19-03-2023

Divide & conquer (class 2)

Date.....

* **Quick Sort** :- The name "Quick sort" comes from the fact that, Quick sort is capable of sorting a array of data elements twice or thrice faster than any of the common algorithms.
It is one of the most efficient sorting algorithm & is based on the splitting of an array (partitio-n) into smaller ones & swapping (exchange) based on the comparis-on with "pivot" element selected.
This algorithm is an in-place sort so no additional space is required for sorting. It is also known as "partition Exchange sort"

working of Quick Sort :-
we choose an element from the list & place at its proper position in the list, i.e., at the position where it would be in the final sorted list.
we call this element as pivot & it will be at its proper place, if :-

i) all the element to the left of pivot are less than or equal to the pivot, and
ii) all the element to the right of pivot are greated than or equal to the pivot.

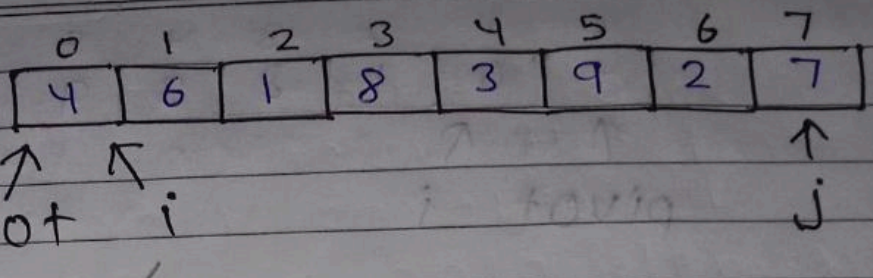
Any element of the array can be taken as pivot, but for convenience the first element is taken as the pivot.
Suppose, our list is [4, 6, 1, 8, 3, 9, 2, 7]
If we take 4 as the pivot then after placing at its proper place, the list becomes [1, 3, 2, 4, 6, 8, 9, 7]
Now, we can partition this list into two sublists, based on this pivot & these sublists are [1, 3, 2] and [6, 8, 9, 7].
Now, we can apply the same proce-dure to these two lists separately

Approach we follow in Quick sort :-

set pivot at 0^{th} index.
set i at 0^{th} index.
set j at $n-1^{th}$ index.
while (i < j),
If arr[i] <= arr[pivot], then i++.
If arr[j] > arr[pivot], then j--.
when i and j both stops, then we check if i < j, then swap arr[i] and arr[j].
If i > j, then swap arr[j] and arr[pivot].
After this we recursively call,
quicksort(arr, start, j-1);
quicksort(arr, j+1, end);

Date.....

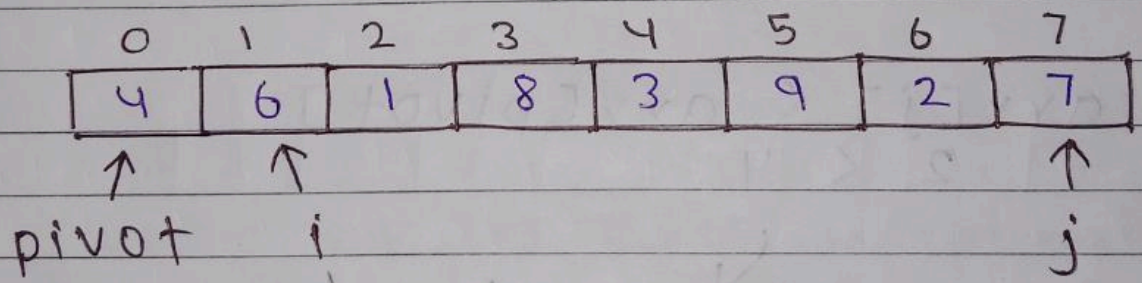
Example:-



jab tak arr[i] pivot ke <= hoga, tb tk i++ krte rhege, else i ruk jayega.

