Задание 1

Гуляев Антон

Оцениваемая домашняя работа 1.

Задача 1

Алгоритм A: $T(n) = 3T(n-2) + O(1) \le 3T(n-2) + c \le$

$\le 9T(n-4) + 4c \le 9T(n-4) + 4c \le 27T(n-6) + 13c = \left[\sum = c\frac{(1-3^{\lfloor n/2 \rfloor})}{1-3}\right]$

$= 3^3 T(n-6) + c\cdot 3^0 + c\cdot 3^1 + c\cdot 3^2 + c\cdot 3^3$ => будет последним слагаемым в геометрической прогрессии — $c\cdot 3^{\lfloor n/2 \rfloor}$

I) n%2=0 $T(n) \le \underbrace{3^{n/2} T(0)}_{0} + \sum_{i=0}^{n/2-1} c\cdot 3^i = c\cdot \frac{1-3^{\frac{n}{2}-1}}{1-3} = c\cdot \frac{3^{\frac{n}{2}-1}-1}{2} =$

$= \frac{c\cdot 3^{\frac{n}{2}-1}}{2} - 1 = C_1\cdot 3^{\frac{n}{2}} - 1 = c_1\cdot 3^n = O(3^n)$, потому что при достаточно больших n

при $C_1 = 20c$, $\frac{20c}{6}\cdot 3^{\frac{n}{2}-1} - 1 \ge \frac{c\cdot 3^{n/2} - 1}{6}$

Ответ: $O(3^n)$

II) n%2≠0 Алгоритм A: $T(n) = 3T(n-2) + O(1) \le 3^{\lfloor n/2 \rfloor} T(1) + \sum_{i=0}^{\lfloor n/2 \rfloor - 1} c\cdot 3^i =$
n - нечёт

$= c\cdot 3^{\lfloor n/2 \rfloor} + c\cdot \frac{3^{\lfloor n/2 \rfloor - 1} - 1}{3-1} = \lim_{n\to\infty} c\cdot 3^{\lfloor n/2 \rfloor} + \underbrace{c\cdot \frac{3^{\lfloor n/2 \rfloor - 1} - 1}{3-1}}_{\text{отбросим, как незначительный член}} =$

$= \lim_{n\to\infty} c\cdot 3^{\lfloor n/2 \rfloor}$ — если взять $g(n) = 2c\cdot 3^n$, то мы получим,

значит, f(n) ограничена сверху $g(n) = 2c\cdot 3^n$, или $f(n) = O(3^n)$

Ответ: $O(3^n)$

Алгоритм B: $T(n) = 4T(\lceil n/2 \rceil) + O(n^2)$

Считаем, что алгоритм асимптотически неотрицателен, тогда можно взять $k \geq n$, где k - ближайшая степень двойки к n, больше n, в случае если n не степень двойки.

$\exists\, k = 2^{(\lfloor \log_2 n \rfloor)+1}$, если n - не степень 2 или n, если n, если n - степень 2.

$T(k) \geq T(n)$, т.к. считаем, что алгоритм асимптотически неотрицателен.

По основной теореме:

$a=4,\ b=2,\ d=2$

$d = \log_b a \Rightarrow T(k) = O(k^2 \log k)$, $k^2 \log k < 2n^2 \log(2n) \Rightarrow$

$\Rightarrow T(n) = O(n^2 \log n)$

Ответ: $O(n^2 \log n)$

Алгоритм C: $T(n) = 10T(\lceil n/3 \rceil) + O(n)$

Аналогично выберем $k = 3^{(\lfloor \log_3 n \rfloor + 1)}$, если n не степень 3, $k = n$, если n - степень 3.

Считаем, что алгоритм асимптотически неотрицателен.

По основной теореме

$a=10,\ b=3,\ d=1$

$d < \log_{10} 3 \Rightarrow T(k) = O(k^{\log_3 10})$ $\quad k^{\log_3 10} < (3n)^{\log_3 10} \Rightarrow$

$\Rightarrow T(n) = O(n^{\log_3 10})$

Ответ: $O(n^{\log_3 10})$

Самый быстрый алгоритм: ~~Em~~ $3^n > n^2 \log n$ и $3^n > n^{\log_3 10}$, т.к. показательная растёт быстрее степенной.

$\lim\limits_{n\to\infty} \frac{n^2 \log n}{n^{\log_3 10}} = \lim\limits_{n\to\infty} \frac{\log n}{n^{\log_3 10 - 2}} = 0$, т.к. $\frac{\log n}{n} = 0$, а $n^{\log_3 10 - 2} > n^1$.

