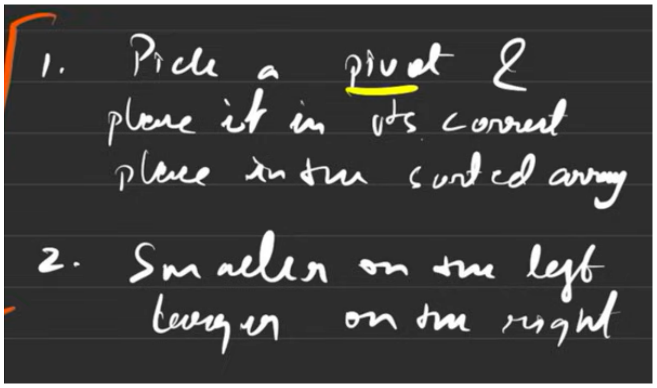
Sorting :

Merge sort

- Divide and conquer type sorting

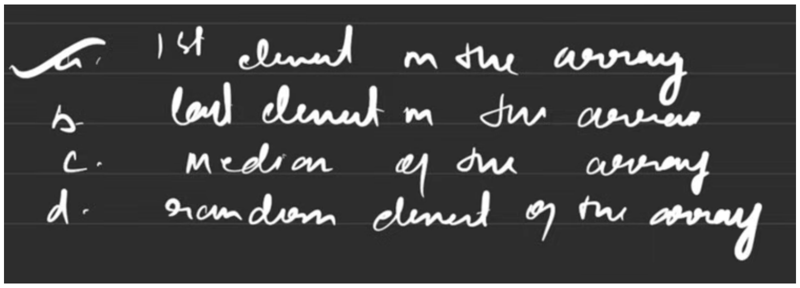
```
void merge(vector<int> &arr, int low, int mid, int high) {
    vector<int> temp; // temporary array
    int left = low;      // starting index of left half of arr
    int right = mid + 1;   // starting index of right half of arr

    //storing elements in the temporary array in a sorted manner//

    while (left <= mid && right <= high) {
        if (arr[left] <= arr[right]) {
            temp.push_back(arr[left]);
            left++;
        }
        else {
            temp.push_back(arr[right]);
            right++;
        }
    }

    // if elements on the left half are still left //

    while (left <= mid) {
        temp.push_back(arr[left]);
        left++;
    }

    //  if elements on the right half are still left //
    while (right <= high) {
        temp.push_back(arr[right]);
        right++;
    }

    // transfering all elements from temporary to arr //
    for (int i = low; i <= high; i++) {
        arr[i] = temp[i - low];
    }
}

void mergeSort(vector<int> &arr, int low, int high) {
    if (low >= high) return; // {> never happens == is enough}
    int mid = (low + high) / 2 ;
    mergeSort(arr, low, mid);  // left half
    mergeSort(arr, mid + 1, high); // right half
    merge(arr, low, mid, high);  // merging sorted halves
}
```

Quick Sort :



1. Pick a pivot & place it in its correct place in the sorted array

2. Smaller on the left larger on the right



a. 1st element in the array
b. last element in the array
c. Median of the array
d. random element of the array

```
int partition(vector<int> &arr, int low, int high) {
    int pivot = arr[low]; // taking case a
    // {The result will be same for anyother}
    int i = low;
    int j = high;

    while (i < j) {
        while (arr[i] <= pivot && i <= high - 1) {
        // we need to find the first element which is bigger than pivot
            i++; // if smaller keep looking ahead
        }

        while (arr[j] > pivot && j >= low + 1) {
        // we need to find the first element which is smaller than pivot
            j--;
        }
        if (i < j) swap(arr[i], arr[j]);
    }
    swap(arr[low], arr[j]);
    // Now excluding low the starting elements till j will be
    // less than pivot and the rest will be greater than pivot
    return j; // This is now for sure in its correct place !
}

void qs(vector<int> &arr, int low, int high) {
    if (low < high) {
        int pIndex = partition(arr, low, high);
        // Divide and Conquer
        qs(arr, low, pIndex - 1);
        qs(arr, pIndex + 1, high);
    }
}

vector<int> quickSort(vector<int> arr) {
    qs(arr, 0, arr.size() - 1);
    // along with the array we pass in two pointers low and high
    return arr;
}
```