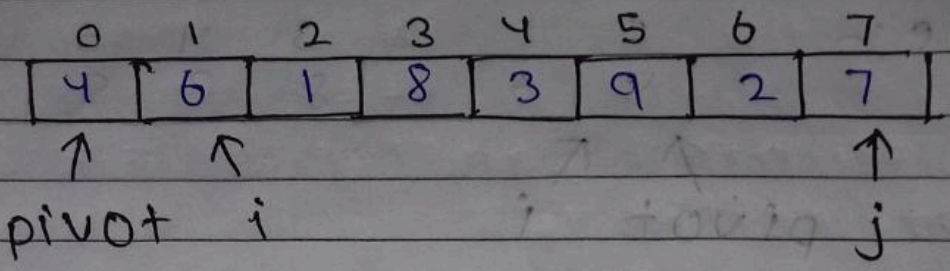
, i.e., arr[pivot] = 4 & arr[i] = 4
arr[i] <= arr[pivot]
4 <= 4, so, i++.

arr[pivot] = 4 & arr[i] = 6
arr[i] <≠ arr[pivot]

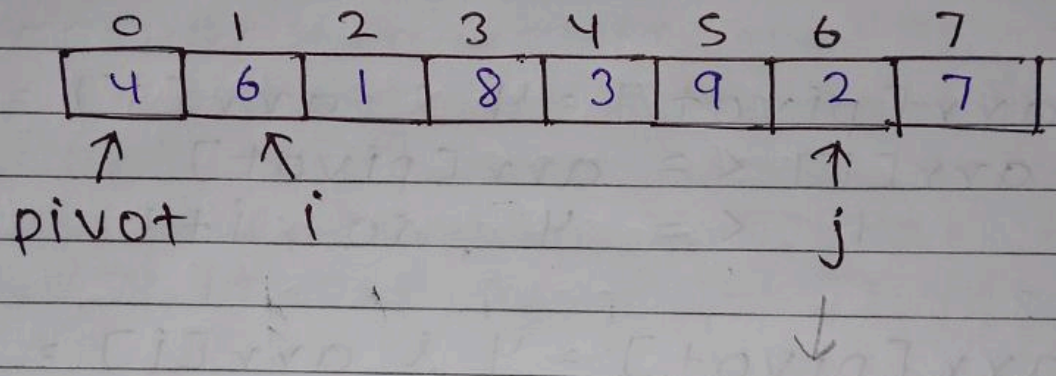


arr[i] > arr[pivot] hogya to i ruk jayega & j start hoga, jab tak arr[j] > arr[pivot] hoga, tab tak j--krte rhege, else j ruk jayega.

Date.....



, i.e., arr[pivot] = 4 & arr[j] = 7
arr[j] > arr[pivot]
7 > 4, so j--.

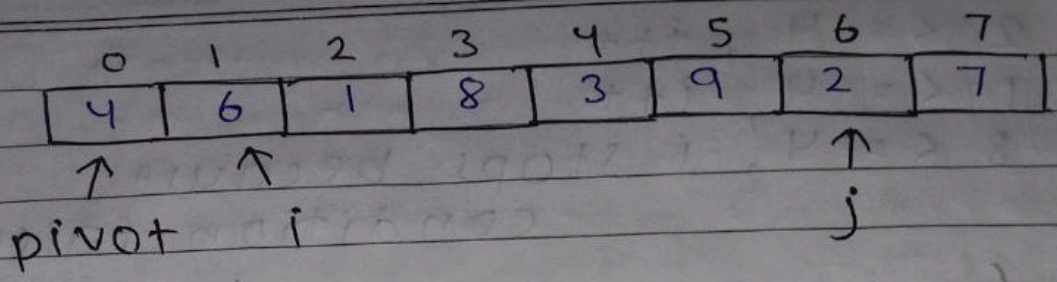


now arr[j] < arr[pivot].

, i.e., arr[j] < arr[pivot]
2 < 4

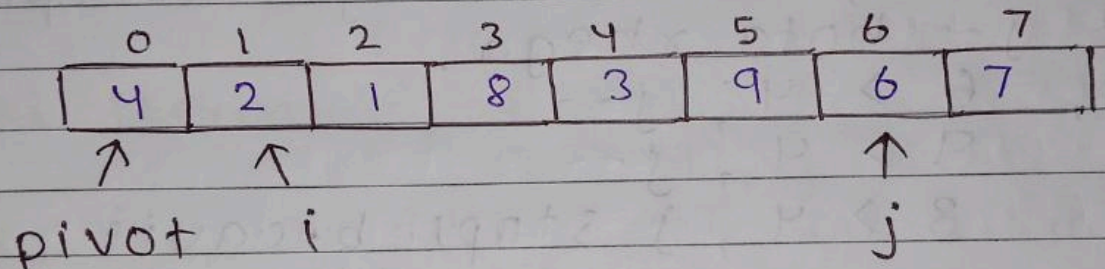
we stop here because arr[j] should be greater than arr[pivot], if not then we stop.

