12/03/2023

① Determine whether array is sorted in ascending order or not.

i/p → 10 20 30
o/p → True

We will be linearly traversing the array & compare i^{th} and $(i+1)^{th}$ element and if the $(i+1)^{th}$ element is less than i^{th} element, then return false otherwise by recursion we go to the next index. Hence we have simply solved one case & rest recursion will see.

Base case ⇒ If we traversed full array, then return true as it is sorted that's why false was not returned.

Code

```
bool checkSorted (int arr [], int i, int size) {
        //Base Case
        if (i == size - 1) → Traversed array as i+1 is not valid index
            return true;
        if (arr [i] > arr [i+1]) //1 case solve
            return false;
        // Recursive relation
        return checkSorted (arr, i+1, size);
}
```

If we have written i == size as the base case, then when i = size-1, base case

is not satisfied and hence i+1 i.e size-1+1 will be size & arr[size] is not a valid index and hence the base case was written as i == size-1. So last index that will be compared in the processing will be i+1 i.e size-2+1 ⇒ size-1 (last index) and i i.e size-2 which are valid indexes.

Note → Whenever the return type of function is void, then we can simply write

return checkSorted(---);
↳ not needed

② Binary Search using Recursion.

We have already seen the iterative soln and dry run of binary search. Hence we need to convert iterative to the recursive approach.

Code

```
int searchElement(vector<int>& arr, int s,
                  int e, int target){
    // Base case
    if(s > e)
        return -1; // Element not found
    // 1 case solve
    int mid = s + (e-s)/2;
    if(arr[mid] == target)
        return mid;
```

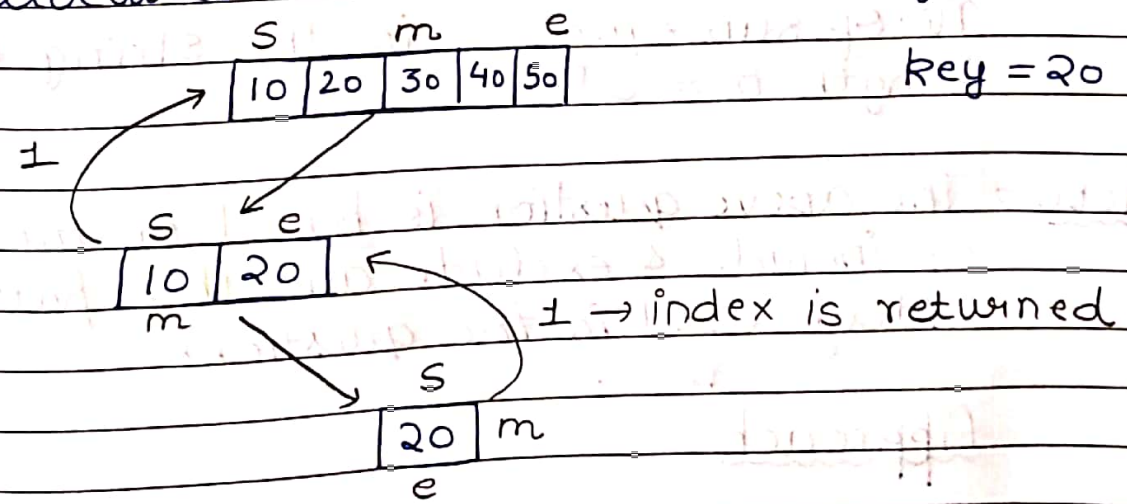
```
//Recursive relation
else if (arr[mid] > target) {
    return searchElement(arr, s, mid-1, target);
                                  ↳ search in left
}
else {
    return searchElement(arr, mid+1, e, target); ↳ search in right
}
}
```

Cases in binary search

1) arr[mid] == target → return mid.
2) arr[mid] > target → The element is present at the left side so update end = mid-1;
3) arr[mid] < target → The element is present at the right side so update start = mid+1;

Note → When we say entry is maintained in the call stack, this means that memory has been allocated to variables inside the function.

