Expected Time Complexity

10^12	O(√n)
10^8	O(n)
10^6	O(nlogn)
10^5	O(n√n)
10^4	O(n^2)
10^3	O(n^2√n)
n<=25	O(2^n)
n<=10	O(n!)

Sliding Window (Shrinkable)

```
// Time: O(N)
// Space: O(1)
class Solution {
public:
       int longes tSu bar ray (ve cto r<i nt> & A)
{
               int i = 0, j = 0, N = A.size(), cnt
= 0, ans = 0;
               for (; j < N; ++j) {
                       cnt += A[j] == 0;
                        while (cnt > 1) cnt -=
A[i++] == 0;
                       ans = max(ans, j - i); //
note that the window is of size j - i + 1. We use
j - i here because we need to delete a number.
               }
                return ans;
       }
};
```

find All Substrings

```
vector<string> findAllSubstrings(const string& s) {
        vec tor <st rin g> substr ings;
       int length = s.leng th();

       // Generate all possible substrings
       for (int start = 0; start < length;
++start) {
               for (int end = start + 1; end <=
length; ++end) {
                        sub str ing s.p ush _ba -
ck( s.s ubs tr( start, end - start));
```

find All Substrings (cont)

```
>      }
  }

  return substrings;
}
```

BFS

```
void bsf(int start,vector<int>
adj[],vector<bool>&vis,vector<int>&result){
                que ue< int >que;
                que.pu sh( start);
                vis [st art ]=true;
                res ult.pu sh_ bac k(s tart);
                whi le( !qu e.e mpt y()){
                       int u=que.f ro nt();
                        que.pop();
                        for (auto x:adj[u]){
                                if( !vi s[x]){
                                        que.pu -
sh(x);
                                        vis [x] -
=true;
                                        res -
ult.pu sh_ bac k(x);
                                }
                        }
                }
        }
```

DFS

```
void dfs( unordered_map<int,vector<int>> adj,int
start,vector<bool>&vis,vector<int>&result){
                if( vis [st art ]== true) return;
                vis [st art ]=true;
                res ult.pu sh_ bac k(s tart);
                for (auto x:adj[ sta rt]){
                        if( !vi s[x]){
                                dfs (ad j,x ,vi -
s,r esult);
                        }

                }
```

DFS (cont)

> }

Time Complexity & Algorithms

O(logn)	Binary Search, Balanced Binary Search Trees (AVL Tree, Red-Black Tree), Heap Operations
O(n)	Breadth-First Search (BFS), Depth-First Search (DFS), Single-pass Hash Table Operations, Prefix Sum Array Calculation
O(nlogn)	Sorting algorithms (general), heap
O(n^2)	dp, Brute-force

Sliding Window (Non-Shrinkable)

```
// Time: O(N)
// Space: O(1)
class Solution {
public:
       int longes tSu bar ray (ve cto r<i nt> & A)
{
               int i = 0, j = 0, N = A.size(), cnt
= 0;
               for (; j < N; ++j) {
                       cnt += A[j] == 0;
                       if (cnt > 1) cnt -= A[i++]
== 0;
               }
                return j - i - 1;
       }
};
```

Binary Search

```
int binarySearch(vector<int>& nums, int target){
    if( num s.s ize() == 0)
        return -1;
   int left = 0, right = nums.s ize() - 1;
    whi le(left <= right){
       // Prevent (left + right) overflow
       int mid = left + (right - left) / 2;
        if( num s[mid] == target){ return mid; }
```

Binary Search (cont)

```
>     else if(nums[mid] < target) { left = mid + 1; }
   else { right = mid - 1; }
 }
 // End Condition: left > right
 return -1;
}
```

Dynamic Programming Patterns

- Minimum (Maximum) Path to Reach a Target

Approach:
Choose the minimum (or maximum) path among all possible paths before the current state, then add the value for the current state.
Formula:routes[i] = min(routes[i-1], routes[i-2], ..., routes[i-k]) + cost[i]

- Distinct Ways

Approach:
Choose minimum (maximum) path among all possible paths before the current state, then add value for the current state.
Formula: routes[i] = routes[i-1] + routes[i-2], ... , + routes[i-k]

- Merging Intervals

Approach:
Find all optimal solutions for every interval and return the best possible answer
Formula: dp[i][j] = dp[i][k] + result[k] + dp[k+1][j]

- DP on Strings

Approach:
Compare 2 chars of String or 2 Strings. Do whatever you do. Return.
Formula: if s1[i-1] == s2[j-1] then dp[i][j] = //code. Else dp[i][j] = //code

- Decision Making

Approach:
If you decide to choose the current value use the previous result where the value was ignored; vice-versa, if you decide to ignore the current value use previous result where value was used.
Formula: dp[i][j] = max({dp[i][j], dp[i-1][j] + arr[i], dp[i-1][j-1]}); dp[i][j-1] = max({dp[i][j-1], dp[i-1][j-1] + arr[i], arr[i]});