Date.....



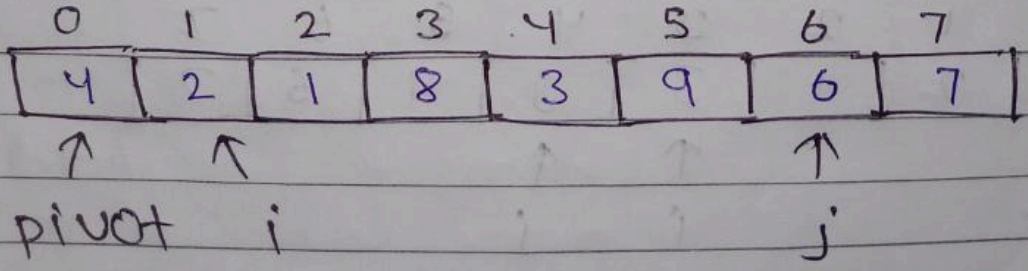
now i & j both are at this position

Now, check if i < j then swap arr[i] and arr[j]



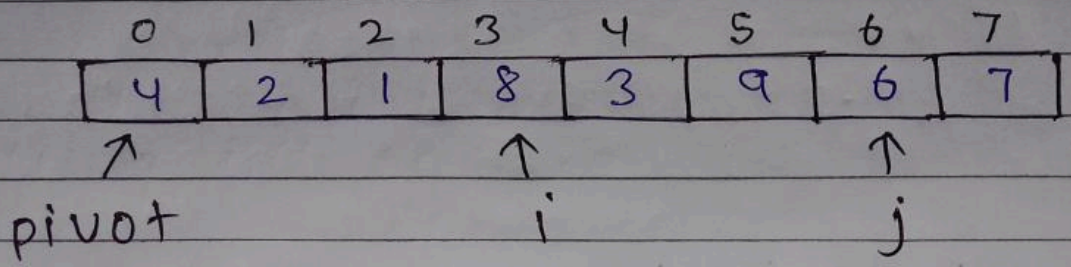
after swapping

Now, when values are swapped, we again follow the precedure, jab tk arr[i] chota ya equal rhega arr[pivot] se, tb tk i++ hota rhega Means,

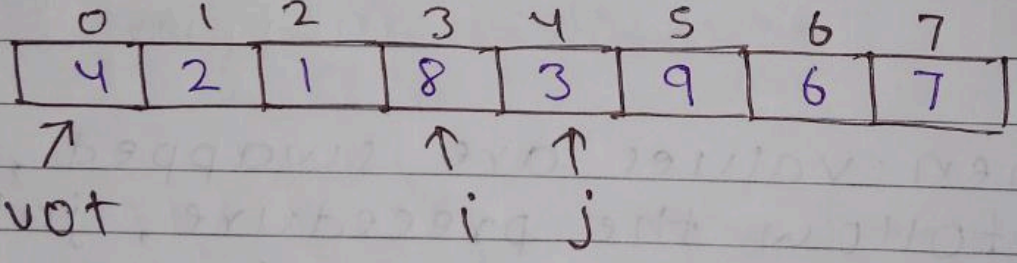


Teacher's Sign

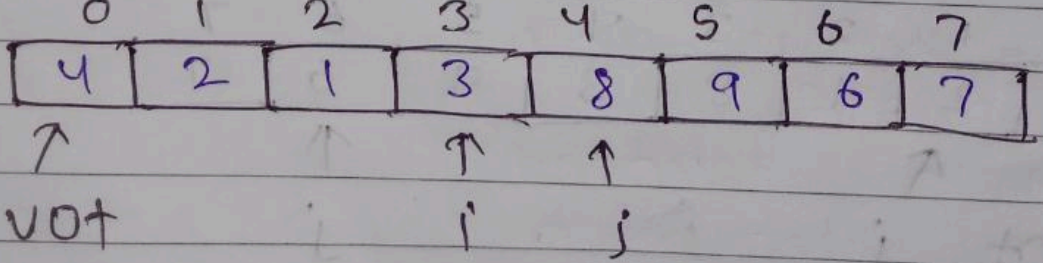
, i.e; 2 <= 4, i++
1 <= 4, i++
8 <= 4, i stops because condition false



