
3. if(kadastr[i].id == id){
4. return kadastr[i];
5. }
6. }
7. return Kadastr{-1, "NULL"};
8. }

Метод линейного поиска с барьером основан на сокращении операций в цикле за счет установки барьера

Рассмотрим код функции:

Предусловие – искомый ключ id

Постусловие – искомая запись

1. Kadastr KadastrTable::LinearSearchBarrier(int id){
2. if(size == 0) {
3. return Kadastr{0, "NO NAME"};
4. }
5. Kadastr last = kadastr[size-1];
6. kadastr[size-1].id = id;
7. int i;
8. for(i=0; kadastr[i].id!= id;++i){}
9. kadastr[size-1] = last;
10. if(i != size-1 || id == last.id){
11. return kadastr[i];
12. }
13. return Kadastr{0, "NO NAME"};
14. }

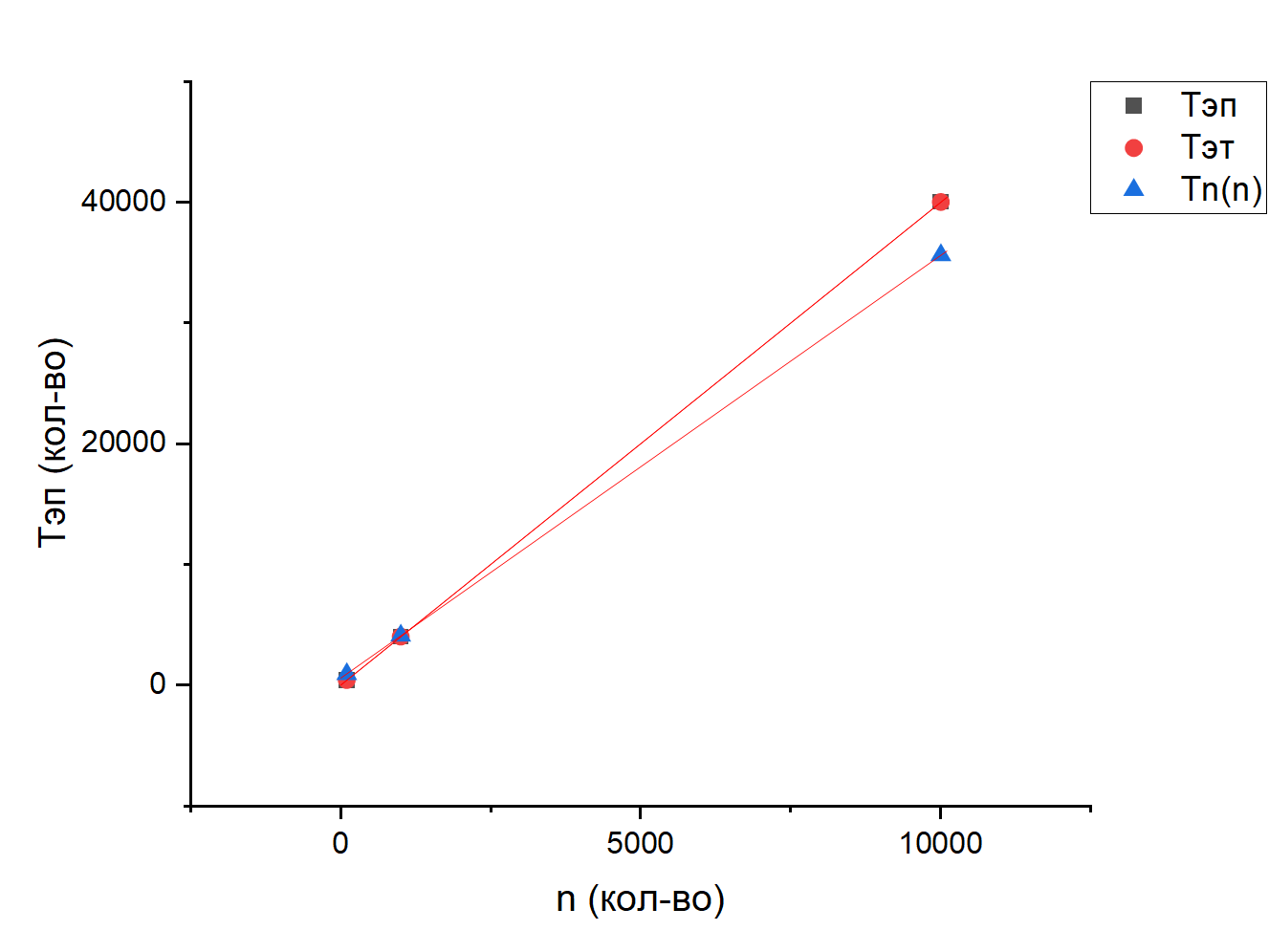
Оба этих поиска являются линейными, а значит их асимптотическая сложность:

Определим зависимость времени от количества входных данных эмпирически. В действительности эта цифра зависит от характеристик ВМ, типа операций, оптимизации самого компилятора.

Рассмотрим эмпирически количество выполняемых операций. В соответствии с кодом задачи, получим:

***Таблица 1. Сводная таблица результатов Brute force***

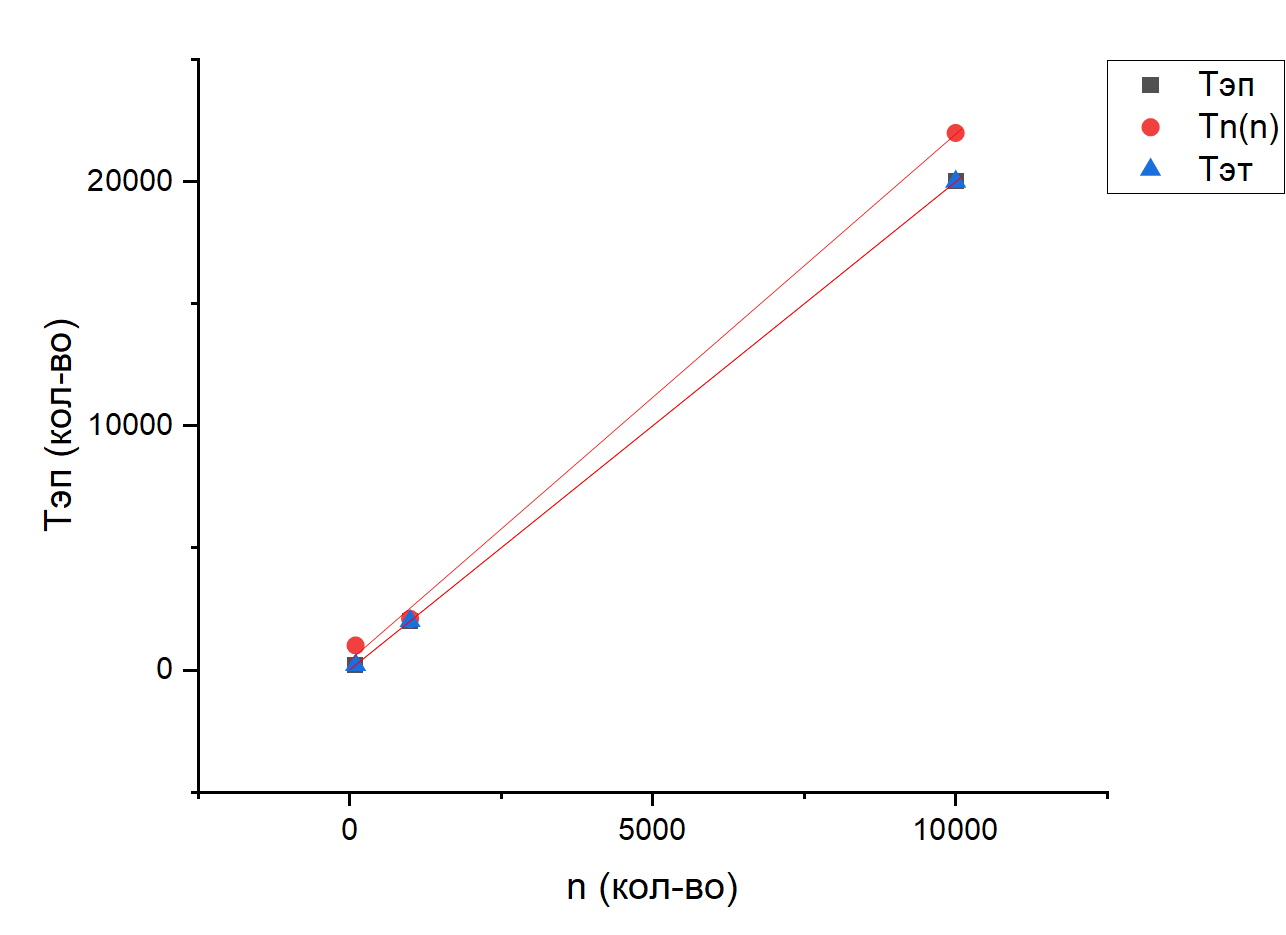
|  |  |  |  |
| --- | --- | --- | --- |
| **n** | **T(n) время,ns** | **Тэт=f(C+M)-функция, ns** | **Т**эп**=Cф+Mф- количество** |
| 100 | 900 | 399 | 399 |
| 1000 | 4100 | 3999 | 3999 |
| 10000 | 35600 | 39999 | 39999 |



