* sorting *

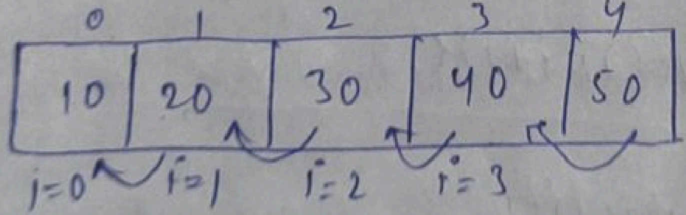
* To check an array is sorted or not through recursion

```
bool isSorted (vector <int> & arr,  int i) {
    //base case
    if (i == arr.size() - 1){
        return true;
    }
    if (arr[i+1] < arr[i]){
        return false;
    }
    return isSorted.(arr, i+1);
}

int main(){
    int size;
    cin size;
    vector<int> arr(size);
    for (int i=0 ; i< arr.size(); i++){
        cin >> arr.size();
    }
    int i = 0
    if(isSorted (arr, i)){
        cout<< "Array is Sorted";
    }
    else{
    cout<< " Array is not sorted";
    }
```

output

```
10 20 30 40 50
Array is sorted

10 20 30 50 40
Array is Not sorted
```

* As we have passed an array by reference bcz any changes will be made inside the original array no copy of an array will be created



20>10 T   30>20T   40>50T   50>40T

```
if (arr[i] > arr[i]){
        true }
else false;
```

if i reach to last index after traversing to each elements.
than we can say that i have checked all i+1 index
∴ array is in sorted order. return true.

0	1	2	3	4
10	20	30	50	40
i=0	i=1	i=2	i=3	i=4

10<20T → 30>20T → 50>30T → 40>50F (return false)

⁼* Binary search with Recursion:

```
int binary search (vector<int>&arr, int start, int end, int &key)
{
  if (start > end){
       return-1.
  }
  int mid= start+(end-start)/2;
  if (arr[mid]==key){
       return mid.
  }
  if (arr[mid] < key){
       return binary search (arr, mid+1, end, key);
  }
  else ~~if~~{
    return binary search (arr, start, mid-1, key).

}

int main(){
     int size;
     cin>> size;
     vector<int> size.
     for(int i=0 ; i<arr.size(); i++){
          cin>> arr[i];
     }
     int start=0, end= arr.size()-1;
     int key.
     cin>> key.
     int index = binary search (arr, start, end, key).
     cout << index ;
```

output

→ 10 20 30 40 50 60 → Input

60 → key.

5 → index found.

→ 10 20 30 40 50 60 → Input .

70 Key.

-1 → index Not found.

* subsequences of a string:

① I/P :→ "abc"

O/P :→ print all subsequence i.e

<_>., <a>, <b>, <c>, <a,b>, <a,c>, <b,c> } →

<a,b,c> = 8

Powerset

The Question uses famous pattern named include - exclude pattern.

② I/P → "xy"

O/P → <_>, <x,y>, <x,_>, <y> = 4

we can observe that it is power set i.e 2^n

n here stands for ~~no. of~~ length of string.

