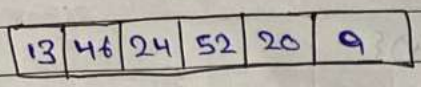
Page No.
Date

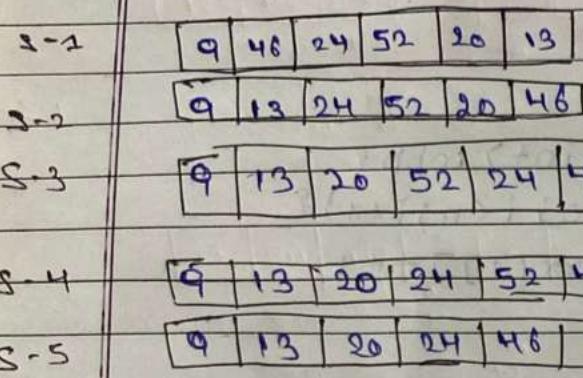
STEP-2

Sorting

⊗ Selection Sort



Steps
1) Select min. and swap it with first element
2) Now check for min. and again swap it
3) Do the same until it sorted completely



observation: ↑ (0 to n-1)
1) Swap at index 0 & min. no. index
2) swap at index 1 & min. no. index ↑ (1 to n-1)
3) Swap at index 2 & Min. no. index (2 to n-1)

Swap at Index (n-2) & (n-1)

Code:

```
void Selection_Sort(int arr[], int n){
    for(int i=0; i<=n-2; i++){
        int mini = i;          // declaring first element as min.
        for(int j=i; j<=n-1; j++){
            if(arr[j] < arr[mini]){
                mini = j;
            }
        }
    }
    int temp = arr[mini];
    arr[mini] = arr[i];
    arr[i] = temp;
}
}
```

finding and swapping element

logic for swapping

```
int main () {
    int n;
    cin >> n;
    int arr[n];
    for (int i = 0; i < n; i++) {
        cin >> arr[i];
    }
    Selection_Sort (arr, n);
    for (int i = 0; i < n; i++) {
        cout << arr[i] << " ";
    }
    return 0;
}
```

Time complexity for selection Sort

1st time ⇝ loop runs for n time

2nd time ⇝ loop runs for n-1 times

3rd time ⇝ loop runs for n-2 times

$$n + n-1 + n-2 + \cdots + 2 + 1 \leq \frac{n(n+1)}{2}$$

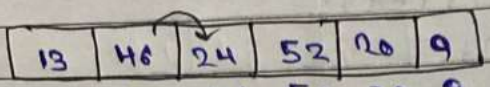
$$T.C \leq O(n^2)$$

Page No.
Date

⊗ Bubble Sort

pushes the maxm to the last by adjacent swaps

Steps:
1) 2 element ko ek sath pkdo and compare krke swap krdo and then move to next.
After iterating completely for first time, The biggest number should be at the end.
2) ~~dubara se same repeat~~ kro. har step k sath last element sorted hote jayenge

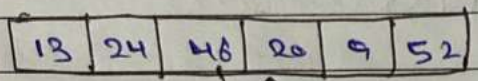


24 46 52 20 9
24 46 20 52 9
13 24 46 20 9 52

last element sorted

S-1

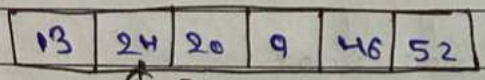
S-2



13 24 20 46 9 52
13 24 20 9 46 52

last 2 element sorted

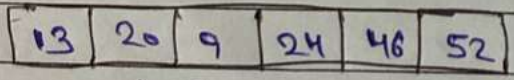
S-3



20 24 9
13 20 9 24 46 52

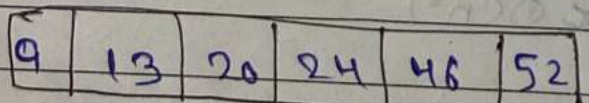
last 3 element sorted

S-4



9 20 24 46 52

S-5



Sorting done

Observation:
1) 1st time ⇝ loop runs from 0 to n-1
2) 2nd time ⇝ loop runs from 0 to n-2
3) 3rd time ⇝ loop runs from 0 to n-3

Page No.
Date

code:

```
void bubble_sort(int arr[], int n){
    for(int i=n-1; i>=0; i--){
        for(int j=0; j<=i-1; j++){
            if(arr[j] > arr[j+1];        ~> logic to find max size element
                int temp = arr[j+1];
                arr[j+1] = arr[j];       } swapping logic
                arr[j] = temp;
            }
        }
    }
}
```