Ответ: самый эффективный алгоритм: B, где
$T(n) = O(n^2 \log n)$

Задание 2

Задание 2

```
int majorRep (int array [1..n]) { // вход: массив полож чисел
 // выход: -1, если нет majorRep , иначе - majorRep
  если n > 1
    mid := n/2
   int mR1 := majorRep (array [1..mid])
   int mR2 := majorRep (array [mid+1, n])

   int countMR1 := 0    int countMR2 := 0
    если mR1 ≠ -1
       для i := 1 ; i <= mid инкремент i
           если mR1 = array[i]
               инкремент count mR1

       для i := mid+1 ; i <= n инкремент i
          если mR1 = array[i]
              инкремент count mR1

    если mR2 ≠ -1 и mR2 ≠ mR1
       для i := 1 ; i <= mid инкремент i
           если mR2 = array [i]
               инкремент count mR2

       для i := mid+1  i <= n инкремент i
          если mR2 = array [i]
              инкремент count mR2
     int half := (n/2)   ~~int halfE := (half % 2 = 0) ? half : half-1~~
    если count mR1 > half
         вернуть mR1
    если count mR2 > half
         вернуть mR2
   вернуть -1
 вернуть array[1]
```

Корректность алгоритма

При длине массива = 1, алгоритм корректен - вернёт свой единственный член

При длине 2, алгоритм получит оба этих члена, как претендентов на majorRep. Алгоритм подсчитает кол-во mr1 и mr2, далее, проверит: больше половины раз встречается какой-то mr, если да - вернёт его. В противном случае вернёт -1.

База индукции: Размер массива 1 - верно

Предположение: Пусть размер массива равен k, для него алгоритм работает корректно.

Переход k→2k. Так как метод использует метод разделяй и властвуй, то mr1 - это mr у длины массива k, что работает корректно, mr2 - mr у длины массива k - что тоже работает корректно. Далее у алгоритма есть 3 варианта:

1) оба mr = -1 - ни в одной половине массива нет представителя большинства, то и в результирующем массиве не будет: максимум k/2 чисел в чётном массиве!

33 22 33 1 0

две 3 две 3

3333 - 4, а размер массива - 8

2) Один mr ≠ -1, тогда алгоритм посчитает количества чисел в 2 массивах и сравнит с количеством половины массива.

3) Аналогичная ситуация будет, когда оба $mR \neq 1$: Тогда алгоритм посчитает количество mR_1 и mR_2 в 2 подмассивах, сравнит эти количества с половиной массива, очевидно, что на этом этапе, как и на прошлом может "победить" только представитель большинства.

Соответственно, алгоритм работает корректно, завершаемость алгоритма есть в базе индукции: при разделении массива его размер станет равным 1.

Оценим время работы алгоритма:

алгоритм вызывает самого себя для 2 частей массива, в 2 раза меньшей длины, подсчитывает в каждом подмассиве за $\Theta(n)$ операций кол-во mR_1 или mR_2.

$T(n) = 2T(\lceil n/2 \rceil) + \Theta(n)$

По основной теореме:

$a = 2,\ b = 2,\ d = 1,\ \log_2 2 = 1 \Rightarrow T(n) = \Theta(n^1 \log n)$

Ответ: $T(n) = \Theta(n \log n$

Код программы для задания 2

```
public class Main {


    public static int majorityRepresentative(int[] array,
                                              int leftIndex, int rightIndex) {     //right это array.length - 1
        if (rightIndex - leftIndex > 0) {
            int mid = (rightIndex + leftIndex) / 2;
            int majorRepFirst = majorityRepresentative(array, leftIndex, mid);
            int majorRepSecond = majorityRepresentative(array, mid + 1, rightIndex);


            int countMR1 = 0;   //константное число операций
            int countMR2 = 0;   //константное число операций


            if (majorRepFirst != -1) {     //c*n операций
                countMR1 = getCountMR(array, leftIndex, rightIndex, mid, majorRepFirst, countMR1);
            }
            if (majorRepSecond != -1 && majorRepFirst != majorRepSecond) {      //c*n операций
                countMR2 = getCountMR(array, leftIndex, rightIndex, mid, majorRepSecond, countMR2);
            }


            int half = (rightIndex - leftIndex + 1) / 2;            //количество элементов в массиве / 2


            if (countMR1 >= half + 1) {     //константное число операций
                return majorRepFirst;
            }
            if (countMR2 >= half + 1) {     //константное число операций
                return majorRepSecond;
            }
            return -1;
        }
        return array[leftIndex];
    }
```

```java
    private static int getCountMR(int[] array, int leftIndex, int rightIndex, int mid, int majorRep, int countMR) {
        for (int i = leftIndex; i <= mid; ++i) {
            if (majorRep == array[i]) {
                ++countMR;
            }
        }
        for (int i = mid + 1; i <= rightIndex; ++i) {
            if (majorRep == array[i]) {
                ++countMR;
            }
        }
        return countMR;
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int n = scanner.nextInt();

        int[] array = new int[n];

        for (int i = 0; i < n; ++i) {
            array[i] = scanner.nextInt();
        }

        int k = majorityRepresentative(array, 0, array.length - 1);

        System.out.println(k);
    }
}
```

