Searching and Sorting

* merge Sort:→

if the given array is [10, 5, 2, 0, 7, 6, 4] How to Sort?

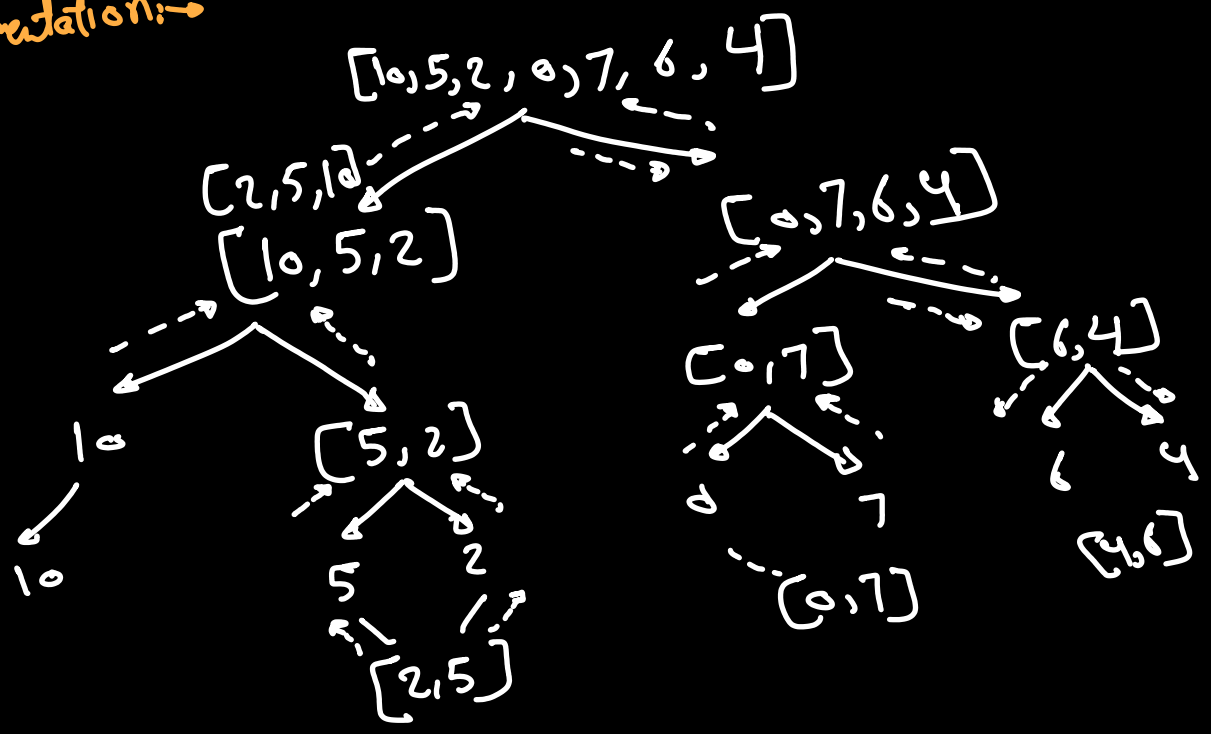
By applying Merge Sort teqnique :→

① Divide the array into 2 parts
[10, 5, 2], [0, 7, 6, 4]

② Sort each Part recursively.
[2, 5, 10], [0, 4, 6, 7]

③ Merge the two parts together, sorting them.
[0, 2, 4, 5, 6, 7, 10]