Now, j will start its processing.
jb tk arr[j] bada rhega arr[pivot]
tb tk j-- hota rhega.
Means, 6 > 4, j--
9 > 4, j--
3 > 4, j stops because condition false.



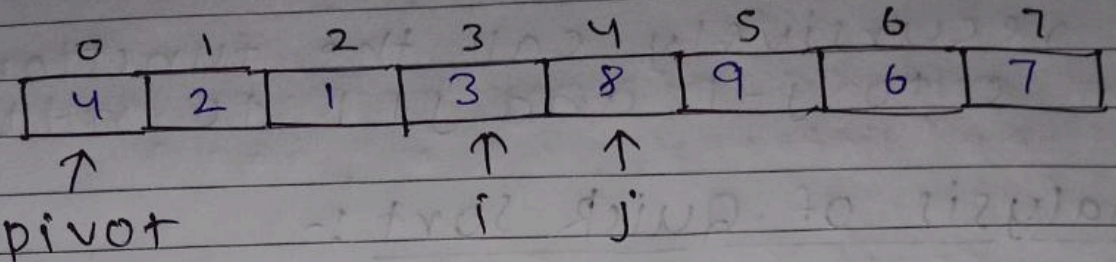
Now, i < j, then swap arr[i] & arr[j].



after swapping

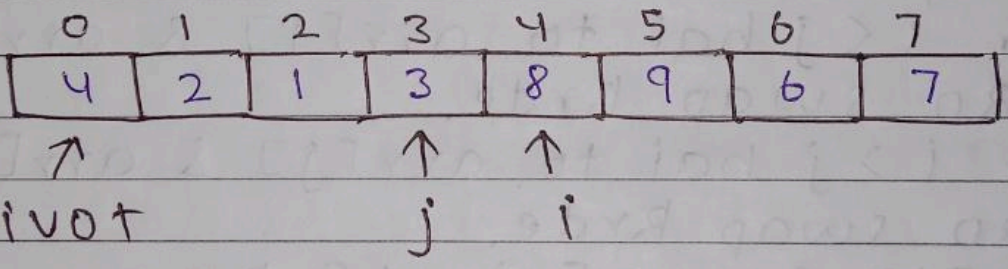
Date.....

Now, values are swapped, we again start the procedure.

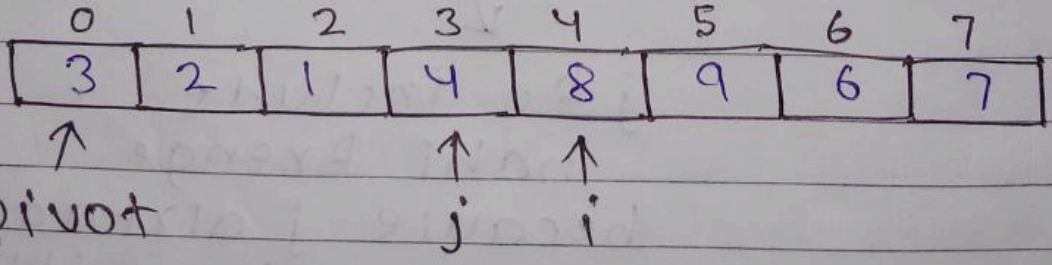


i Ki iteration, 3 <= 4, i++
8 <= 4, i stops because condition false.

j Ki iteration, 8 > 4, j--
3 > 4, j stops because condition false.



Now, i > j, if i > j, then swap arr[j] with arr[pivot].



after swapping arr[j] with arr[pivot]

Spiral

Teacher's Sign.....

Date.....

After swapping arr[j] with arr[pivot] our pivot element is place at its correct position.
Now recursively call the function from 0 to j-1 and j+1 to n-1.