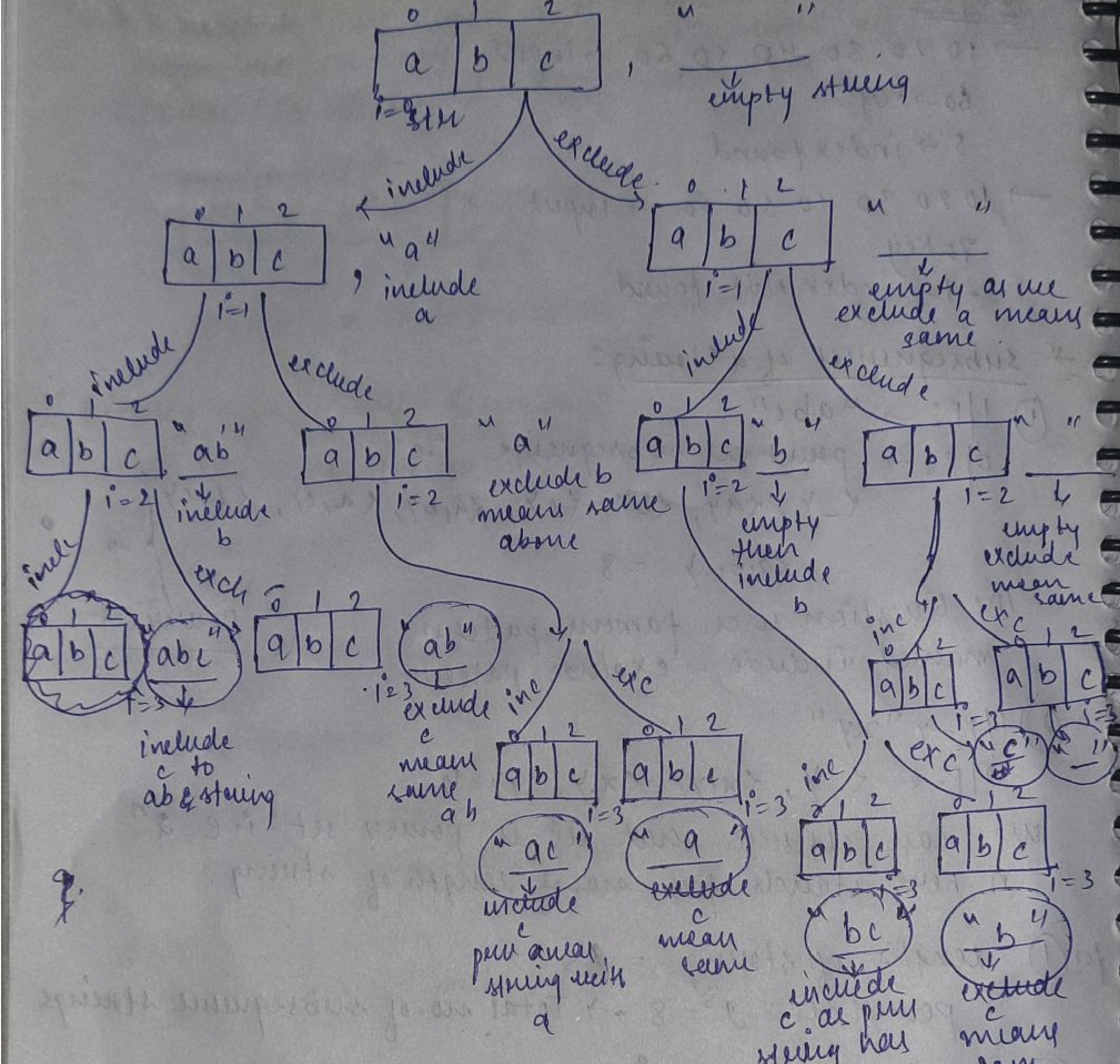
for ① length of string = ~~2~~3

power set = 2^3 = 8 → Total no. of subsequence strings

for ② length of string = 2

power set = 2^2 = 4 → Total no. of subsequence strings.

* Include- Exclude pattern means we will have an empty string which means we ~~are~~ creating extra space for new string one time we include a character to empty string & one time we will exclude that character in an empty string. we will iterate through i=0 for every character in a string once we include- exclude move forward i.e i=i+1;



