DnC Lecture → 2

Quick Sort

i/p ⇒

8	3	4	1	20	50	30
0	1	2	3	4	5	6

Algorithm

↳ 1 no. of ko usbi sahi jagah pe pahucha diya baaki recursion dekh lega.

↳ logic is called partition logic

A → choose pivot
B → pivot placed at right position
C → pivot

chote elements

bade elements

8	3	4	1	20	50	30

1	3	4	8	20	50	30

pivot

Recursion sort karega

↳ Recursion sort karega

Partition algo :

A → pivot = 8
B → pivot → right place → pivot → count small = 3
↓
swap karo

8	1	3	4	20	50	30
0	1	2	3	4	5	6

(i)

1) A → pivot choose ⇒ pivotIndex = 0

2) B → pivot → sahi position → count no < pivot element
4 ko place karo
here there are 3 no. less than pivot
so 8 will be at 4th position.

3) swap(arr[pivotIndex], arr[s + count])

4	1	3	8	20	50	30
0	1	2	3	4	5	6

4) Recursive call left

4	1	3
0	1	2

right

20	50	30
4	5	6

Code

```
int main() {
  int arr[] = {8, 1, 3, 4, 20, 50, 30};
   int n = 7;
   int s = 0;
   int e = n-1;
   quicksort(arr, s, e);
    return 0;
}

void quicksort(int arr[], int s, int e){
// base care
    if(s>=e)
      return;

  // partition logic
int p = partition(arr, s, e);

  // recursive call.
    // left
     quicksort(arr, s, p-1);

  // right
    quicksort(arr, p+1, e);

}
   int partition(int arr[], int s, int e){
//choose pivot
    int pivotIndex = s;
   int pivotElement = arr[s];
```

// right position of pivot and place there

```
int count=0;
for(int i=s+1; i<=e; i++){
    if(arr[i]<=pivotElement){
        count++;
    }
}
```

// jab loop ke bahar aayo to correct index ready ehh.

```
int rightIndex = s+count;
swap(arr[pivotIndex], arr[rightIndex]);
pivotIndex = rightIndex;
```

// left me chote, & right me bade..

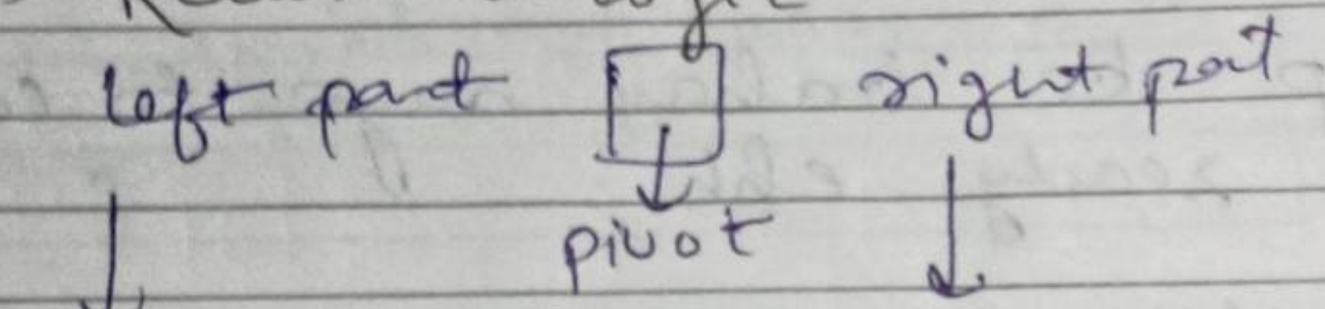
```
int i=s;
int j=e;
while(i<pivotIndex && j>pivotIndex){
    while(arr[i]<=pivotElement){
        i++;
    }
while(arr[j]>pivotElement){
    j--;
}
if(i<pivotIndex && j>pivotIndex)
    swap(arr[i], arr[j]);
}
return pivotIndex;
}
```

