```
for (i = 0; i < n; i++) {
        for (j = 0; j < m; j++) {
                sequence of statements
} }
```

$T(n) = nm$

```
for (i = 0; i < n; i++) {
        for (j = i+1; j < n; j++) {
                sequence of statements
} }
```

$T(n) = n*(n+1)/2 = O(n^2)$

```
for (i = 0; i < n; i++) {
        for (j = i+1; j < n; j++) {
                Algo ( ); ➔ O (n)
} }
```

$T(n) = n*(n+1)/2 * cn = O(n^3)$

```
Algo (A)
    d = ∞
    for i = 1 to n − 1
        for j = i + 1 to n
                if |X[i] − X[j]| < d
                        d = |X[i] − X[j]|
    return d
```

$T(n) = n*(n+1)/2 = O(n^2)$

X is an array of coordinates of points on the x-axis

(a) What does Algo do? give a tight asymptotic bound for the complexity of Algo?

(b) Write an algorithm that is functionally equivalent to Algo (A), but with a better asymptotic complexity.

```
Algo (A)
    for i = 3 to n
        for j = 2 to i − 1
            for k = 1 to j − 1
                if |A[i] − A[j]| = = |A[j] − A[k]| or |A[i] −
                                                A[k]| = = |A[j] − A[k]|
                                                return true
    return false
```

$T(n) = \sum_{i=3}^{n} \sum_{j=2}^{i-1} \sum_{k=1}^{j-1} O(1) = O(n^3)$

Write an algorithm equivalent to Algo (A) (for all A) but with a strictly better asymptotic complexity than Algo (A).

```
Algo-X (A, k)
    i = 1
    while i ≤ n
        if A[i] = = k
                Algo-Y(A, i)
        else i = i + 1

Algo-Y (A, i)
    while i < n
        A[i] = A[i + 1]
        i = i + 1
    A[i] = null
```

$T(n) = O(n^2)$

$T(n) = O(n)$

Analyze the complexity of Algo-X and write an algorithm called Better-Algo-X that does the same thing but with a better asymptotic complexity

```
Algo(A)
    i = 1
    j = 1
    m = 0
    c = 0
    while i ≤ n
        if A[i] == A[j]
                c = c + 1
        j = j + 1
        if j > n
                if c > m
                        m = c
                c = 0
                i = i + 1
                j = i
    return m
```

Do it yourself.

(a) Analyze the complexity of Algo.

(b) Write an algorithm that does exactly the same thing as Algo but with a better asymptotic time complexity

```
Factorial (n) {
        if n=0 return 1
        else return Factorial (n-1) * n
}
```

T(n) = T(n-1) + c

Check Palindromes like "racecar".

```
bool isPalindrome (string s) {
        if (s.length() <= 1) return true;
        return (s[0]==s[s.length()-1]) &&
                isPalindrome(s.substr(1,s.length()-2));
}
```

T(n) = T(n-2) + c

```
int Fibonacci (int N) {
        if (N==1) return 1;
        if (N==2) return 1;
        return Fibonacci(N-1) + Fibonacci(N-2);
}
```

$T(n) = T(n-1) + T(n-2) + c$

```
int search (vector<int> v, int from, int to, int val) {
        if (from>to) return -1; //val not found
        int mid = (from+to)/2;
        if (v[mid] == val)
                return mid;
        else if (val > v[mid])
                return search(v,mid+1,to,val);
        else
                return search(v,from,mid-1,val);
}
```

$T(n) = T(n/2) + c$