Задание 3

Код программы для задания 3

```
public class Main {

    public static int select(int[] array, int k, int left, int right, int index) {

        Pair pair = divide(array, left, right, index);

        int postLess = pair.getPostLess();
        int postEqual = pair.getPostEqual();

        if (k <= postLess) {
            return select(array, k, left, postLess + left, -1);
        }

        if (k > postEqual) {
            return select(array, k - postEqual, postEqual + left, right, -1);
        }

        return array[postEqual - 1 + left];
    }

    public static Pair divide(int[] array, int left, int right, int index) {
        int randomIndex;
        if (index == -1) {
            randomIndex = (int) (Math.random() * (right - left - 1));    //смещение индекса
        } else {
            randomIndex = index;
        }

        int elemAtRandomIndex = array[randomIndex + left];   //смещение индекса
        int postLess = 0;      //количество меньших элементов
```

```
    for (int i = left; i < right; ++i) {
        if (array[i] < elemAtRandomIndex) {
            int temp = array[i];
            array[i] = array[postLess + left];
            array[postLess + left] = temp;
            ++postLess;
        }
    }
    int postEqual = postLess;      // количество меньших или равных

    for (int i = postEqual + left; i < right && postEqual + left < right; ++i) {
        if (array[i] == elemAtRandomIndex) {
            int temp = array[postEqual + left];
            array[postEqual + left] = array[i];
            array[i] = temp;
            ++postEqual;
        }
    }
    if (index != -1) {
        for (int j : array) {
            System.out.print(j + " ");
        }
        System.out.println("\n" + postLess + " " + (postEqual - 1));
    }
    return new Pair(postLess, postEqual);
}

public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    int n = scanner.nextInt();

    int[] array = new int[n];

    for (int i = 0; i < n; ++i) {
```

```
            array[i] = scanner.nextInt();
        }

        int index = scanner.nextInt();

        int indexOfMedian = array.length / 2;
        if (array.length % 2 != 0) {
            ++indexOfMedian;
        }

        System.out.print("median: " + select(array, indexOfMedian, 0, array.length, index));
    }
}

class Pair {

    private final int postLess;
    private final int postEqual;

    public Pair(int postLess, int postEqual) {
        this.postLess = postLess;
        this.postEqual = postEqual;
    }

    public int getPostLess() {
        return postLess;
    }

    public int getPostEqual() {
        return postEqual;
    }
}
```

Задание 3.

Корректность алгоритма divide [array[1...k]

Алгоритм возвращает индекс элемента, в котором лежит элемент, вокруг которого надо „разделить" массив.

Алгоритм проходит 2 раза по массиву, сначала ставя на место меньшие, затем - равные.

Докажем по индукции, что алгоритм корректен.

База: при $k = 1$, алгоритм ничего не сделает с массивом, массив и так будет в правильном „положении"

Индукционный переход:] при $k = n$, алгоритм корректен

Докажем, что алгоритм корректен при $k = n+1$:

] новый элемент - temp, разделяющий элемент - p.

а) При $temp < p$, за первый проход по массиву, алгоритм передвинет temp в левую часть, увеличит её размер на единицу. (a_L)

б) При $temp = p$, за второй проход по массиву, алгоритм поставит temp в ряд элементов, которые равны p (a_p), увеличит их количество на единицу

в) При $temp > p$, алгоритм уберет его с a_L и a_p в a_R, место в массиве, где элементы больше p.

Соответственно, temp окажется на правильном месте, что оставит корректным разделяющий массив.

Докажем линейность времени работы алгоритма!

Алгоритм выполняет:

7 константных операций

3 или 2 операции со вложностью $O(n)$

Значит $T(n) = O(n) + O(7) = O(n)$

Докажем по определению:

семейство функций порядка n

$\exists\, c_1 > 0 : \forall n > n_0 \quad f(n) \le c_1 \cdot g(n)$

кол-во операций в 1 итерации

$3 \cdot c \cdot n + 7 \le c_1 \cdot g(n)$

Очевидно, что при достаточно больших n, можно отбросить 7.

$3 \cdot c \cdot n \le c_1 g(n)$

$n \le \frac{c_1}{3c} g(n)$

$1 \le \frac{c_1}{3c}$, то неравенство выполняется.

$\exists\, c_1 = 5c$, тогда $1 \le \frac{5}{3}$ ▨