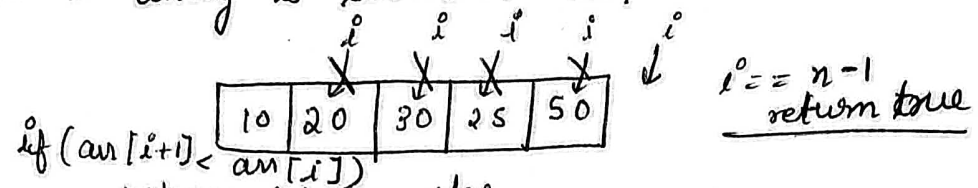
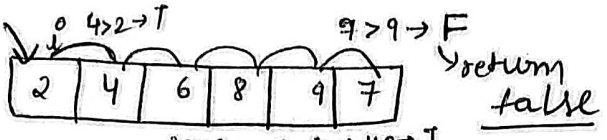
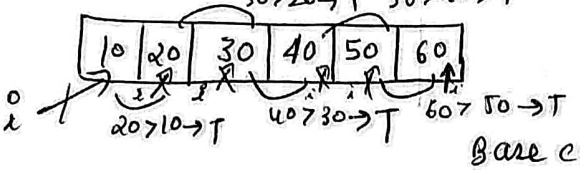
Recursion - 3

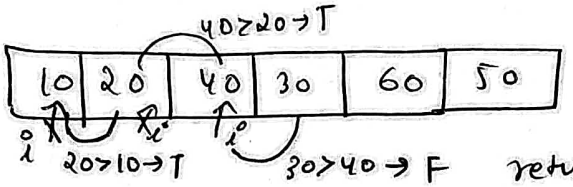
Ques i/p → An array.
o/p → array is sorted or not? True or False?



Test Cases →

① 

② 

③ 

algo →
if (arr[i+1] > arr[i]
aage badho
Else
return false.

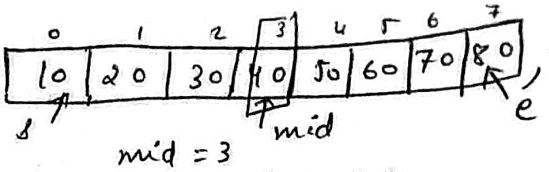
code -

```
vector<int> v{10,20,30,50,60};
int n = v.size();
bool isSorted = checkSorted(v,n,i);
if (isSorted){
    cout<<"Array is sorted";
}
else{
    cout<<"Array is not sorted";
}
bool checkSorted(vector<int>&arr, int &n , int i){   // pass by reference → to save memory and time.
    if(i==n-1)
        return true;
    // 1 case solve krna hai → sirf phle do element check karlo.
    if(arr[i+1]<arr[i])
        return false;
    // Baki recursion sambhal lega.
    return checkSorted(arr,n,i+1);
}
```

Ques → Binary Search

Iterative →

```
int bs ( arr , n){
    s = 0 , e = n-1;
    mid = s + (e-s)/2.
    while (s <= e){
        if ( arr[mid] == key){
            return mid;
        }
        else if (arr[mid] < key)
            s = mid + 1;
        else
            e = mid - 1;
    }
    mid = s + (e-1)/2;
}
```

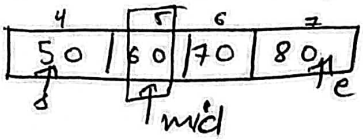


mid = 3

40 >= 70 → F

40 < 70 → T → search in right

s = mid + 1 = 4

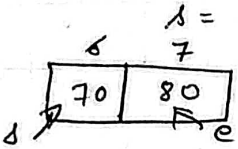


mid = 5

60 >= 70 → F

60 < 70 → T

s = mid + 1 = 6



mid = 6

70 >= 70 → T

return mid = 6

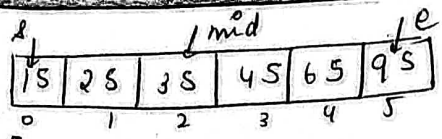
Now Binary search using Recursion →

```
int binarySearch (vector<int>& arr, int s,
    int e, int key){
    // Base Cases
case1→ if (s > e)          // key not found.
        return -1;
    int mid = s + (e-s)/2;
case2→ if (arr[mid] == key)  // key found.
        return mid;
    // right search when arr[mid] < key
    if (arr[mid] < key)
        return binarySearch (arr, mid+1, e, key);
    // left search when arr[mid] > key
    else
        return binarySearch (arr, s, mid-1, key);
}
```

s > e means invalid array, logically and array's starting element is always smaller than it's ending index.