* implementation:→



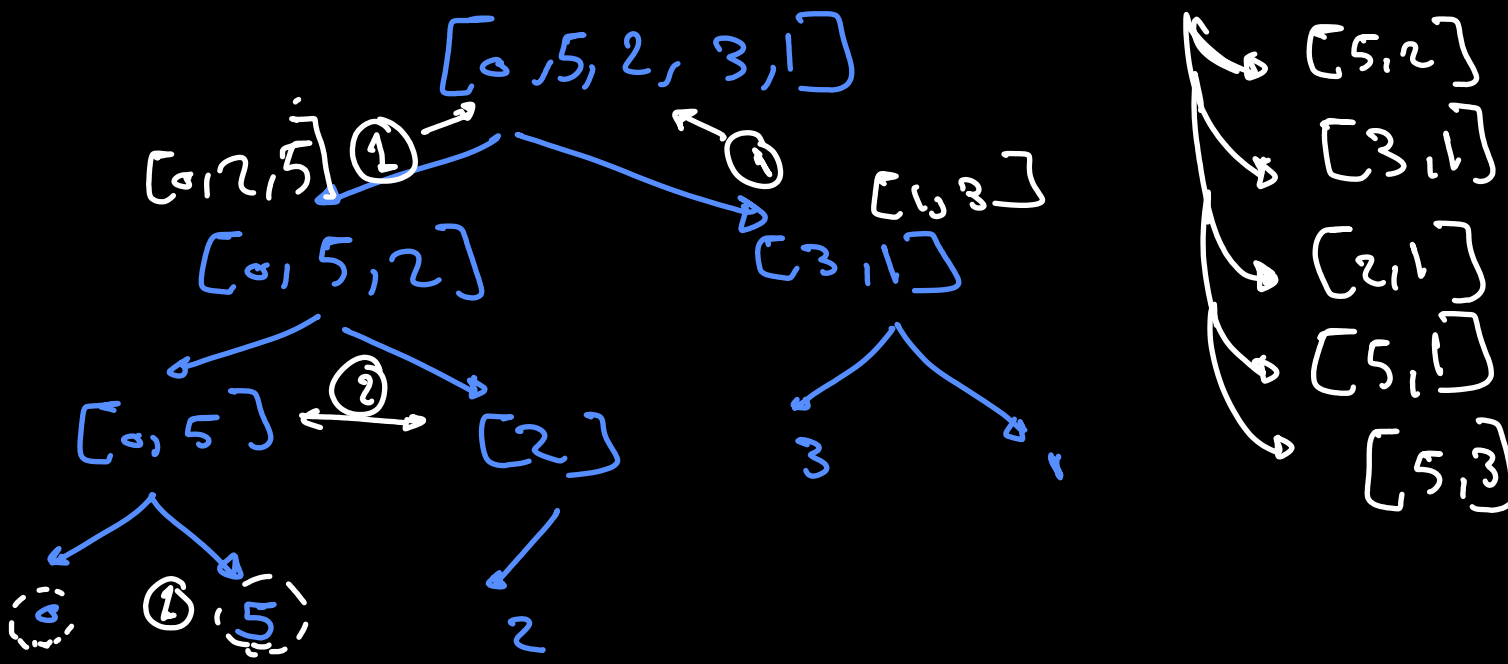

```
void mergeSort(vector<int> &nums, int start, int end) {
    if(start >= end)
      return;
    int mid = start + (end - start) / 2;
    mergeSort(nums,start, mid);
    mergeSort(nums, mid+1, end);
    performSorting(nums, start, mid, end);
}
```

* Inversion Count:→

Given arr = [0, 5, 2, 3, 1]

inversion → { arr[i] > arr[j] , i < j } Conditions

↳ using devide and conquer approach:→



to get the total numbers of inversions I calc the inversions in the left half of the array and the inversions on the Right half, then calc the Cross inversions Between L and R.

* at the first 0, 5 has Zero inversions So we Sort them and return them Back

* Then goto the Right Part which has ② only Compare it with the left half, we found then [5,2] will form an inversion.

* Sort the left and Right then return.
* at the Right half there is one inversion Between [3, 1] so, we Sort them and Return Back.
* Then we start Performing the Same approach Between the left and Right.

* let's Code up *

```
int inversionCount(vector<int> &nums, int start, int end) {
    if(start >= end)
        return 0;
    int mid = (start + end ) / 2;
    int c1 = inversionCount(nums, start, mid);
    int c2 = inversionCount(nums, mid + 1, end);
    int c3 = mergeSort(nums, start, end);
    return c1 + c2 + c3;
}
```

```
int mergeSort(vector<int> &nums, int start, int end) {
    int i = start;
    int mid = (start + end) / 2;
    int j = mid + 1;
    int cnt = 0;
    vector<int> tempArray;
    while(i <= mid && j <= end) {
        if(nums[i]  <= nums[j]) {
            tempArray.push_back(nums[i++]);
        } else {
            cnt += (mid - i + 1);
            tempArray.push_back(nums[j++]);
        }
    }

    while(i <= mid)
      tempArray.push_back(nums[i++]);
    while(j <= end)
      tempArray.push_back(nums[j++]);

    return cnt;
}
```

*Quick Sort:→

① Chose a Pivot number (Where to devide the array).

