* Merge sort

Merge sort is a divide - conquer sorting algorith that splits the array into smaller subarrays, sorts them, and them merges back together in sorted order,

Time complexity :- o(n log n) in all cases making it an efficient

* How merge sort works

(i) Divide :- split the array into two halves

2- conquer - Recursively Sort both halves
3- Merge - merge the sorted halves back together.

merge function

```
let combined = [];
let i = j = 0;
while (~~arr1.length~~ i < arr1.length &&
                     j < arr2.length)
{ combined.push(arr1[i]);
   i++
}else{
   combined.push (arr2 [j])
   j++
 }
}
while (i < arr1.length){
   combined.push(arr1[i]);
   i++;
 }
while (j < arr2.length){
   Combined. push (arr2[j]);
   j++;
 }
return Combined
}
```

function of mergeSort(arr)

```
if (arr.length === 1) return arr;
 let ~~left~~ mid = Math.floor(arr.length/2);
 let left = arr.slice(0, mid);
 let right = arr.slice (mid);
 return merge(mergeSort(left), mergeSort
                          (right)
}
```

☆ Big O
→ Time complexity ≡O (n log n)
- Best case :- O(n log n)
- Worst Case : O(n log n)
- Average Case : O(n log n)

☆ Quick Sort - A Fast Divide - and conquer Sorting Algorithm.
→ Quick sort is divide and conquer algorithm that picks a pivot, partitions that array into halves, and recursively sort them. It is known for its efficiency in ~~pra~~ practical use

☆ Swap function

```
function swap (arr, firstInd, secInd){
  let temp = arr [firstInd];
  arr[firstInd] = arr[secondInd];
  arr[secondInd] = ~~arr~~ temp;
```

★ Pivot function

```
function Pivot(arr, pivotInd=0, endInt
                                = arr.length-1){
  let swapInd= pivotInd;
  let pivot Valu = arr[pivotInd];
for(i=pivoitInd+1; i <=endInd; i++){
    if (arr[i]< pivotValue){
       swapInd ++;
       swap(arr, swapInd, i)
    }
  }
  swap(arr, pivotInd, swapInd);
  return SwapInd;
}
```

★ Quick Sort function

```
function quickSort(arr, left=0,
                   right=arr.length-1){
  if (left <right){
    let Pivot Ind = pivot(arr, left, right);
    quickSort(arr, left, pivioInd-1);
    quick Sort(arr, pivotInd+1, right)
  }
  return arr;
```

★ Big O

→ Time complexity :-

- Base case - $O(n \log n)$
- Averg case - $O(n \log n)$
- worst - $O(n^2)$

→ Space complexity - $O(\log n)$

★ Tree Traversal order

→ It is the process of visiting all the nodes of a tree data structure in a systematic way.

→ There three main types of tree traversal:-

1- Depth - First Traversal (DFT)

In DFT,

a- Inorder Traversal (Left → Root → Right)

- First, visit the left subtree.
- Then, visit the root node.
- Finally, visit the right subtree.

→ used (Binary Search Tree) BST to retrieve sorted values.

b- Preorder Traversal (Root → Left → Right)

- First, visit the root node.
- Then, visit the left subtree.
- Finally, visit the right subtree.
- used for copying trees & prefix

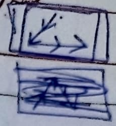
C- Postorder Traversal (Left -> Right -> Root)

- First, visit the left subtree.
- Then, visit the right subtree.
- Finally, visit the root node.
- used for deleting nodes and postfix expression

2- Breadth-First Traversal (BFT) / Level Order Traversal

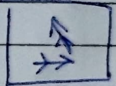
- visit all nodes at each level moving to the next level.
- uses a queue (FIFO) nodes.

```
BFS(){
   currentNode = this.root;
   let queue = [];
   let res   = [];
   ~~while (queue.length){~~
    ~~currentNode=~~
   queue.push(currentNode);
    while (queue.length){
    currentNode = queue.shift();
     res.push(currentNode.value);
    if (currentNode.left) queue.push(currentNode.left);
   if (currentNode.right) queue.push(currentNode.right);
  }
  return res;
}
```



☆ DFS - Preorder (RLR)

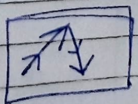
```
DFSPreorder(){
   let res = [];
   function traverse(currNode){
       res.push(currNode.value);
       if (currNode.left) traverse(currNode.left)
      if (currNode.right) traverse(currNode.right)
   }
   traverse(this.root);
  return root;
}
```



☆ DFS-Post Order (LRR)

```
DfsPostOrder(){
      let res = [];
      function traverse (currNode){
       if (currNode.left) traverse(currNode.left);
      if (currNode.right) traverse (currNode.right);
      res.push(currNode.value);
    }
    traverse(this.root);
    return res;
}
```

☆ ~~In~~ DFS - InOrder (LRR)



```
DfsInOrder(){
  let res = [];
  function traverse (currNode){
  if (currNode.left) traverse(currNode.left
  ~~if (currNode.right) traverse (currNode.right~~
  res.push(currNode.value);
  if (currNode.right) traverse(currNode.right
 }
 ~~return~~ traverse (this.root);
 return res;
}
```