In short (with ternary operator).

```
return (arr[mid] < key) return binarySearch (arr, mid+1, e, key);
binarySearch ( arr, s, mid-1, key);
```

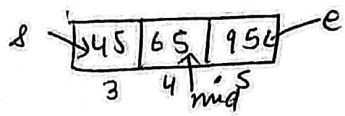


Key = 95

s > e → F

35 = = 95 → F

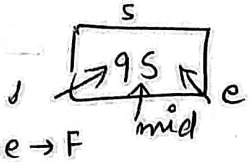
35 < 95 → T → right me



s > e → F

65 = = 95 → F

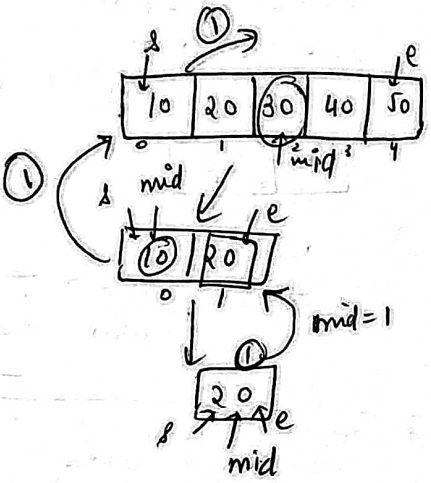
65 < 95 → True → right me.



s > e → F

95 = = 95 → True

return mid → = 5



Ques 3 → Subsequence of a string →

i/p → "abc"

