Time Complexity:

Big-Oh Notation:
g(n) = O(f(n)) if g(n) <= c.f(n)

Order:
O(1) < O(log n) < O(sqrt(n)) < O(n) < O(nlogn) < O(n^2) < O(n^3) < … < O(2^n) < O(n^n)

1.
```
int n, k=0;
cin>>n;
for(int i=1; i<=n; i++){      //O(n)
      k+=i; //O(1) - > Constant time operation
}
```

2.
```
int n, k=0;
cin>>n;
for(int i=1; i<=n; i++){      //O(n)
      for(int j=0; j<i; j++){ //O(i)
            k++;
      }
}     //1 + 2 + 3 + .. + n = O(n^2)
```

3.
```
for(int i=1; i*i<=n; i++){    //O(sqrt(n))
      k+=i; // O(1)
}     //Overall O(sqrt(n))
```

4.
```
for(int i=1; i<=n; i++){
      for(int j=0; j<i; j++){
            k++;
      }
}
for(int i=0; i<n; i++){
      q+=i;
}
```

Overall -> O(n^2) -> O(n^2 + n) but n is ignored w.r.t n^2

5.
```
for(int i=1; i<=n; i++){
      for(int j=0; j<i; j++){
            k++;
      }
}
for(int i=0; i<m; i++){
      q+=i;
}
```
Overall -> O(n^2 + m) -> O(max(n^2, m)) is technically correct but not conventional to use

6.
```
for(int i=n; i>1; i/=2){
      for(int j=0; j<i; j++){
            res++;
      }
}
```

Overall -> n + (n/2) + (n/4) + … + 1 = 2n = O(n)
O(nlogn) is technically correct but it is possible to have a tighter upperbound of O(n)

7.
```
for(int i=1; i<=sqrt(n); i++){
      res *= i;
}
```

Overall -> O(root(n) * log(n)) because sqrt() is called root(n) times and time complexity of sqrt() is O(log(n))