**Theoretical Analysis**

**1.**

long f1(long n){  
    long k = 0;  
    for (long i = 0; i < n; i++){  
           for (long j = 0; j < n; j++){  
                  k++;  
           }  
    }  
    return k;  
}

First loop iterates n times and each time inner loop also iterate n times so time complexity will be n2

Worst-case running: O(n2)

**2.**

void f2(long n){  
       long k = 0;  
       for (long i = 0; i < n; i++){  
              for (long j = 0; j < i; j++){  
                      k++;  
              }  
              for (long t = 0; t < 10000; t++){  
                      k++;  
              }  
       }  
}

First loop iterates n times, first inner loop will also do n times in the worst-case, second inner loop do 10000 (which is constant) so time complexity will be n2

Worst-case running: O(n2)

3.

void f3(long n){  
    if (n > 0){  
           f3(n / 2);  
           f3(n / 2);  
    }  
}

Each node will do 2 call so 2n with n is the depth of the tree, and each time n will be divided by 2 so the depth of the tree is log2(n). In total, time complexity is 2log2n = n

Worst-case running: O(n)

4.

void f4(long n){  
    if (n > 0){  
           f4(n / 2);  
           f4(n / 2);  
           f2(n);  
    }  
}

Worst-case running: O(n3)

5.

void f5(long n){  
       long k = 0;  
       for (long i = 0; i < 10; i++){  
              long j = n;  
              while (j > 0){  
                      f1(1000);  
                      k++;  
                      j = j / 2;  
              }  
       }  
}

Worst-case running: O(log2n)

6.

void f6(long n){  
    f2(n);  
    f3(n);  
    f5(n);  
}

Worst-case running: O(n2) + O(n) + O(n2) = O(n2)

7.

void f7(long n){  
    long k = 0;  
    for (long i = 0; i < f1(n); i++){  
           k++;  
    }  
    for (long j = 0; j < n; j++){  
           k++;  
    }  
}

Worst-case running: O(n4)