By 2 chiz ka dhyaan rakho

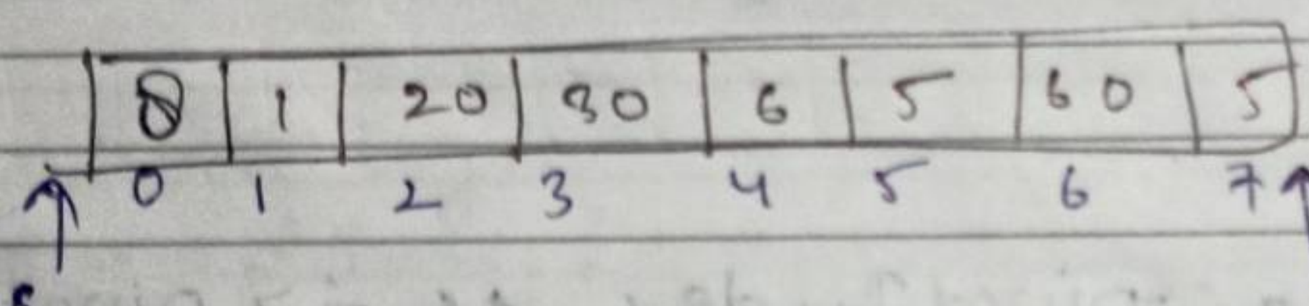
↳ Partitioning logic

↳ for placing pivot at right position.

↳ Recursive logic:



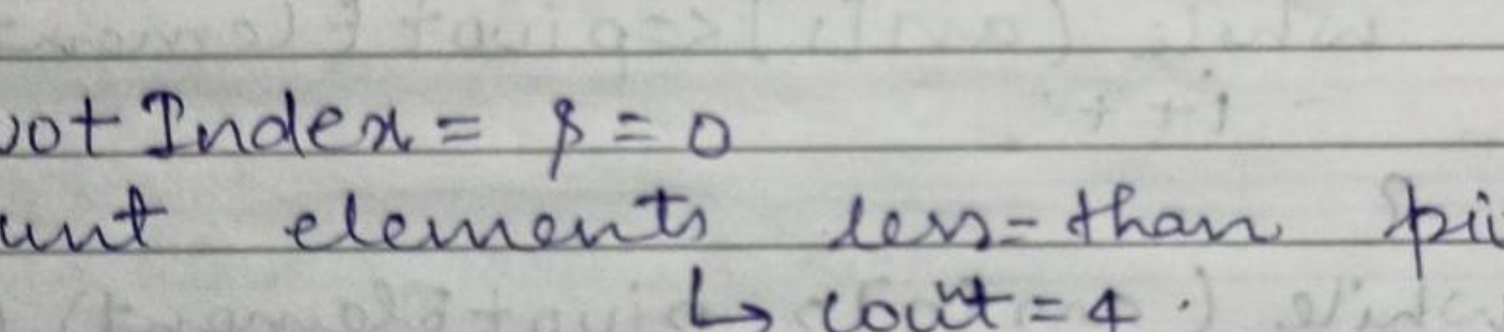
Dry Run Quick Sort

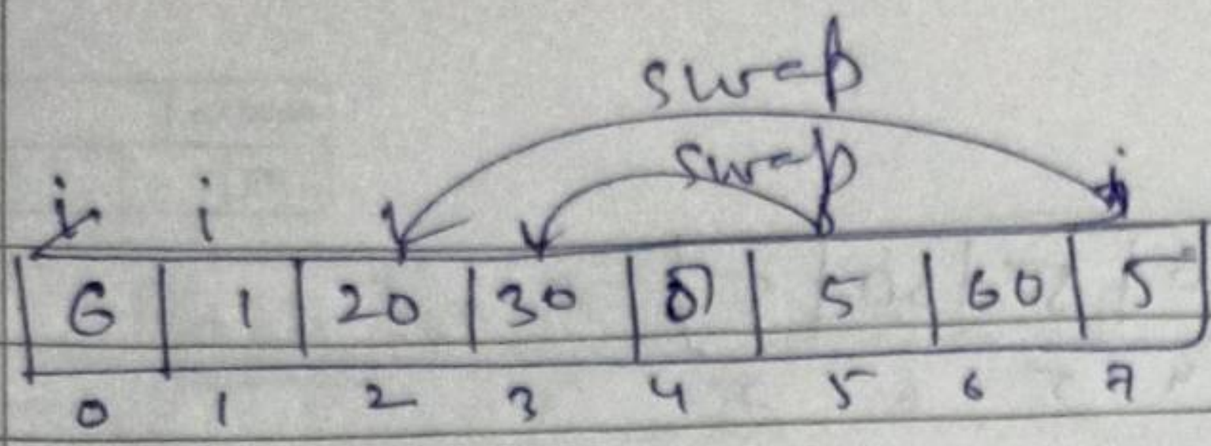


1) pivotIndex = s = 0
2) count elements less-than pivot.
↳ count = 4.

rightIndex = s + count
= 0 + 4
= 4.

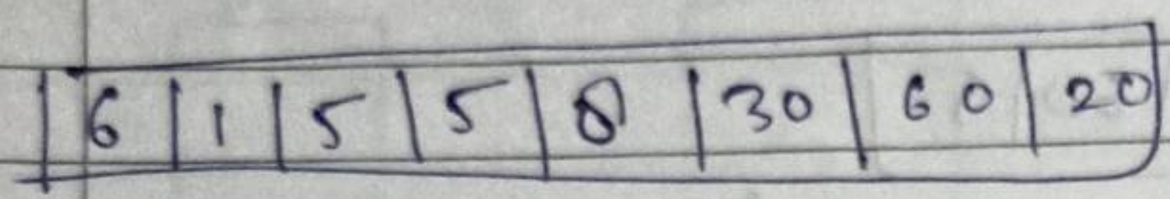
swap[arr[0], arr[4]];





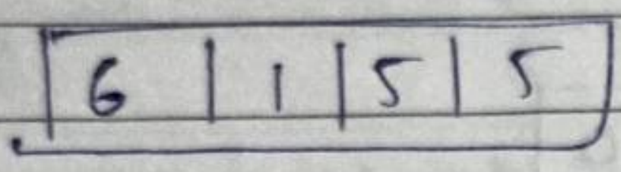
wrong element

↳ left part
me element > pivot



↳ right part

element < pivot.



P.I

pivotIndex = 0
count = 3
swap(arr[pivotIndex], arr[s+count]);

Now Recursive call will sort the arrays of left side & right side.

8	1	20	30	6	5	60	5
0	1	2	3	4	5	6	7

Pivot

6	1	20	30	8	5	60	5

wrong Number swap around pivot.