* Analysis of Quick Sort :-

i) pivot = 0^{th} index
i = 0^{th} index
j = 0^{th} index.

ii) i ko tb tk increment krte rho jb tk arr[i] <= arr[pivot] hoga. Then,

iii) j ko tb tk decrement krte rho jb tk arr[j] > arr[pivot] hoga. Then,

iv) Agr i < j hai to arr[i] & arr[j] ko swap krde.
Agr i > j hai to arr[j] & arr[pivot] ko swap krde.

v) arr[j] & arr[pivot] ko swap krne ke baad recursively call krdenge 0 to j-1 tk and j+1 to n-1 tk.

↓

j ko include nahi krenge because j already apni sahi position pe aa chuka hai.

code:-

```
#include<bits/stdc++.h>
using namespace std;
void quickSort(int arr[], int
                start, int end)
{
    int i, j, pivot;
    if(start < end)
    {
        pivot = start;
        i = start;
        j = end;
        while(i<j)
        {
            while(arr[i] <= arr[pivot])
            {
                i++;
            }
            while(arr[j] > arr[pivot])
            {
                j--;
            }
            if(i<j)
            {
                swap(arr[i], arr[j]);
            }
        }
        swap(arr[j], arr[pivot]);
        quickSort(arr, start, j-1);
        quickSort(arr, j+1, end);
    }
}
```

jb tk arr[i] <= arr[pivot] hoga tb tak i++.

jb tk arr[j] > arr[pivot] tb tk j-- hoga.

Date.....

```
int main()
{
    int arr[] = {5, 7, 5, 0, 3, 8, 10};
    int n = 7;
    cout << "array before sort:";
    for(int i = 0; i < n; i++)
    {
        cout << arr[i] << " ";
    }
    quickSort(arr, 0, n-1);
    cout << "array after sort:";
    for(int i = 0; i < n; i++)
    {
        cout << arr[i] << " ";
    }
    return 0;
}
```

Date.....

Backtracking

A backtracking algorithm is a probl-em solving algorithm that uses a Brute Force Approach for finding the desired output.

* The brute force approach tries out all the possible solutions & choose the desired / best solution.

The term backtracking suggest that if the current solution is not suitable, then backtrack & try other solutions. Thus, recursion is used in this approach.
This approach is used to solve problem that have multiple soluti-ons. In Backtracking, we create State Space Tree.

State Space Tree is a tree repres-enting all the possible states (solution or non-solution) of the problem from the root as a initial state to the leaf as a terminal state.

Date.....

Backtracking is a form of recursion.

Que: Permutation of string.

If a given string is of length 'n', then the number of permutation which is possible is "n!"

Example :-

S = "abc"

↓ possible permutations.

abc
acb
bac
bca
cab
cba

} 3!, i.e., 6 possible permutations.