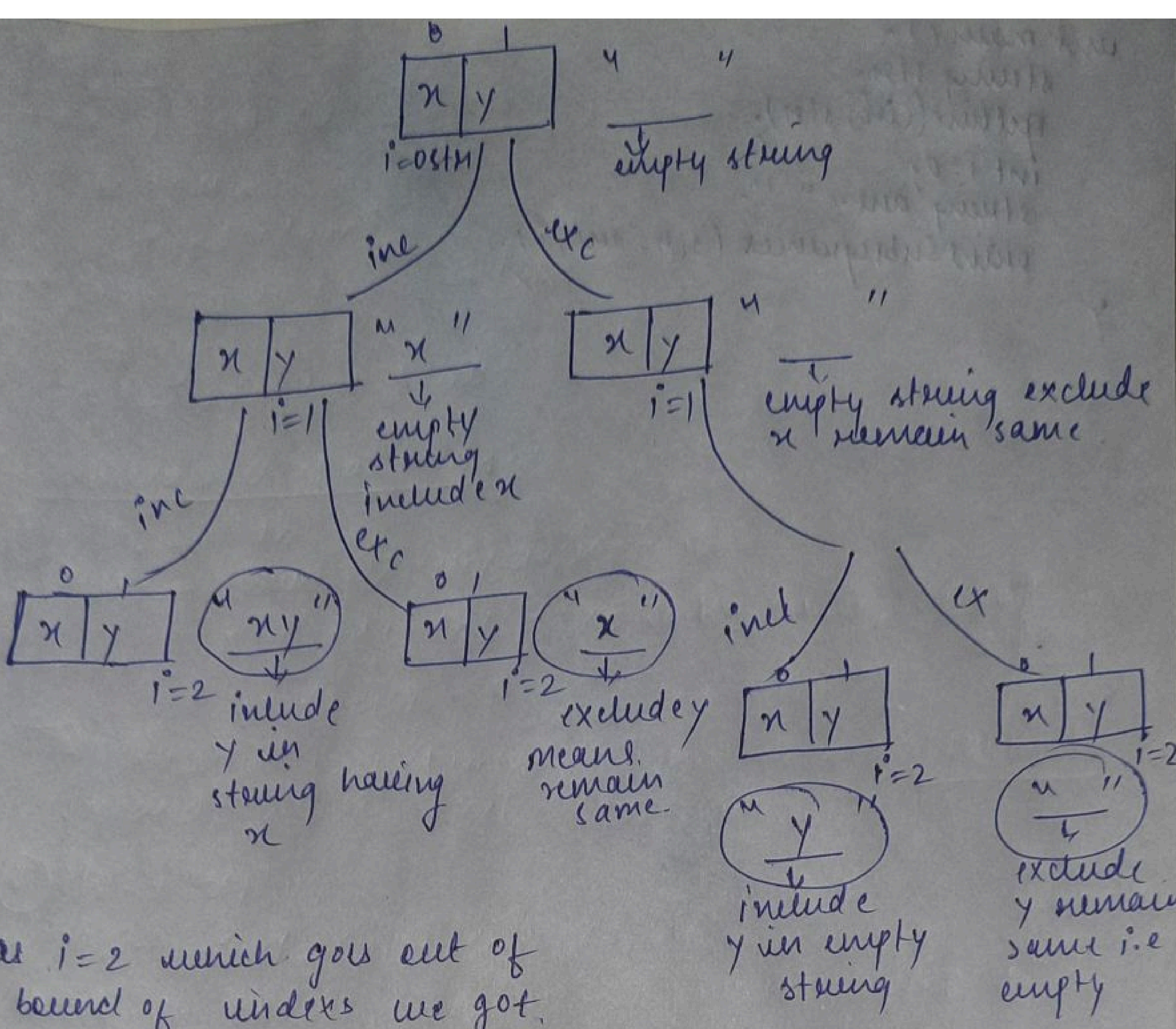
As i goes out of the bound i.e i=3 we will print all the base cases that will be subsequences of a string.

8 str = "abc"

→ {abc}, {a}, {b}, {c}, {ac}, {bc}, {ac}, {_}

8 subsequences

print all the circle part when the base case hit i.e when index goes out of bound.


0 1
x y
i=0str
" "
empty string
inc
exc
"x"
empty string include x
i=1
" "
empty string exclude x remain same
inc
exc
inc
ex
i=2
"xy"
include y in string having x
"x"
exclude y means remain same
"y"
include y in empty string
" "
exclude y remain same i.e empty

as i=2 which goes out of bound of index we got base case output the string i.e circle part $2^2 = 4$ subsequences.

```
void printsubsequences(string &str, string& ans, int i){
        //base case
        if(i>= str.length()){
              cout << ans << " ";
              return;
        }

        //exclude
        printsubsequences(str, ans, i+1);

        // include
        ~~printsubsequences( str, ans~~
        ans.push_back(str[i]);
        printsubsequences(str, ans, i+);

}
```

```
int main() {
    string str;
    getline(cin, str);
    int i=0;
    string ans= "";
    printSubsequences(str, ans, i);
}
```