Time complexity = $O(n^2)$ ~> worst case / avg. case

for best case ~> $O(n)$

⑧ Insertion Sort

har step mai ek element ko leke jayega and correct order mai lagaega

1st time ~> array with 1 element
2nd time ~> array with 2 element
3rd time ~> array with 3 elem

14	9	15	12	6	8	13

har step mai assumed array mai ek element ko add karte rho (size increase)

S-1 14 9 15 12 6 8 13

S-2 9 14 15 12 6 8 13

S-3 9 14 15 12 6 8 13

S-4 9 12 14 15 6 8 13

S-5 6 9 12 14 15 8 13

S-6 6 8 9 12 14 15 13

S-7 6 8 9 12 13 14 15

Page No.
Date

Code:

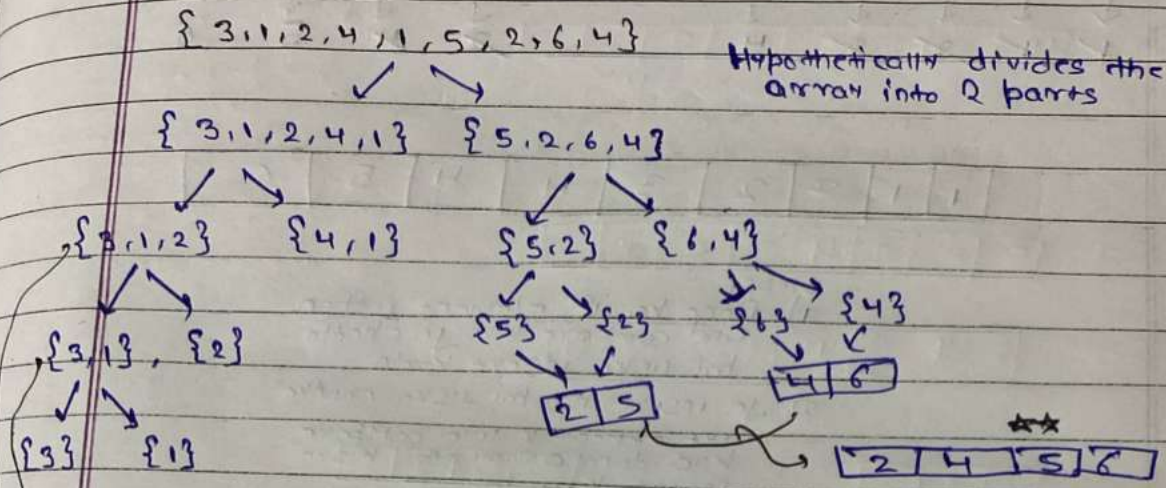
```
void insertion_sort(int arr[], int n){
    for(int i=0; i<=n-1; i++){
        int j=i;
        while(j>0 && arr[j-1] > arr[j]){
            int temp = arr[j-1];
            arr[j-1] = arr[j];
            arr[j] = temp;
            j--;
        }
    }
}
```