Example :-

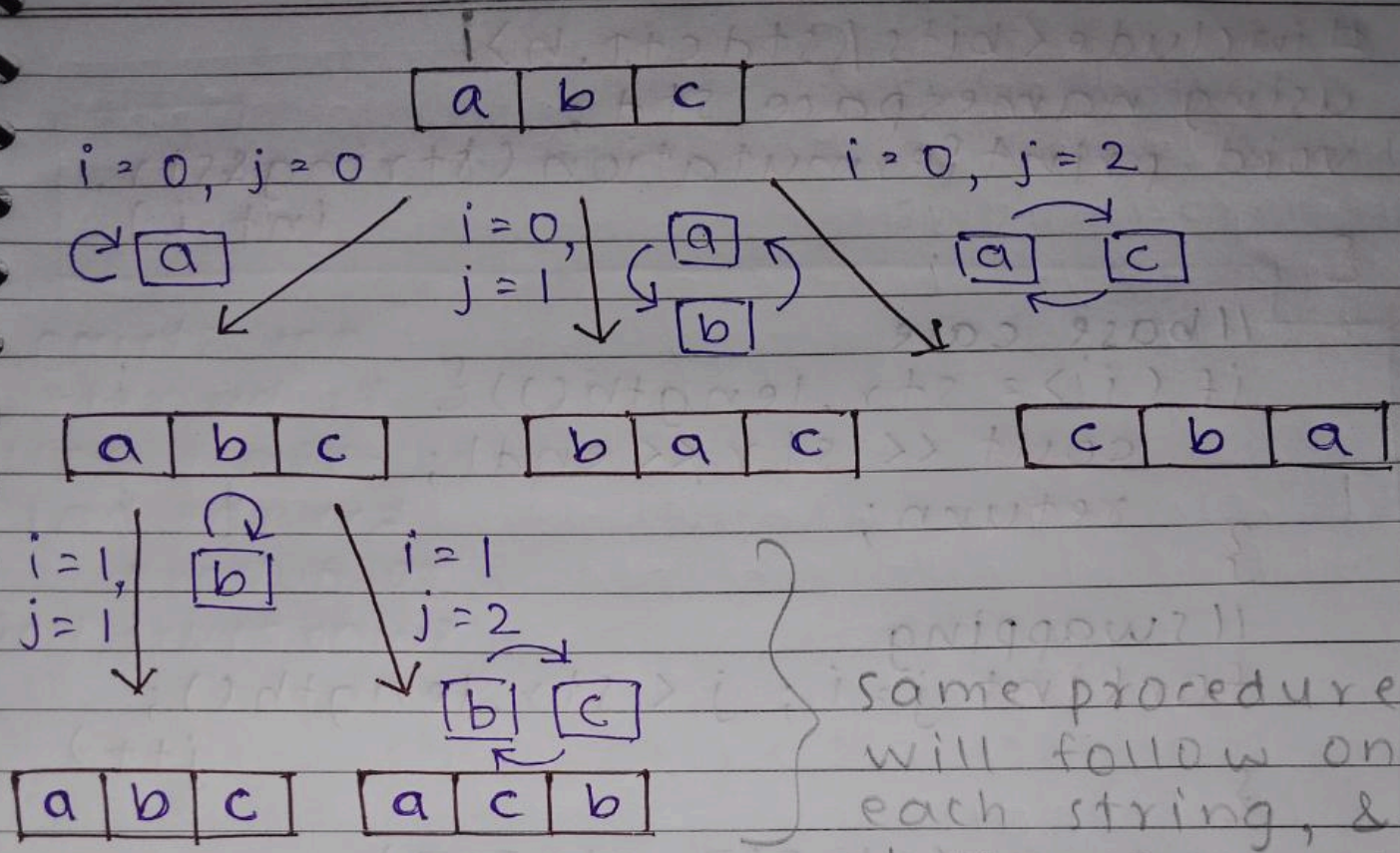
S = "abcd"

↓ possible permutations.

abcd, abdc, acbd, acdb, adcb, adbc, bacd, badc, bcda, bcad, bdac, bdca, cabd, cadb, cbad, cbda, cdab, cdba, dabc, dacb, dbac, dbca, dcab, dcba.

↳ 4!, i.e., 24 possible permutations.

Date.....



same procedure will follow on each string, & at the end we get all our permutations.

1st Iteration, i = 0, j = 0, swap i & j, then increment j, i.e., i = 0, j = 1, swap i & j, then increment j, i.e., i = 0, j = 2, swap i & j.

2nd Iteration, i = 1, j = 1, swap i & j, then increment j, i.e., i = 1, j = 2, swap i & j.

3rd Iteration, i = 2, j = 2, swap i & j.

Date.....

```
#include<bits/stdc++.h>
using namespace std;
void printPermutation(string&str,
                      int i)
{
    //base case
    if(i >= str.length()){
        cout << str << endl;
        return;
    }
    //swapping
    for(int j=i; j<str.length();
                          j++)
    {
        swap(str[i], str[j]);
        printPermutation(str, i+1);
        swap(str[i], str[j]);
    }
}
int main()
{
    string str = "abc";
    int i=0;
    printPermutation(str, i);
    return 0;
}
```

agr string ko by refe pass kr rhe hai to dubara swap zruri hai

this last step is of backtracking

Date.....

If we don't send the string by reference, then we don't need the backtracking step.

```
void printPermutation(string str,
                      int i)
{
    // base case
    if (i >= str.length()) {
        cout << str << endl;
    }
    // swapping
    for(int j=i; j<str.length(); j++)
    {
        swap(str[i], str[j]);
        printPermutation(str, i+1);
    }
}
```