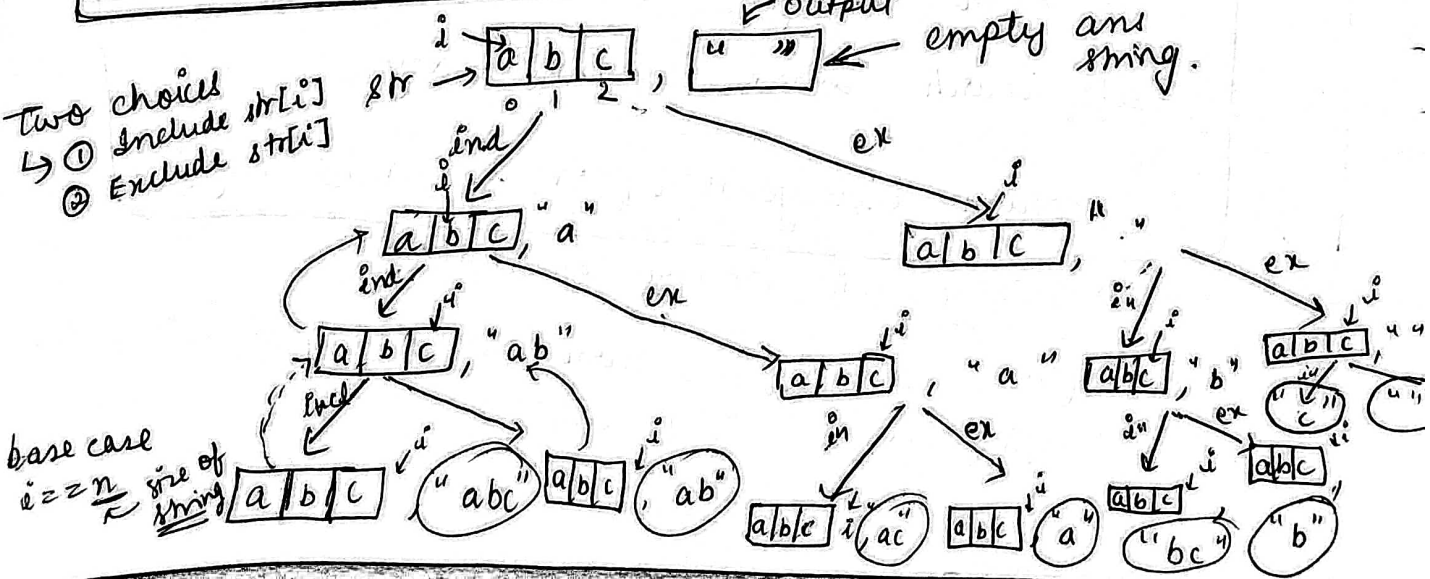
o/p → all subsequence

⇒ { }, {a}, {b}, {c}, {ab}, {bc}
{ac}, {a, b, c}

size (n) = 3 so subsequences = $2^n = 8$

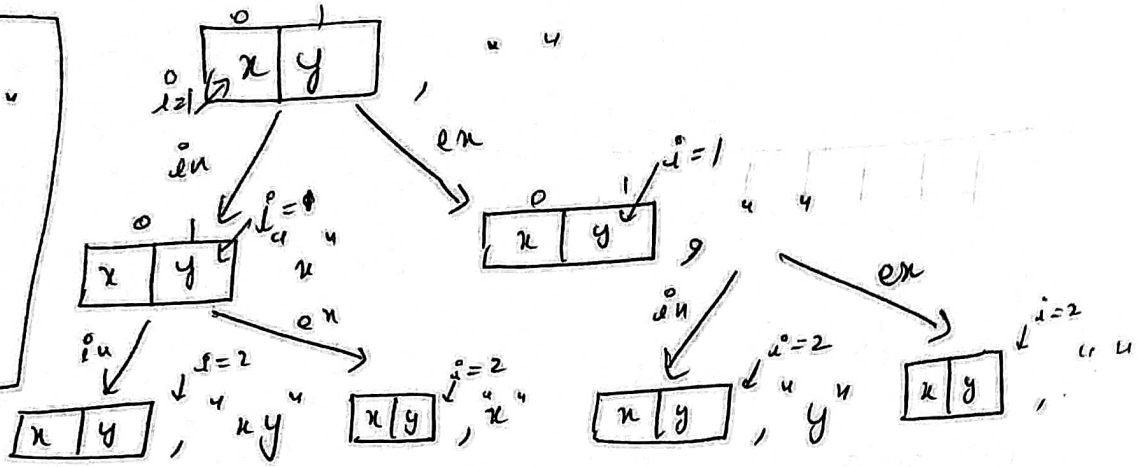
a b c
✓ ✓ ✓ → abc
✓ ✓ ✗ → ab
✓ ✗ ✗ → a
✗ ✗ ✗ →
✗ ✓ ✗ → b
✗ ✗ ✓ → c
✗ ✓ ✓ → bc
✓ ✗ ✓ → ac

Pattern → Include - Exclude / Pick - NotPick.

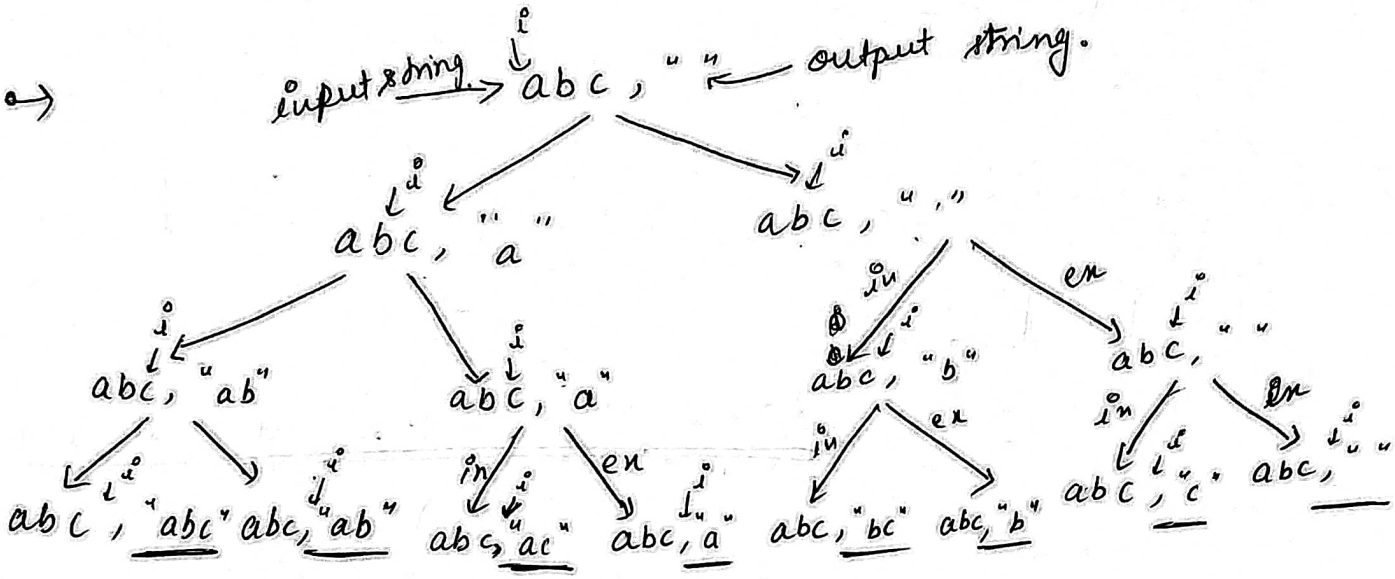


→ Include → "a" + "b" = "ab" (Concatenate)

→ Exclude → "" + "" = "" (Do nothing)



→



code →

```
void printSubsequences(string str, string output, int i){
    if(i >= str.length()){
        cout << output << endl;
        return;
    }
    //exclude
    printSubsequences(str, output, i+1);
                      ↑ output remains same.

    // include
    output.push_back(str[i]);
    printSubsequences(str, output, i+1);
}
```

H.W → 1. Subsequence with Bitmasking. (man hai to)
2. Phone keypad problem.
3. Revision.

If we have to store all subsequences in a string, then we will pass a vector of strings into the function.

```
void printSubsequences(string str, string output, vector<string> v, int i){
    if(i >= str.length()){
        v.push_back(output);
        return;
    }
```

→ Only changes in the above code.