6	1	5	5	8	30	60	20

1 5 5 | 6 | 8 | 20 | 30 | 60

1 | 5 5

Sorted array

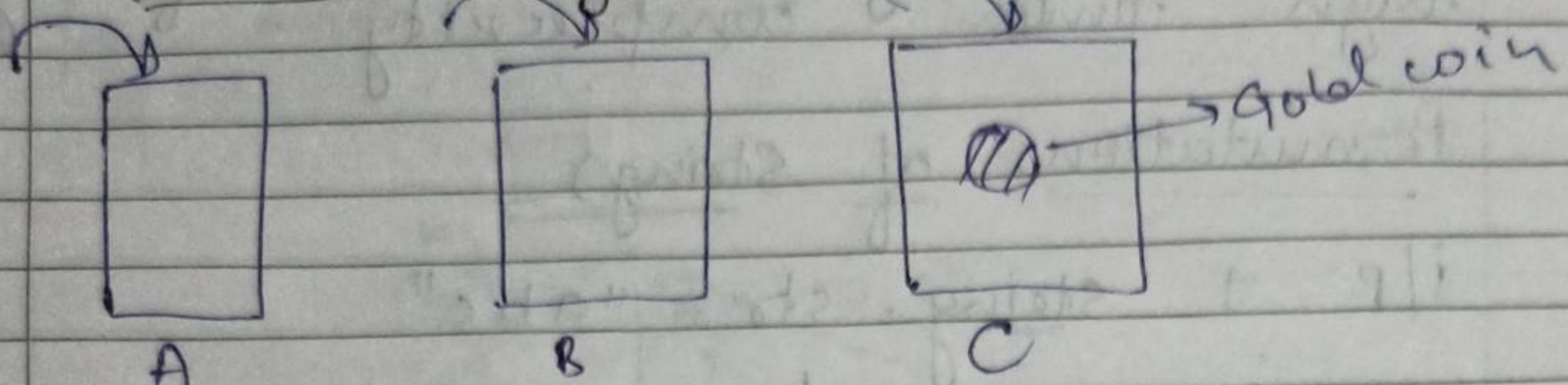
1	5	5	6	8	20	30	60

Time Complexity ⇒ O(nlogn)

Reverse wein ⇒ $O(n^2)$

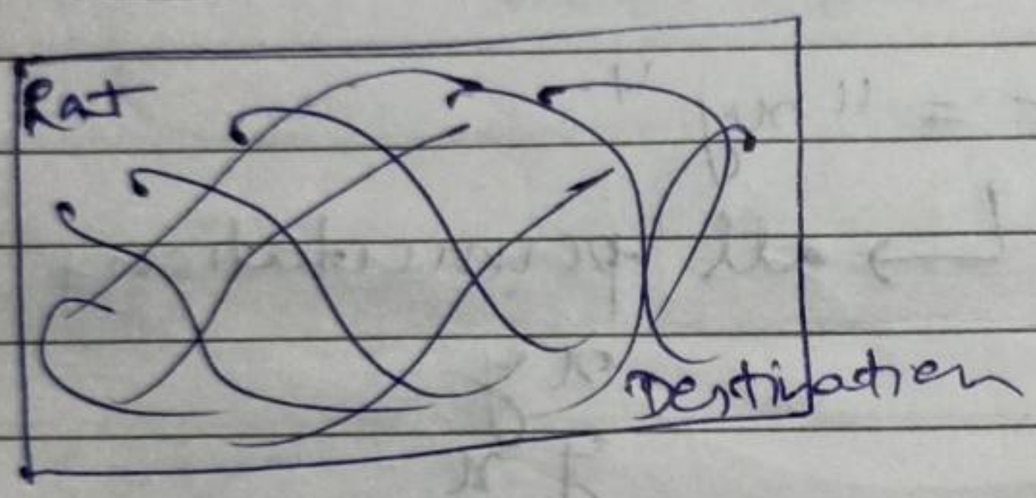
PAGE No
DATE

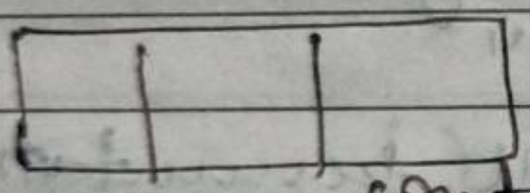
Back Tracking



Ek Ek karke check karenge.
In backtracking we explore all types of solution. It is a problem solving technique properly based on recursion.

Eg. Rat in a maze



eg. 
Password of 3 digit that contains only digit from 0-9
In back tracking we chelk all sol form
000
001
999

★ Backtracking gives solutions of very high time complexity.

Permutations of Strings

i/p → string str = "abc"

↳ all permutations

abc
bac
bca
cab
cba
acb

ip → string str = "xy"

↳ all permutations

xy
yx

ip → string str = "a"

