2. Определить сложность следующих алгоритмов:

-. Поиск элемента массива с известным индексом O(1)

-. Дублирование одномерного массива через foreach O(n)

-. Удаление элемента массива с известным индексом без сдвига O(1)

-. Удаление элемента массива с неизвестным индексом без сдвига O(n)

-. Удаление элемента массива с неизвестным индексом со сдвига O(n2)-если будем искать элемент для удаления и сразу сдвигать или О(n)-если мы найдем и выйдем из массива, запомним элемент, а вторым циклом сдвинем

.

3. Определить сложность следующих алгоритмов. Сколько произойдет итераций?

a) O(n)\*O(log n)=O(n\*logn)

int n = 10000;

List<Integer> arrayList = new ArrayList<>();

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

for (int j = 1; j < n; j \*= 2) { //O(log n)

arrayList.add(i \* j);

}

}

b) O(n)\*O(n/2)=O(n2/2)=O(n2)

int n = 10000;

List<Integer> arrayList = new ArrayList<>();

for (int i = 0; i < n; i += 2) { //O(n)

for (int j = i; j < n; j++) { //O(n/2)

arrayList.add(i \* j);

}

}

с) O(n/2)\*O(n/2)=O(n2/4)=O(n2)

int n = 10000;

List<Integer> arrayList = new ArrayList<>();

for (int i = 0; i < n; i ++) { //O(n/2)

for (int j = 0; j < n; j++) {// O(n+(n-1)+(n-2)+…+1)=O(n/2)

arrayList.add(i \* j);

n--;

}

}

d) O(n)

```

factorial(BigInteger.valueOf(10000))

public static BigInteger factorial(BigInteger n) {

if (n.equals(BigInteger.ONE)) {

return n;

}

return n.multiply(factorial(n.subtract(BigInteger.valueOf(1))));

}

e) O(2n)=O(n)

fib(BigInteger.valueOf(50));

public static BigInteger fib(BigInteger n) {

if (n.equals(BigInteger.ONE)) {

return BigInteger.ZERO;

}

if (n.equals(BigInteger.TWO)) {

return BigInteger.ONE;

}

return fib(n.subtract(BigInteger.ONE)).add(fib(n.subtract(BigInteger.TWO)));

}