② Partition the array.

③ Recursively Sort the two Parts of the Arr.

if arr = [10, 5, 2, 0, 7, 6, 4]

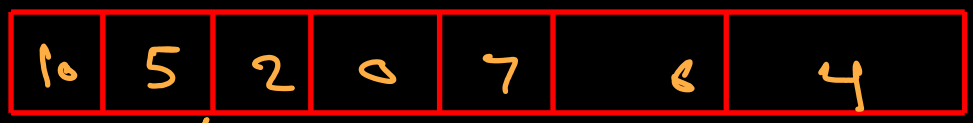
[10, 5, 2, 0, 7, 6, 4] ← Pivot elemt

i j

* First we Chose the Pivot elemt which is the last in the array.

* we put the elemds less than Pivot in the left Part of the array, and the greater on the Rig

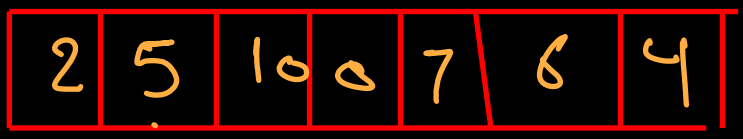
* if arr[j] > Pivot ⇒ j++



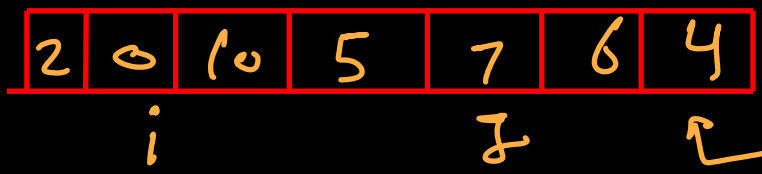
① 10 > 4 ⇒ j++

② 5 > 4 ⇒ j++

③ 2 < 4 ⇒ swap(arr[i+1], arr[j]) i++, j++



④ $a < 4 \Rightarrow$ Swap(arr[i+1], arr[J])
then Perform i++, J++



⑤ $7 > 4 \Rightarrow$ J++

⑥ $6 > 4 \Rightarrow$ J++

we Reach the Pivot

Let's Code up

```
void quickSort(vector <int> &arr, int start, int end) {
    // Base Case
    if(start >= end)
        return;
    // Recursive Case
    int p = getPartition(arr, start, end);
    quickSort(arr, start, p - 1);
    quickSort(arr, p + 1, end);
}
```

→ get the index of Pivot
→ Quick Sort on the left
→ Quick Sort on the Right

```
int getPartition(vector<int> & arr, int start, int end) {
    int pivot = arr[end];
    int i = start - 1;
    for(int j = start; j < end; j++) {
        if(arr[j] < pivot) {
            i++;
            swap(arr[i], arr[j]);
        }
    }
    swap(arr[i+1], arr[end]);
    return i + 1;
}
```

Partition technique.

* Quick Select algorithm :→

Quick Select is similar to quick sort algorithm But, we chose if we want to sort the left part or the Right part, Based on Compare the value of p with k.

```
8   void QuickSelect(vector<int> &nums, int start, int end, int k) {
9       // Base Case
0       if(start >= end)
1         return;
2       // Recursive Case
3       int p = getPartition(nums, start, end);
4       if(p < k)
5         QuickSelect(nums, p + 1, end , k);
6       else if(p > k)
7         QuickSelect(nums, start, p - 1, k);
8       else if (p == k)
9        return;
0
1
2   }
3
```

These checks will Make the Required Part only Sorted.

---

* Smallest string :→

given vector of strings like that {"a", "ab", "aba"} and I want to Concatenate them forming the Smallest lexographically vector.