↳ all permutations

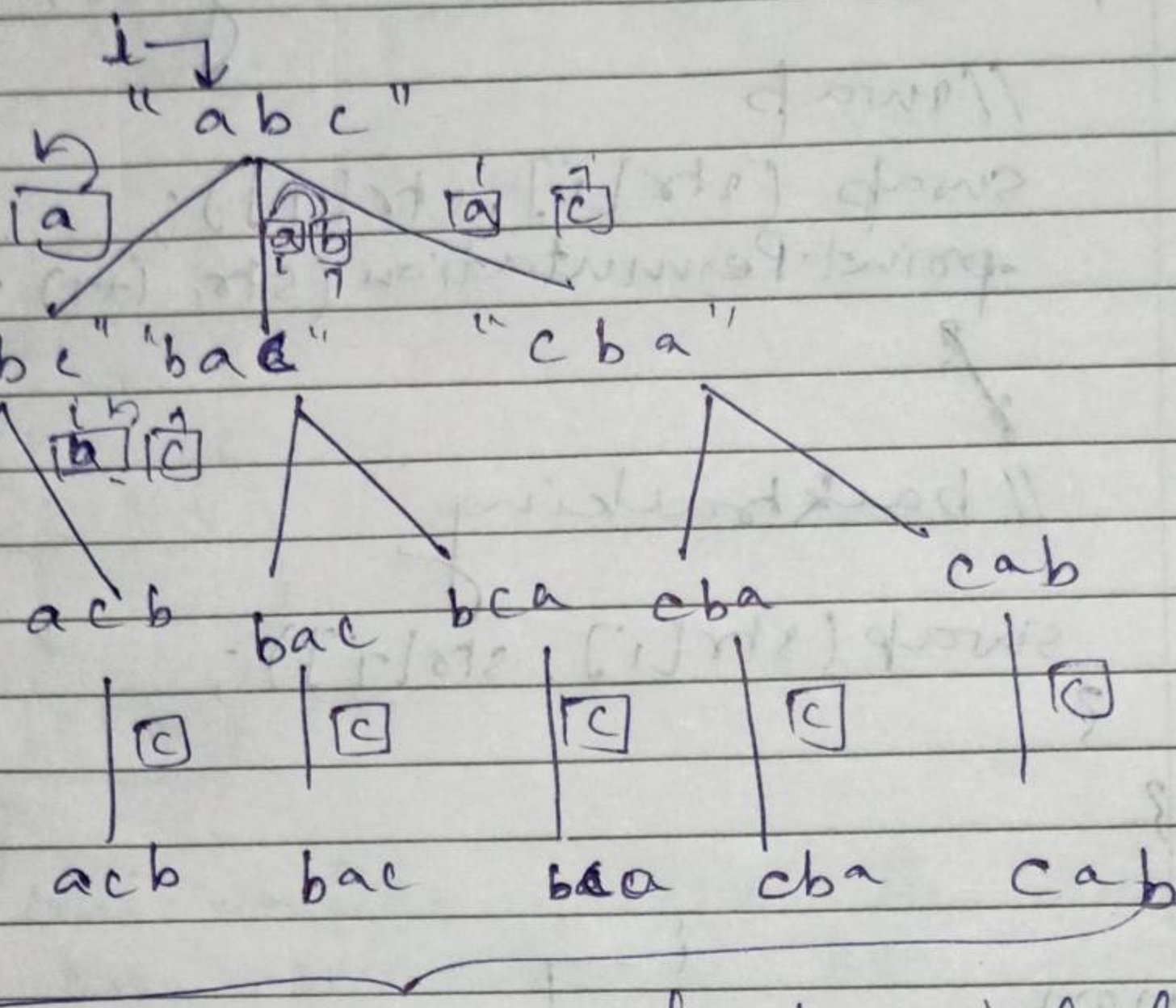
a

ip → string str = "abcd"

all permutations

abcd	bacd	cabc	dabc
abdc	badc	cacb	dacb
acbd	bcad	cbad	dbac
acdb	bcda	cbda	dbca
adcb	bdac	cdab	dcab
adbc	bdca	cdba	dcba

Har ek character har ek position pe aana chahta hai.