Рассмотрим эмпирически количество выполняемых операций. В соответствии с кодом задачи, получим:

***Таблица 2. Сводная таблица результатов Linear Search Barrier***

|  |  |  |  |
| --- | --- | --- | --- |
| **n** | **T(n) время,ns** | **Тэт=f(C+M)-функция, ns** | **Т**эп**=Cф+Mф- количество** |
| 100 | 1000 | 206 | 206 |
| 1000 | 2100 | 2006 | 2006 |
| 10000 | 22000 | 20006 | 20006 |



Графики построены в программе OriginPro 2022. Отчетливо видна линейная зависимость всех графиков. Расхождения с теоретическим расчетом времени выполнения пренебрежимо малы.

Данные алгоритмы подходят для небольшого объема данных и очень просты в написании. Однако их асимптотику нельзя назвать подходящей для больших данных

**Задание 2. Разработать программу поиска записи по ключу в таблице записей с применением алгоритма, определенного в задании варианта.**

1. Таблица содержит записи, структура которых определена вариантом. Ключи уникальны в пределах таблицы.
2. Разработать алгоритм поиска, определенный в варианте. Реализовать алгоритм функцией.
3. Провести практическую оценку времени выполнения алгоритмов на таблицах объемом 100, 1000, 10 000 записей на случайно заполненных таблицах (худший случай). На таблицах с лучшим временем и средним.
4. Составить таблицу с указанием: времени выполнения алгоритма, его фактическую и теоретическую вычислительную сложность.

**Выводы**

Определим более эффективный алгоритм из всех предложенных:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Название алгоритма | Асимптотическая сложность алгоритма | | | |
| Наихудший случай | Наилучший случай | Средний случай | Емкостная сложность |
| Простого выбора | O(n^2) | O(n^2) | O(n^2) | O(1) |
| Сортировка бинарными включениями | O(n^2) | O(nlogn) | O(n^2) | O(1) |
| Сортировка слиянием | O(nlogn) | O(nlogn) | O(nlogn) | O(n) |

Таким образом, на большом массиве данных лучше всего себя показывает сортировка слиянием, однако для чрезвычайно больших объемов данных следует использовать внешние сортировки (k-way merge, polyphase sort). Если брать в учет труд программиста, то для большинства мелких задач будут удобны простые алгоритмы сортировок, а в погоне за производительностью следует выбрать оптимизированные сортировки, временная сложность которых описывается O(nlogn) или еще оптимальнее.