Substring, Subsequence and Subarray — Detailed Notes with Code

This PDF explains the differences between **substring**, **subsequence**, and **subarray** with clear examples, all possible lists for a small sample, formulas, and example implementations in C++, Java and Python.

1. Definitions

Substring: A substring is a continuous part of a string. Characters must be contiguous and maintain order.
Subsequence: A subsequence is formed by deleting zero or more characters from the original string without changing the relative order. Continuity is NOT required.
Subarray: A subarray is a continuous slice of an array. It is analogous to a substring but for arrays (numbers or any elements).

2. Examples (Complete lists) — String: "abc" and Array: [1, 2, 3]

Example string: **"abc"**
- All substrings (continuous): "a", "b", "c", "ab", "bc", "abc" (total = 6, formula n(n+1)/2)
- All subsequences (any selection preserving order): "", "a", "b", "c", "ab", "ac", "bc", "abc" (total = 2^n = 8, includes empty string)

Example array: **[1, 2, 3]**
- All subarrays (continuous): [1], [2], [3], [1,2], [2,3], [1,2,3] (total = 6)

3. Quick Difference Summary

- **Continuity:** Substring & Subarray require continuity; Subsequence does NOT.
- **Structure:** Substring → string only; Subarray → array only; Subsequence → usually discussed for strings (but applies to arrays too).
- **Count:** Substrings/Subarrays = n(n+1)/2; Subsequences = 2^n (including empty).
- **Use cases:** Substring & Subarray often used when a contiguous chunk is required (sliding window problems). Subsequences used in problems like LCS (longest common subsequence), subsequence counting, etc.

4. Code Examples — generate substrings, subsequences and subarrays

A) C++

```
// C++: generate all substrings, subsequences, and subarrays for small inputs
#include <bits/stdc++.h>
using namespace std;

// Substrings (continuous) of a string s
vector<string> allSubstrings(const string &s) {
    int n = s.size();
    vector<string> res;
    for(int i=0;i<n;i++){
        for(int len=1;i+len<=n;len++){
            res.push_back(s.substr(i,len));
        }
    }
    return res;
}

// Subsequences (all) of a string s (including empty)
vector<string> allSubsequences(const string &s) {
    int n = s.size();
    vector<string> res;
```

```
    for(int mask=0; mask < (1<<n); ++mask){
        string cur;
        for(int i=0;i<n;i++){
            if(mask & (1<<i)) cur.push_back(s[i]);
        }
        res.push_back(cur);
    }
    return res;
}

// Subarrays (continuous) of a vector<int> a
vector<vector<int>> allSubarrays(const vector<int> &a) {
    int n = a.size();
    vector<vector<int>> res;
    for(int i=0;i<n;i++){
        for(int len=1;i+len<=n;len++){
            vector<int> seg(a.begin()+i, a.begin()+i+len);
            res.push_back(seg);
        }
    }
    return res;
}

int main(){
    string s = "abc";
    auto subs = allSubstrings(s);
    cout << "Substrings:\\n";
    for(auto &x: subs) cout << x << "\\n";

    auto subsq = allSubsequences(s);
    cout << "\\nSubsequences (including empty):\\n";
    for(auto &x: subsq) cout << '\"' << x << '\"' << "\\n";

    vector<int> a = {1,2,3};
    auto sarr = allSubarrays(a);
    cout << "\\nSubarrays:\\n";
    for(auto &seg: sarr){
        cout << '[';
        for(size_t i=0;i<seg.size();++i){
            if(i) cout << ',';
            cout << seg[i];
        }
        cout << "]\\n";
    }
    return 0;
}
```

B) Java

```
// Java: generate all substrings, subsequences, and subarrays for small inputs
import java.util.*;
public class AllCases {
    // Substrings
    static List<String> allSubstrings(String s){
        int n = s.length();
        List<String> res = new ArrayList<>();
        for(int i=0;i<n;i++){
            for(int len=1;i+len<=n;len++){
                res.add(s.substring(i,i+len));
            }
        }
        return res;
    }
```

```
    // Subsequences (including empty)
    static List<String> allSubsequences(String s){
        int n = s.length();
        List<String> res = new ArrayList<>();
        int total = 1<<n;
        for(int mask=0; mask<total; mask++){
            StringBuilder cur = new StringBuilder();
            for(int i=0;i<n;i++){
                if((mask & (1<<i)) != 0) cur.append(s.charAt(i));
            }
            res.add(cur.toString());
        }
        return res;
    }

    // Subarrays
    static List<List<Integer>> allSubarrays(int[] a){
        int n = a.length;
        List<List<Integer>> res = new ArrayList<>();
        for(int i=0;i<n;i++){
            for(int len=1;i+len<=n;len++){
                List<Integer> seg = new ArrayList<>();
                for(int j=i;j<i+len;j++) seg.add(a[j]);
                res.add(seg);
            }
        }
        return res;
    }

    public static void main(String[] args){
        String s = "abc";
        System.out.println("Substrings:");
        for(String x: allSubstrings(s)) System.out.println(x);

        System.out.println("\\nSubsequences (including empty):");
        for(String x: allSubsequences(s)) System.out.println('\"'+x+'\"');

        int[] a = {1,2,3};
        System.out.println("\\nSubarrays:");
        for(List<Integer> seg: allSubarrays(a)) System.out.println(seg);
    }
}
```

C) Python

```
# Python: generate all substrings, subsequences, and subarrays for small inputs
from itertools import combinations

# Substrings (continuous)
def all_substrings(s):
    n = len(s)
    res = []
    for i in range(n):
        for l in range(1, n-i+1):
            res.append(s[i:i+l])
    return res

# Subsequences (including empty)
def all_subsequences(s):
    n = len(s)
    res = []
    for r in range(n+1):
        for comb in combinations(range(n), r):
```

```
            res.append(''.join(s[i] for i in comb))
    return res

# Subarrays (continuous)
def all_subarrays(a):
    n = len(a)
    res = []
    for i in range(n):
        for l in range(1, n-i+1):
            res.append(a[i:i+l])
    return res

if __name__ == "__main__":
    s = "abc"
    print("Substrings:")
    print(all_substrings(s))
    print("\\nSubsequences (including empty):")
    print(all_subsequences(s))
    a = [1,2,3]
    print("\\nSubarrays:")
    print(all_subarrays(a))
```

5. Notes & Complexity

- Generating all subsequences is O(2^n * n) time and O(2^n) space (for storing results). Practical only for small n (n <~ 20).
- Generating all substrings/subarrays is O(n^2) in count; constructing each may cost up to O(n) if you copy, so naive cost can be O(n^3) overall if not careful. But listing them as (start,end) pairs costs O(n^2) space/time to enumerate pairs.
- Use these exhaustive generators only for small inputs or demonstrations. For algorithmic problems look for optimized approaches (sliding window, DP, two pointers).