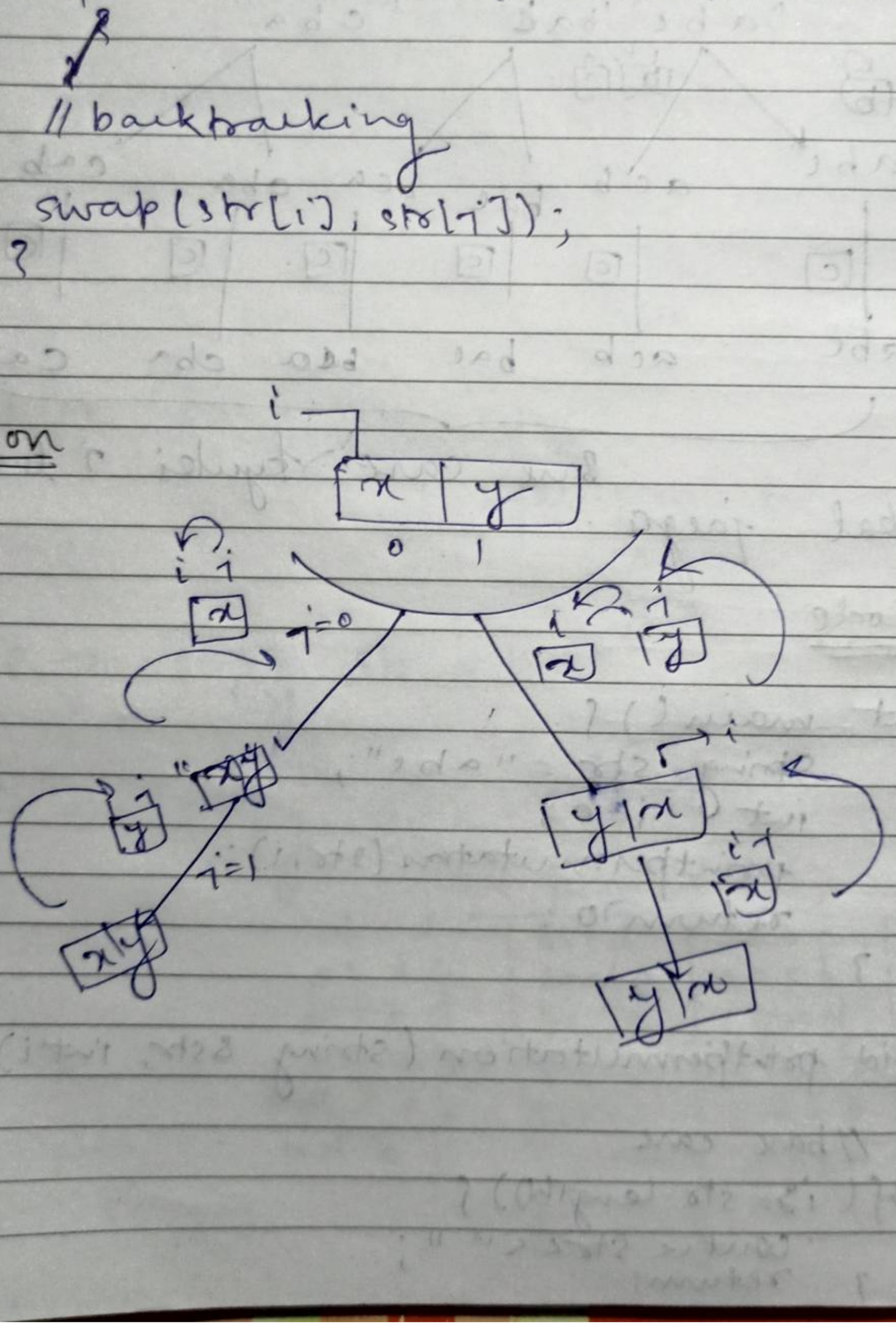
Base case tyuki i bahar nikal jaega.

