Метод test1(int n)

```
void test1(int n) {
 if (n==1) return;

for (int i=1; i<=n; i++)
   for (int j=1; j<=n; j++)
   System.out.println("*"); break;
 }
```

if (n == 1) return; — этот блок выполняется за время O(1), так как это проверка условия.

Циклы for (int i = 1; i <= n; i++) и for (int j = 1; j <= n; j++):

- Внешний цикл i выполняется n раз.
- Внутренний цикл j должен был бы выполняться n раз для каждого значения i, но тут есть break, который завершает внутренний цикл после первой итерации.

Таким образом, несмотря на наличие двух вложенных циклов, внутренний цикл выполняется только один раз на каждую итерацию внешнего цикла.

Поэтому операция System.out.println("*"); выполняется всего n раз.

Итоговая временная сложность для test1(int n) — O(n).

Метод test2(int n)

```
void test2(int n) {

 int a = 0;
   for (i = 0; i < n; i++)
     for (j = n; j > i; j--)
       a = a + i + j;
}
```

Внешний цикл for (i = 0; i < n; i++): выполняется n раз.

Внутренний цикл for (j = n; j > i; j--):

- В первый раз (когда i=0) цикл j выполнится n раз.
- Когда i=1, цикл выполнится n−1 раз.
- И так далее, до последнего значения i = n − 1, где цикл выполнится 1 раз.

Суммарное количество итераций можно выразить как сумму арифметической прогрессии:

Общее количество итераций=n+(n−1)+(n−2)+⋯+1=n(n+1)/2

Итоговая временная сложность для test2(int n) — O(n^2).

Method test3 (int n)

```
void test3(int n) {

 int i, j, a = 0;

   for (i = n/2; i <=n; i++)    // (1)

     for (j = 2; j <=n; j=j*2)    // (2)

        a=a+n/2;

}
```

Внешний цикл (1) начинается с i = n/2 и идет до i = n. Таким образом, этот цикл выполняется примерно n/2 раз, что равно O(n).

Внутренний цикл (2) начинается с j = 2 и умножает j на 2 при каждой итерации, пока j <= n. Количество итераций внутреннего цикла можно определить как $\log_2(n)$.

Таким образом, общая временная сложность функции test3:

- Внешний цикл: O(n)
- Внутренний цикл: O(log n)

Итоговая сложность: O(n * log n).

Method test4 (int n)

```
void test4(int n) {
 int a = 0, i = n;
   while (i > 0) {          // (1)
    a += i;
    i /= 2;              // (2)
  }
}
```

В данном случае, while цикл (1) выполняется до тех пор, пока i > 0. При каждой итерации i делится на 2 (2).

Каждый раз i делится на 2, то есть количество итераций можно определить как $\log_2(n)$.

Итоговая временная сложность функции test4: O(log n).

Резюме:

- Временная сложность test3: **O(n * log n)**
- Временная сложность test4: **O(log n)**