Лабораторная работа №3

Исследование алгоритмов поиска

Теоретические сведения

Множество из N элементов задано в виде некоторого массива. Задача заключается в поиске элемента, равному некоторому «аргументу поиска» x. Полученный в результате индекс i искомого элемента должен удовлетворять условию $a[i] = x$.

Прямой перебор

Если нет никакой дополнительной информации о разыскиваемых данных, то очевидный подход – простой последовательный просмотр массива с увеличением шаг за шагом той его части, где желаемого элемента не обнаружено. Такой метод называется линейным поиском или поиском перебором.

Условия окончания поиска таковы:

1. Элемент найден, т. е. $a_i = x$.

2. Весь массив пройден, и совпадения не обнаружено.

Этот алгоритм выглядит следующим образом:

```
i = 0;
while ((i < N) && (a[i] != x))
    i++;
```

Бинарный поиск

Очевидно, что поиск перебором невозможно ускорить, если нет никакой дополнительной информации о данных, среди которых идет поиск. Поиск можно сделать значительно более эффективным, если данные будут упорядочены. Наглядным примером тому служит телефонный справочник, где фамилии абонентов упорядочены по алфавиту.

Основная идея этого алгоритма – выбрать некоторый элемент, предположим a_m, и сравнить его с аргументом поиска x. Если он равен x, то поиск заканчивается. Если он меньше x, то заключаем, что все элементы с индексами, меньшими или равными m, можно исключить из дальнейшего поиска.

Если же он больше x, то исключаются индексы больше или равные m. Это соображение приводит к следующему алгоритму, называемому бинарный поиск, двоичный поиск или же поиск делением пополам.

Здесь две индексные переменные L и R отмечают соответственно левый и правый концы секции массива a, где еще может быть обнаружен требуемый элемент.

```
L = 0;
R = N - 1;
Found = 0;
while ((L <= R) && (Found == 0))
{
    m = любое значение между L и R;
    if (a[m] == x)
        Found = 1;
    else
        if (a[m] < x)
            L = m + 1;
        else
            R = m - 1;
}
```

Выбор m совершенно произволен в том смысле, корректность алгоритма от него не зависит. Однако на его эффективность выбор m влияет. Ясно, что основной задачей является исключение на каждом шаге как можно большего числа элементов массива. Оптимальным решением будет выбор среднего элемента, так как в любом случае будет исключена половина массива.

В результате максимальное число сравнений будет равно $\log N$, округленному до ближайшего целого. Таким образом, приведенный алгоритм

существенно выигрывает по сравнению с поиском перебором, ожидаемое число сравнений которого равно $N / 2$

```
L = 0; R = N - 1;
while (L < R)
{
    m = (L + R) / 2;
    if (a[m] < x)
        L = m + 1;
    else
        R = m;
}
```

Задание на лабораторную работу

1. Реализовать методы поиска данных в массиве:
    a. Реализовать метод прямого поиска.
    b. Реализовать метод бинарного поиска по массиву (сортировку массива производить любым алгоритмом).
    c. Реализовать метод бинарного поиска с использованием бинарного дерева поиска
2. Реализовать управляющую программу(ы), включающую:
    a. ввод исходных данных: из файла, с консоли, генерацией случайных чисел (способа ввода данных предусмотреть в программе путем ввода выбора пользователя, при этом размер массива тоже указывается во время выполнения программы);
    b. ввод на экран исходных данных;
    c. вывод на экран результата работы;
    d. замер времени выполнения сортировки.
3. Выполнить исследование реализованных трех алгоритмах на разных коллекциях:
    a. Выполнить замеры времени для коллекций размера 10, 100, 1 000, 10 000, 10 000, 1 000 000 000 элементов.
    b. Заполнить таблицы 1 для каждого анализируемого алгоритма (всего три таблицы).
    c. Сделать выводы по таблицам и графику.

Таблица 1 – время выполнения поиска

	10	100	1 000	10 000	1 000 000 000
Алгоритм 1					
Алгоритм 2					
Алгоритм 3					
Мин. значение					
Макс. значение					
Ср. значение					