Tree



(3) Subsequences of a string

By this question, we will be able to understand recursion better.

i/p → "abc"
o/p →

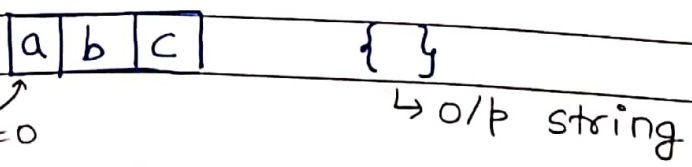
Subsequence is basically including some characters & excluding some characters but the ordering remains same as that of i/p string.

a	b	c	
✓	✓	✓	abc
✓	✓	×	ab
✓	×	×	a
×	×	×	–
×	✓	×	b
×	×	✓	c
×	✓	✓	bc
✓	×	✓	ac

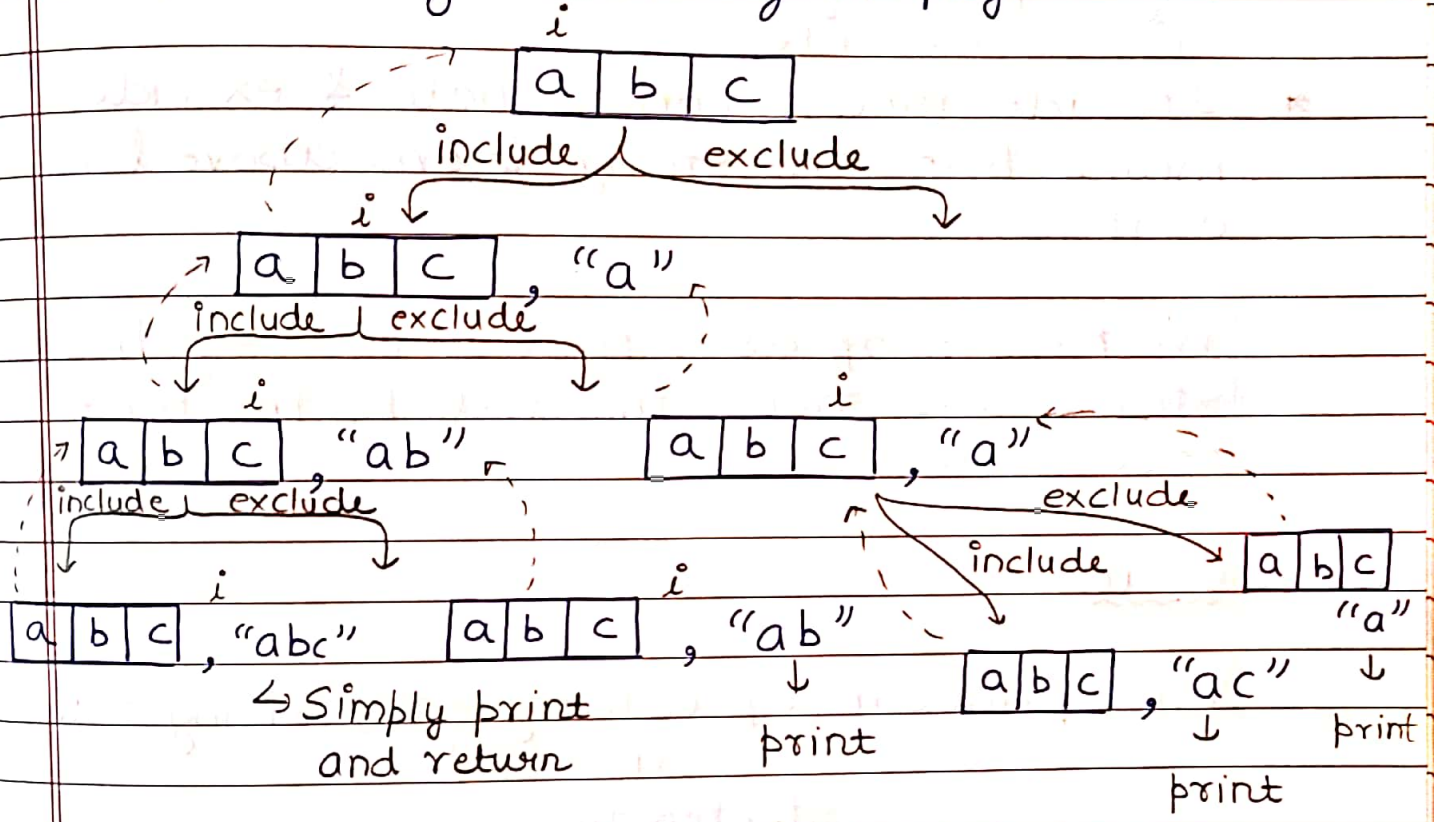
No. of subsequences in the string of length $n = 2^n$.

