Big-O Worksheet

1. Approximate the runtime of the following code fragment, in terms of *n*:

```
int sum = 0;
for (int j = 1; j < n; j++) {
   sum++;
   if (j % 2 == 0) {
       sum++;
   }
}
```

O(n)

2. Approximate the runtime of the following code fragment, in terms of *n*.

```
int sum = 0;
for (int i = 1; i <= n * 2; i++) {
   for (int j = 1; j <= n; j++) {
       sum++;
   }
}
for (int j = 1; j < 100; j++) {
   sum++;
   sum++;
}
```

O(n^2)

3. Approximate the runtime of the following code fragment, in terms of *n*.

```
int sum = 0;
for (int i = 1; i <= n; i++) {

   for (int j = 1; j <= i; j += 2) {
       sum += 4;
   }
}
for (int k = -50; k <= -1; k++) {
   sum--;
}
```

O((n^2))

4. Approximate the runtime of the following code fragment, in terms of *n*.

```
int sum = 0;

for (int i = 1; i <= n; i++) {
  for (int j = 1; j <= 1000000; j++) {
     sum += 10;
  }
}
sum += 5555;
```

O(n^2)

5. Approximate the runtime of the following code fragment, in terms of *n*:

```
int sum = 0;
int j = 1;
while (j <= n) {
  sum++;
  j = j * 2;
}
```

O(n)

6. Approximate the runtime of the following code fragment, in terms of *n*:

```
int sum = 0;
for (int j = 1; j < n; j++) {
  sum++;
  if (j % 2 == 0) {
     sum++;
  }
}
```

O(n)

7. What is the time complexity of the ArrayList remove(index) method?

O(1)

8. Approximate the runtime of the following code fragment, in terms of *n*:

```
int i, j, k = 0;
for (i = n / 2; i <= n; i++) {
    for (j = 2; j <= n; j = j * 2) {
        k = k + n / 2;
     }
}
```

O(n^2)

9. Approximate the runtime of the following code fragment, in terms of *n*:

```
int a = 0, i = N;
while (i > 0) {
    a += i;
    i /= 2;
}
```

O(n)

10. Approximate the runtime of the following code fragment, in terms of *n*

```
public static long mystery1(int n) {
        if (n == 1)
                return 1;
        return n * mystery1(n-1);
}
```

O(n)

11. Determine the time complexity of the following code fragments as a function of n.
For each code fragment on the left, find the letter of the best-matching term from the right.
You may use each letter once, more than once, or not at all.

1

```
int count = 0;
```

2

```
int count = 0;
for (int i = 1; i <= n; i++) {
    for (int j = 1; j <= n; j++) {
        count++;
    }
}
```

3

```
int count = 0;
for (int i = 1; i <= n; i++) {
    for (int j = 1; j <= n; j = 2*j) {
        count++;
    }
}
```

4

```
public static int f(int n) {
    if (n == 0) return 1;
    return f(n-1) + f(n-1);
}
```

5

```
int count = 0;
for (int i = 1; i <= n; i++)
    count++;
for (int j = 1; j <= 2*n; j++)
    count++;
for (int k = 1; k <= 3*n; k++)
    count++:
```

A. $\Theta(1)$ *constant*

B. $\Theta(\log n)$ *logarithmic*

C. $\Theta(n)$ *linear*

D. $\Theta(n \log n)$ *linearithmic*

E. $\Theta(n^2)$ *quadratic*

F. $\Theta(n^3)$ *cubic*

G. $\Theta(2^n)$ *exponential*