Sorting

(1) Bubble Sort:

Question:

12	23	35	15	11

1st Iteration:

12	23	35	35	11

12	23	15	11	35

2nd Iteration:

12	23	15	11	35

12	15	23	11	35

12	15	11	23	35

3rd Iteration:

12	15	11	23	35

12	11	15	23	35

4th Iteration:

12	11	15	23	35

11	12	15	23	35

```
for (a=0; a<n-1; a++)
    for (i=0; i<n-1; i++)
    {
        if (arr[i] > arr[i+1])
        {
            temp = arr[i]
            arr[i] = arr[i+1]
            arr[i+1] = temp
        }
```

Algorithm:

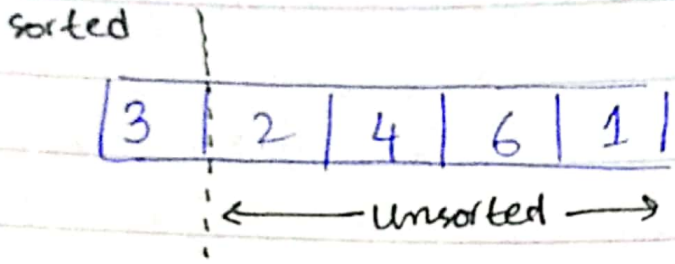
Step 01: Start the program.
Step 02: Initialize an unsorted list.
Step 03: Using for loop from 0 to no. of element - 1. (Outer loop)
Step 04: Apply inner for loop from 0 till n-1.
Step 05: Using if to check if the first number is greater than the second or not.

Step 06: If condition gets false swap those numbers.

Step 07: End the program.

Jenny's Lecture

(2) Insertion Sort:

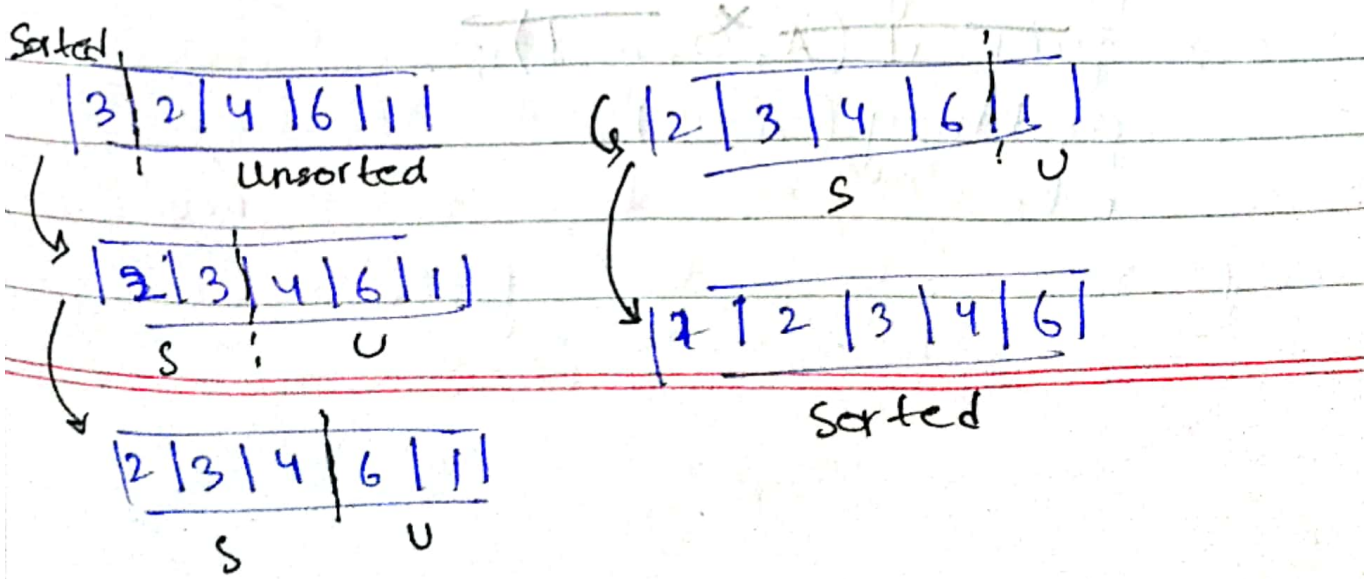


Code:

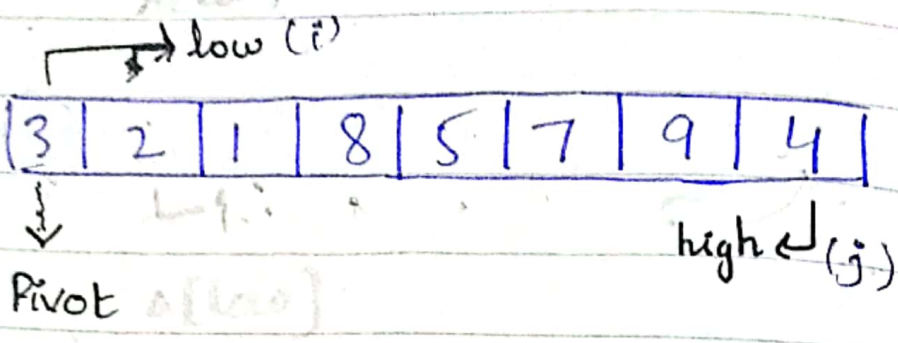
```
for (int i= 1 ; i<=4 ; i++)
{
    temp = arr[i];
    j = i - 1;
    while ( j>=0 && arr[j] > temp)
    {
        arr[j+1] = arr[j];
        j--;
    }
    arr[j+1] = temp;
}
```

Algorithm:

Step 01: Start the program.
Step 02: Initialize an array.
Step 03: Start a for loop from 1 till length of array - 1.
Step 04: Assign value of i in temporary variable.
Step 05: Initialize j as one less i.
Step 06: Apply a while loop with two conditions:
j >= 0
and
arr[j] > temp.
Step 07: assign value of arr[j] into arr[j+1].
Step 08: Decrement j by 1.
Step 09: assign value of temp into arr[j+1].



(3) Quick Sort :



=> Take first num as Pivot.
=> i will be incremented and j will be decremented.
=> i will ~~find~~ stop at pivot se bda num
j will stop at pivot se chota num.
=> agr i or j ne ek dusre ko cross kiya hoga to dono ki values swap else j and pivot swap.

Code.

```
~~void~~ int main()
{
    int A[] = {9,4,4,8,7,5,6};
    int n = 9;
    printArray(A, n);
*   quickSort(A, 0, n-1);
    printArray(A, n);
    return 0;
}
```

```
void quicksort(int A[], int low, int high)
{
    int partitionIndex;

    if (low < high)
    {
        partitionIndex = partition(A, low, high);
        quicksort(A, low, partitionIndex - 1);
        quicksort(A, partitionIndex + 1, high);
    }
}

int partition(int A[], int low, int high)
{
    int pivot = A[low];
    int i = low + 1;
    int j = high;
    int temp;

    do
    {
        while (A[i] <= pivot)
        {
            i++;
        }
        while (A[j] > pivot)
        {
            j--;
        }
```

```
        if (i<j)
        {
            temp = A[i];
            A[i] = A[j];
            A[j] = temp;
        }
    }
    while(i<j);

    temp = A[low];
    A[low] = A[j];
    A[j] = temp;
    return j;
}
```

Algorithm:

Main Function

Step 01: Start the program.
Step 02: Initialize and assay as well as its length(n).
Step 03: Call the function quickSort(A,0,n);
Step 04: Print that assay.
Step 05: End the program.

quicksort (arr, low, high) function.

Step 01: Initialize low = first index element (0th index) of array, it is also called Pivot.
Step 02: Initialize high as last index of arr.
Step 03: If low < high, call the partition(arr, low, high) function.
Step 04: Using recursiveness properly, again call the quicksort(arr, low, p.index-1) but with diff. arguments.
Step 05: PartitionIndex is basically the variable which consist the value of Pivot's position.
Step 06: Again call quicksort(arr, pindex+1, high)
Step 07: End this function.

partition (arr, low, high) function.