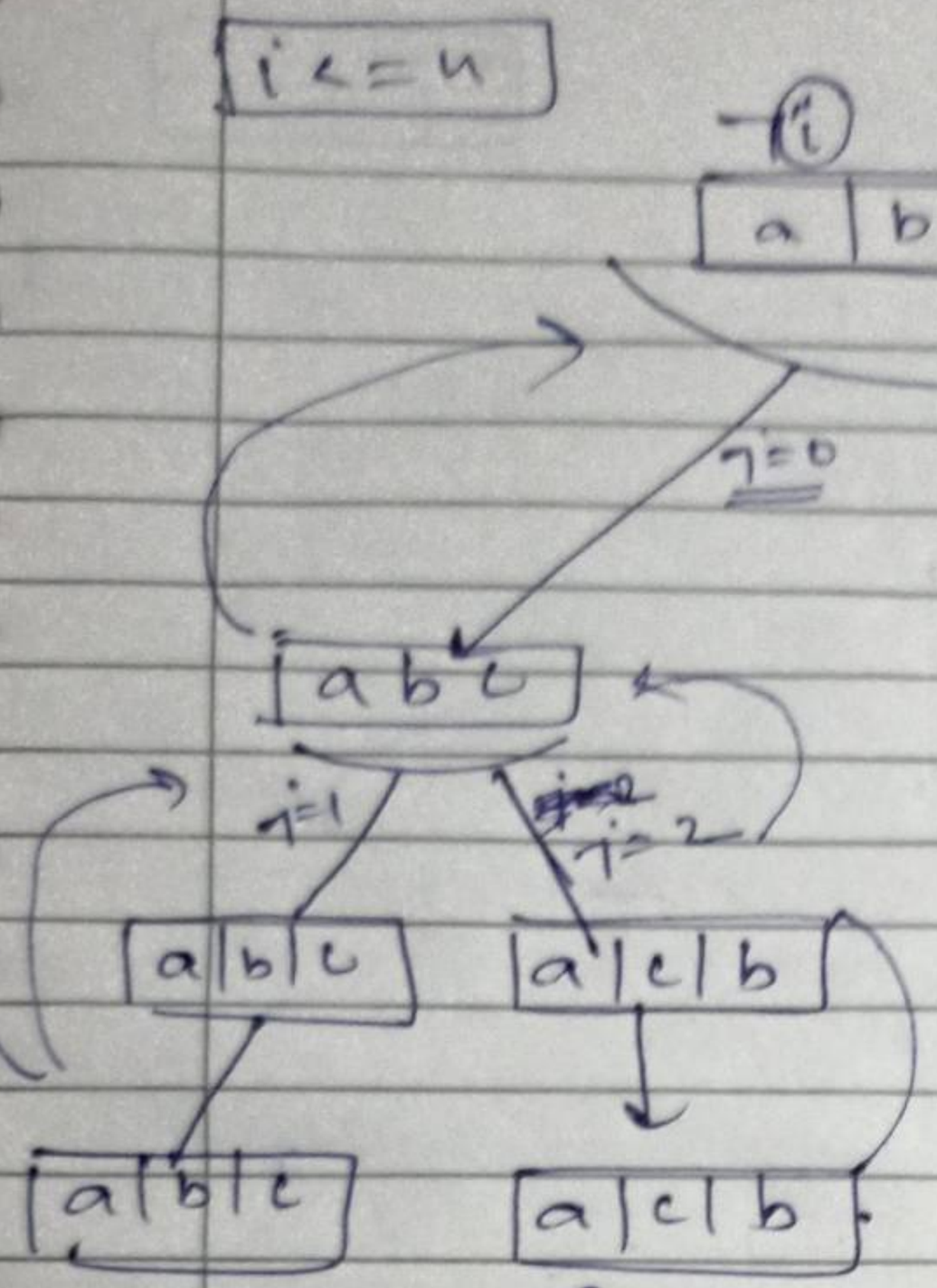
Code

```
int main(){
    string str = "abc";
    int i=0;
    printpermutation(str, i);
    return 0;
}

void printpermutation(string &str, int i){

    //base case
if(i>=str.length()){
    cout<<str<<" ";
    return;
}
```

```
//swapping
for(int i=0; i< str.length(); i++) {
    //swap
    swap (str[i], str[i]);
    printPermutation(str, i+1);
    //backtracking
    swap(str[i], str[i]);
  }
}
```

Reason





Ib bhi wapas jaunga to original string ko recreate kar dunga.

Time complexity → Homework

Dry Run → xy^2, pqrs.