Solution

a	ab	aba

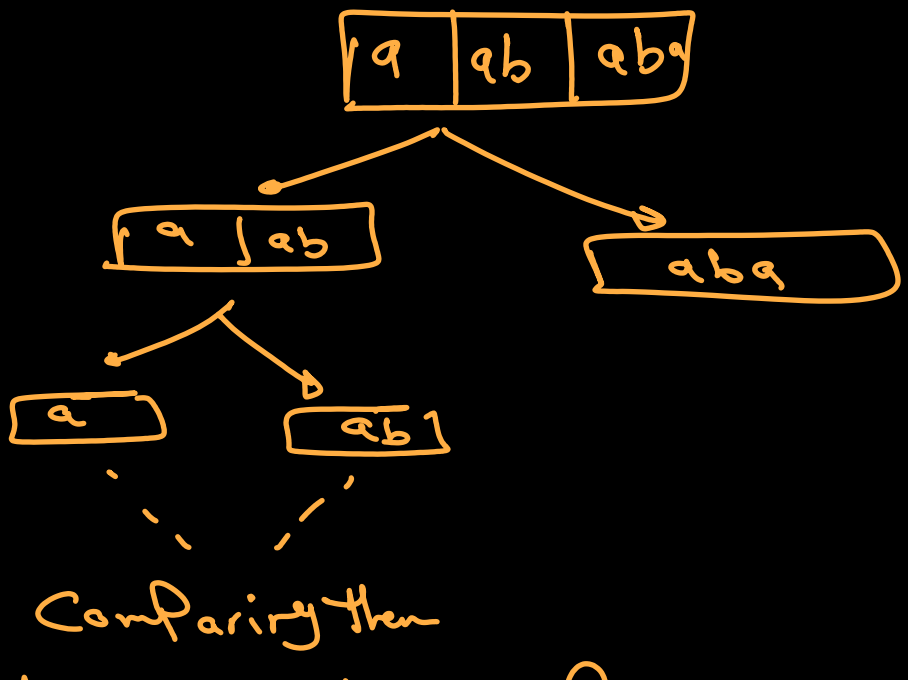
By Performing Merge Sort Based upon Some Comparators, we found that?

```cpp
bool compare(string x,string y){
    return x + y < y + x;
}


int main() {
    string arr[] = {"a","ab","aba"};
    int n = 3;

    sort(arr,arr+n,compare);

    for(auto s : arr){
        cout <<
    }
```

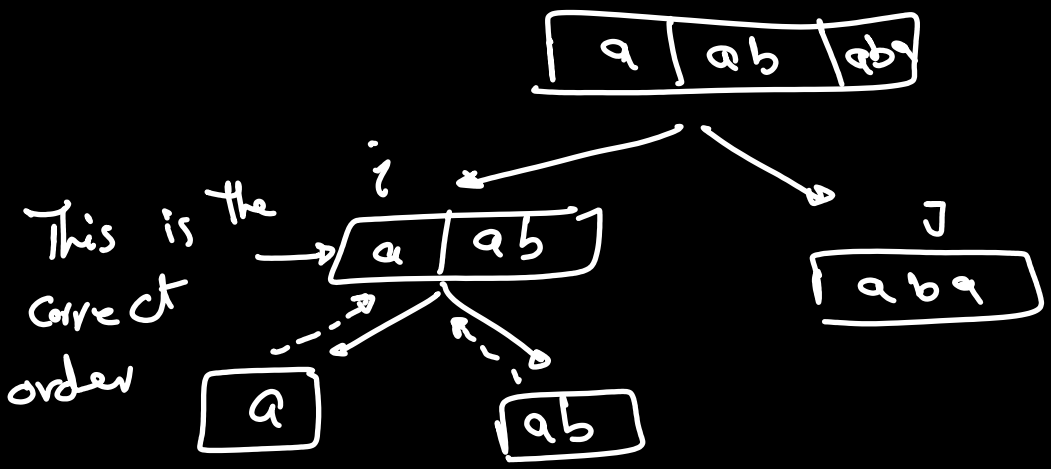


Comparing them

Based on Custom Comp

* a ab → Smaller
* aba

on the Right Part



Start Comparing arr[i] with arr[J] and merging them.

* Sparse Search :→

"ai"	" "	" "	"bat"	" "	" "	"car"	" "	" "	"dog"	" "
s										e

I want to find the index of Bat if the given array is sorted.

approach :→

s = 0

e = end

while (s <= e) {

mid = ←

if(arr[mid] == " ")

· move the mid to the nearest non-empty field

else if (arr[mi] = target) return mid;

else if (arr[mid] > target)

Search on the left

otherwise Search on the Right.

```
if(a[mid]==""){
    //update our mid to point to a nearest non empty string
    while(1){
        if(mid_left < s and mid_right > e){
            return -1;
        }
        else if(mid_right <=e and a[mid_right]!=""){
            mid = mid_right;
            break;
        }
        else if(mid_left>=s and a[mid_left]!=""){
            mid = mid_left;
            break;
        }
        mid_left--;
        mid_right++;
    }
```