Step 01: Initialize pivot = arr[low].
i = low + 1.
j = high.
Step 02: Applying do while loop with the condition (i < j).
Step 03: Inside loop apply while (arr[i] <= pivot)

```
{
    i++;
}
```

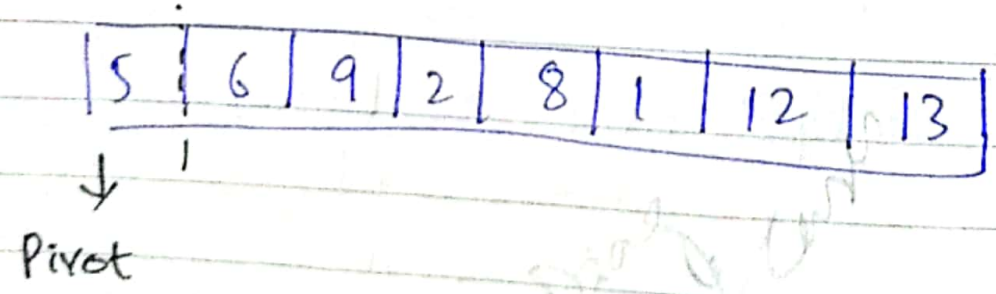
```
while (arr[j] > pivot)
{
    j--;
}
```

Applying if condition:

```
if (i<j)
{
    swap arr[i] with arr[j]
}
```

Step 04: If loop condition gets false, swap arr[j] by pivot (arr[low]).

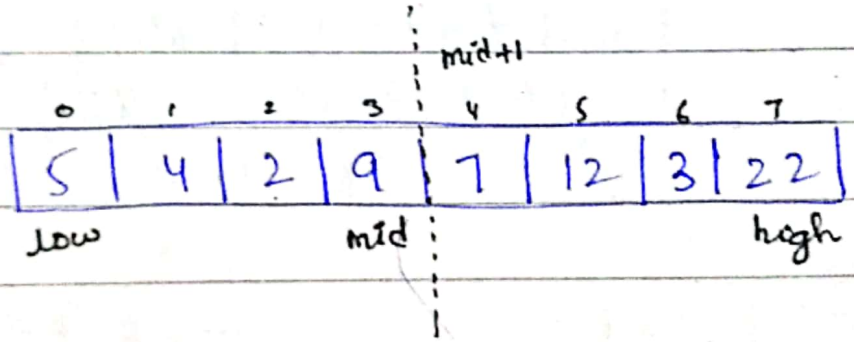
Step 05: Return j.



Merge Sort:

> First we will break an array into two equal parts. We consider first element as low and last as high. The breakpoint will be mid. We do this again and again by recursiveness and when only one element is left we will apply merge() function.

In merge(), we have two parts low → mid and mid+1 → high, we will apply ~~three~~ two diff. loops on each and start comparison and store the values in new array B[]



Algorithm:

Recursive Merge Sort Function.

~~void MS (int A[]).~~

Step 01: Start the program

Step 02: Apply If to check low < high.

Sub-step (2a): Find mid point by using formula mid = $\frac{low+high}{2}$.

Sub-step (2b): Again call this function from low to mid.

Sub-step (2c): Call this function agaian from mid+1 to high.

Sub-step (2d): Call merge() function.

Step 03: End the function.

Pseudocode:

```
void MS (int A[], int low, int high)
{
    if (low < high)
    {
        mid = (low+high)/2
        MS (A, low, mid);
        MS (A, mid+1, high);
        Merge (A, low, mid, high);
    }
}
```

Algorithm for merge() function.

Step 01: Start the function.

Step 02: Initialize i = low, j = mid +1 and k = low.

Step 03: Apply while loop with the condition (i < mid && j < high)

Sub-step(3a): Check if (A[i] < A[j])

Sub-step-3a(i): Copy A[i] in new array B[k] and increment i and k.

Sub-step (3b): Else copy A[j] in B[K] and increment j and K by one.

Step 04: Now for copying remaining elements, ~~from A to C~~, low to mid apply while loop with condition (i ≤ mid)

Sub-step(4a): B[K] = A[i] and i++, K++

Step 05: Copying from ~~B~~ mid+1 to ~~C~~ high, while (j < high)

Sub-step(5a): B[K] = A[j] and K++ & j++;

Step 06: End the function.

Pseudocode:

```
void merge (A[], mid, low, high) {
    int i=low, j= mid+1 , K = low;
    while (i<mid && j< high)
    {
        If A[i] < A[j]) {
            B [K] = A[i]
            i++ ; K++ ;
        }
        else
        {
            B[K] = A[j];
            j++ ; K++ ;
        }
    }
    while (i ≤ mid)
    {
        B[K] = A[i];
        K++; i++;
    }
    while (j < high) {
        B[K] = A[j];
        K++ ; j++ ;
    }
```

Diagram of Merge Sort