Answer: Running time is O(n)
Explanation: There is a single loop that runs n−5 times. Each time the loop runs it executes instruction in the loop header and 1 instruction in the body of the loop. The total number of instructions is $2 * (n - 5) + 1$ (for the last loop check) $= 2n - 9 = O(n)$. (also OK: $\Theta(n)$).

1)
```
num=0;
for (i = n; i>= 0; i−−)
        num−−;
```

Answer: Running time is O(n)
Explanation: after the 1st instruction, there is a single loop that runs (n+1) times; each time the loop runs it executes the instruction in the loop header and 1 instruction in the body of the loop. The total number of instructions is 2*(n+1) +1 (for the last loop check) + 1 = 2n + 4 = O(n).

2)
```
num=0;
for (i = 0; i<= n * n; i=i+2)
        num=num+2;
```

Answer: Running time is O(n^2).
Explanation: after the 1st instruction, there is a single loop that runs (n^2)/2 + 1 (including last loop check);
therefore 2[(n^2)/2] +1 + 1 = n^2 + 2 = O(n^2).

0 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36

n=2	4
n=3	6
n=4	10
n=5	14
n=6	20

3)
```
num=0;
for (i = 1; i<= n; i=i*2)
        num++;
```

Answer: O(logN)
Explanation: Since the number of iterations decreases by half, loop has logN + 1 complexity (inclusive of last loop check); therefore 2(logN) + 1 +1 = 2logN + 2 = O(logN).

4)
```
num=1;
for (i = 0; i<n; i++)
        for (j = 0; j<=i; j++)
                num = num * 2
```

Answer: O(n^2)

Explanation: total loop iteration pattern follows the triangular number sequence which is n(n+1)/2. Thus total time complexity is n(n+1)/2 + n (number of inner loop last checks) + 1 (last outer loop check) = O(n^2).

Eg let n = 4

I value	0	1	2	3	4
Outer iterations	1st	2nd	3rd	4th	5th
Inner iterations	1	2	3	4	5

Total 10 inner instructions for n = 4

*5)
p=10;
num=0;
plimit=100,000;

thus O(n^2) = (n + 1) + (n^2 – np + 1)

```
for (i = p; i<=plimit; i++)                  ((10^5) – p  + 1)
        for (j = 1; j<=i; j++)               ((10^5) – 9) n
                num = num + 1;
```

Answer: O(n^2)
Explanation: let plimit = n; I first find a pattern for the number of inner iterations generated per outer iteration and I found that the inner iterations has the following pattern: 10,21,33,46 . .I then find the primary and secondary difference and used a general quadratic formula an^2 +bn+c; I then sub 3 values n= 10,11,12 to the equations to get 3 equations to solve for the variables a, b and c. The result is ½*n^2 + ½*n -45, which is the algorithm for finding the number of iterations for a n number of outer iterations.

Hence ½ * n^2 + ½ * n -45 + n – p instructions occur, O(n^2)

6)
num=0;

```
for (i = n*n; i>=0; i=i/2)                   log(n^2)
        for (j = 1; j<=n; j++)               n
                num = num + i;
```

Answer: O(nlog(n^2))
Explanation: Since outer loop iterations log(n^2) due to the initial I being n^2 but every iteration decreases by i/2, it shows a log function; inner loop then iterates n times for each iteration of the outer loop.
Thus the total iteration is n * log(n^2) + n (last checks of inner loop) + 1 (last check of outer loop) + log(n^2) (total outer loop iterations = O(n * log(n^2)).

7)

```
num=0;
for (i = 0; i<n*n-1; i++)                              n^2 - 1 times
        for (j = 0; j< i * i; j++)                              I^2 times
                num = 2*num;
```

Answer: O(n^3)
Explanation: as complexity of loop is [n*(n+1)*(2n+1)]/6. We know this because for every iteration, the inner loop has to iterate squared the index of outer iteration which is similar to the sum of squares number sequence.

8)

```
num=0;
for (i = 0; i<= n*n; i++)
        num++;
for (i = 1; i<=n; i=i*2)
        for (j=n; j>= 1; j=j/2)
                num++;
```

Ans: O(n^2)
Explanation: 1st loop is n^2 +1 + 1(last check); 2nd loop is log(n)*log(n) + log(n) because outer of 2nd loop increases in a multiple of 2 while inner of 2nd loop decreases by a division of 2.
Since n^2 is upper bound of (log(n) * log(n)), complexity of this algorithm is O(n^2).

9*)

```
for (i = 0; i < n; i++) {                   n times
        smallest = i;
        for (j = i+1; j <= n; j++) {
                if (a[j] < a[smallest])
                        smallest = j;
        }
        swap(a, i, smallest); // has three instructions
}
```

Ans: O(n^2)
Explanation:
for loop follows the triangular number sequence thus has complexity of n(n+1)/2 * 2 (assume have to swap elements for every iteration); swap has 3n instructions in total.
Thus total complexity is O(n^2).

10)

```
num = 0;
i = 0;

while (i<n) {
        j = 0;
        while (j<100) {
                //constant time operations
```

```
        j++;
    }
    i++;
}
```

Ans: O(n)
Explanation: since while loop is (i<n) and I = 0, outer loop iterates n times;
Inner while loop iterates 100 times for every outer loop iteration, thus complexity of inner loop is 100*n.
Total complexity = 100 * n + n (inner + last inner loop check) + n + 1 (outer + outer last loop check) = O(n).