Note → The above question is based on the pattern of include & exclude and this pattern is widely used in the questions

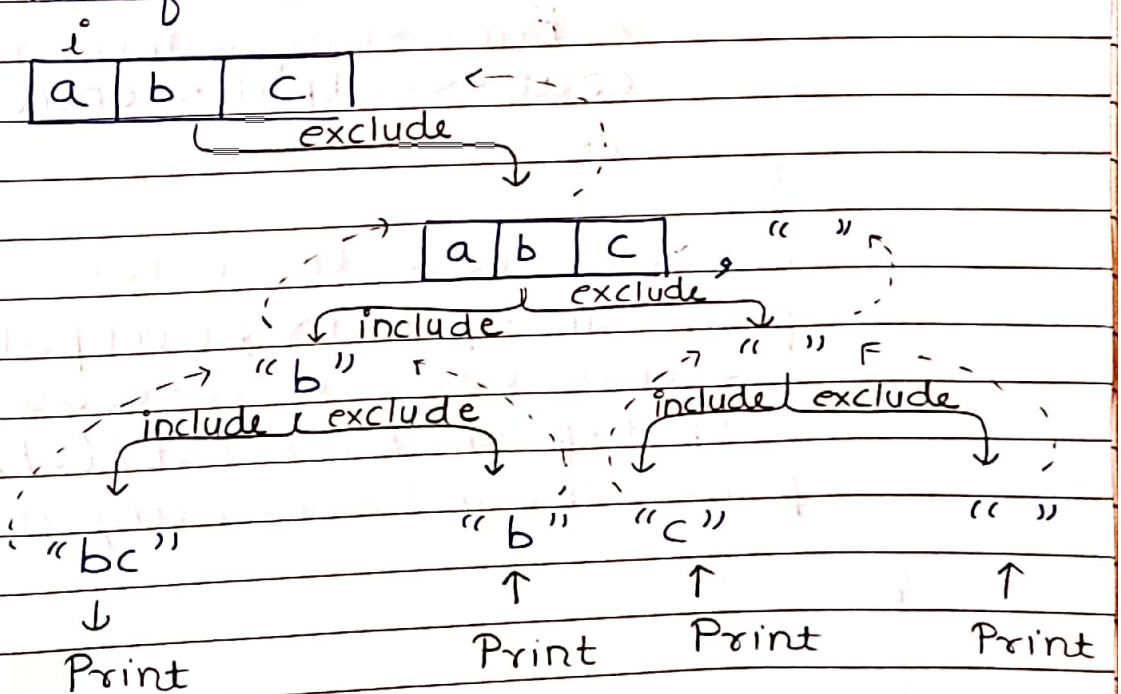
Approach



Output string is initially empty.



The above was for the left subtree.



The above was the right subtree.

* Left part of each subtree will be call for include the character at ith index

* Right part of the subtree will be the call for exclude
* Include means concatenate & exclude means bring the string from above level as it is.

We have stop as i moves out of the string and hence this will be the base case condition.

Code

```
void printSubSeq (string str, string output,
                  int i) {
                    str.length()
    if (i ==                ) { //Base Case
        // Print string & then return
        cout << output << endl;
        return;
    }
    // Exclude call (As it is)
    printSubSeq (str, output, i+1);
    // Include call (Concatenate)
    output.push_back (str[i]);
    printSubSeq (str, output, i+1);

}
```

What is the use of output string?
Output string is basically a temperory string & we use this because we don't want to change in original string as it will lead to loss of characters.

Note → `vector<int> v;`
`v[0] = 5;`
} Gives runtime error as we don't know whether v[0] exists or not.

To fix this problem we need to define the size.

`vector<int> v (10);`
`v[0] = 5;`
} This will work