Time complexity: $O(n^2)$ ⇝ worst case / average case

for best case ⇝ $O(n)$

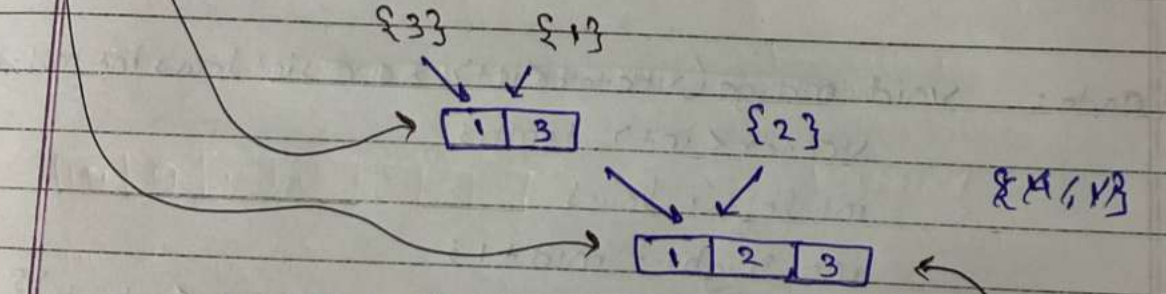
Date

⊗ Merge sort ⇝ Divide & Merge



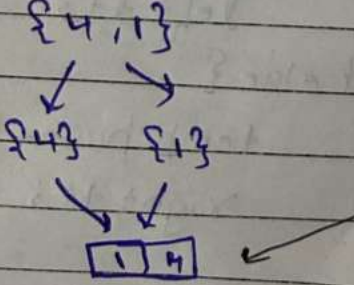
till here divide is done.
now merge

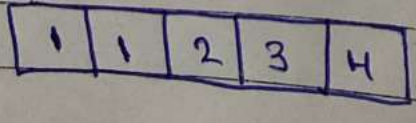
merge but do it in sorted way



Now the left part of divided array is sorted. now go to right one

both will comeback and merged themselves





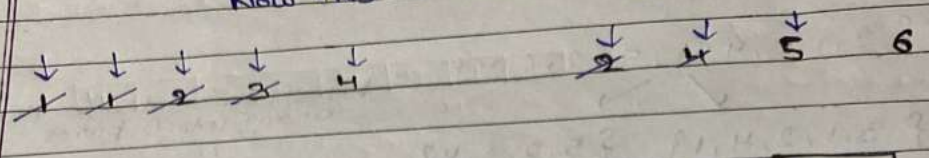
← Result of Merge Sort (left part)

Now the left and right both array sorted

left → 1 1 2 3 4
right → 2 4 5 6

Page No. 14
Date

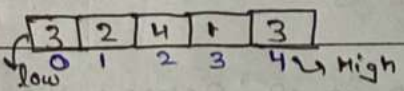
Now this 2 will merge in sorted



1	1	2	2	3	4	4	5	6

Steps:
1) Dono ka 1st element uthao and compare kro jo chotta hai usko insert krdo
2) Jo insert hua hai usko chodke next element k live compare kro dono array mai kisko insert krna hai

Ex:



Code:

```
void merge(vector<int> &arr, int low, int mid, int high){
    vector<int> temp;
    int left = low;
    int right = mid+1;
    while(left <= mid && right <= high){
        if(arr[left] <= arr[right]){
            temp.push_back(arr[left]);
            left++;
        }else{
            temp.push_back(arr[right]);
            right++;
        }
    }
```

```
    while (left <= mid){
        temp.push_back(arr[left++]);
        left++;
    }
    while (right <= high){
        temp.push_back(arr[right++]);
    }
    for(int i = low; i <= high; i++){
        arr[i] = temp[i-low];
    }
}

void ms(vector<int> &arr, int low, int high){
    if(low == high) return;
    int mid = (low + high) / 2;
    ms(arr, low, mid);
    ms(arr, mid+1, high);
    merge(arr, low, mid, high);
}

void mergesort(vector<int> &arr, int n){
    ms(arr, 0, n-1);
}
```

Time complexity : $O(n \log_2 n)$ → bez it is dividing into 2

Merge, divide

Space complexity : $O(n)$

Date

⊗ Quick Sort ⇝ T.C = $O(n \log_2 n)$
S.C = $O(1)$

Steps:
1) Pick a pivot & place it in its correct place in the sorted array
Pivot can be
⇝ 1st element
⇝ last element
⇝ median element
⇝ Random element
2) Smaller on the left & larger on the right w.r.t pivot
3) repeat these steps

Ex: | 4 | 6 | 2 | 5 | 7 | 9 | 1 | 3 |

let pivot = 4
now place it according to when array is sorted

S-1 | 5 | 6 | 2 | 4 | 7 | 9 | 1 | 3 |

S-2 | 2 | 1 | 3 | 4 | 6 | 5 | 7 | 9 |

| 2 | 1 | 3 |

let pivot = 2

Correct order = | 1 | 2 | 3 |

And also Step 2 done

| 6 | 5 | 7 | 9 |

let pivot = 6

| 5 | 6 | 7 | 9 |

⇝ when i placed correct order of pivot. it auto. done step 2 and it is sorted

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

Note: ⇝ We haven't made new array here. We just use concept of low, high from merge sort.

Page No.
Date

code:

```
int partition(vector<int> &arr, int low, int high){
    int pivot = arr[low];
    int i = low;
    int j = high;
    while(i < j){
        while(arr[i] <= pivot && i <= high-1){
            i++;
        }
        while(arr[j] > pivot && j >= low+1){
            j--;
        }
        if (i<j) swap(arr[i], arr[j]);
    }
    swap(arr[low], arr[j]);
    return j;
}

void qS(vector<int> &arr, int low, int high){
    if (low < high){
        int pIndex = partition(arr, low, high);
        qS(arr, low, pIndex-1);
        qS(arr, low, pIndex+1, high);
    }
}

vector<int> quickSort(vector<int> arr){
    qS(arr, 0, arr.size()-1);
    return arr;
}
```