Date.
Page No.

ASSIGNMENT - 02

1)
```
int linearSearch( int arr[], int n, int key){
    for( int i=0 ; i<n ; i++){
        if (arr[i] == key) return 1;
        else if ( arr[i] > key)
            return -1;
        else continue;
    }
    return -1;
}
```

2) Iterative Insertion Sort

~~void function{~~

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

Recursive Insertion Sort

```
void insertion ( int arr[], int n){
    if n<=1
        return;
```

Date.
Page No.

```
insertion(arr, n-1);
int last = arr[n-1];
int j = n-2;
while(j>=0 && arr[j]>last){
        arr[j+1] = arr[j];
}       j--;
arr[j+1] = last;
}
```

Insertion sort is ~~~~ as online sort because It can process the inputs in a serial fashion i.e, in the order that the input is fed to the algorith without having the entire input available from the begining.

Examples → Insertion, Heap, Quick, Bubble, Selection.

3.)

	Sorting	Time complexity			Space complexity
		Best	Avg	Worst	
1.	Bubble	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$
2.	Selection	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$
3.	Insertion	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$
4.	Quick	$O(n\log n)$	$O(n\log n)$	$O(n^2)$	$O(n)$
5.	Merge	$O(n\log n)$	$O(n\log n)$	$O(n\log n)$	$O(n)$
6.	Heap	$O(n\log n)$	$O(n\log n)$	$O(n\log n)$	$O(n)$
7.	Count	$O(n+k)$	$O(n+k)$	$O(n+k)$	$O(n+k)$
8.	Radix	$O(d*(n+k))$	$O(d*(n+k))$	$O(d*(n+k))$	$O(n+k)$

Date.
Page No.

4)

Sorting	In Place	Stable	On-line
Bubble	✓	✓	x
Selection	✓	x	x
Insertion	✓	✓	✓
Quick	✓	x	x
Merge	x	✓	x
Heap	✓	x	x
Count	x	✓	x
Radix	x	✓	x

5) Recursive Binary Search

```
bool binarySearch(int arr[], int l, int r, int key){
    if(l>r)
        return false;
    int mid = l+(r-l)/2;
    if(arr[mid]==key)
        return true;
    else if(arr[mid]>key){
        binarySearch(arr, l, mid-1, key);
    binarySearch(arr, mid+1, r, key);
}
```

Iterative Binary Search

```
int BinarySearch(int arr[], int n, int key){
    int r=n;
    int l=0;
    while(l<=r){
        mid=l+(r-l)/2;
```

Date.
Page No.

```
if (arr[mid] == key)
        return true;
else if (arr[mid] > key)
        r = mid - 1;
else
        l = mid + 1;
return -1;
```

Sorting	Recursive		Iterative	
	T.C	S.C	T.C	S.C
Linear Search	O(n)	O(1)	O(n)	O(1)
Binary Search	O(logn)	O(logn)	O(logn)	O(1)

e) Recursive Binary Search

```
bool binarySearch(int arr[], int l, int r, int key){
    if(l > r) return false;
    int mid = l + (r-l)/2;          ——→ O(1)
    if (arr[mid] == key)
        return true;
    else if (arr[mid] < key)        ——T(n/2)
        binarySearch(arr, mid+1, r, key);
    binarySearch(arr, l, mid-1, key);   —T(n/2)
}
```

$\therefore T(n) = T(n/2) + T(n/2) + O(1)$

$$\boxed{T(n) = T(n/2) + O(1)}$$

Date.
Page No.

7) void search (int a[], int n, int k) {
    msort(A, n) // call for merge sort
    int i=0;
    int j=n-1;
    while (i<j) {
        if (a[i]+a[j]==k)
            cout << i << j ; return;
        else if (a[i]+a[j] < k)
            i++;
        else
            j++;
    }
    cout << " Not found ";
}

8) We can't say or assume any particular algorithm as best algorithm because each algorithm has its own pros and cons. We must consider sorting as per use. For example:-

(i) Merge Sort → can be used where stability is important and has sample amount of free memory. No suitable for large database.

(ii) Quick Sort → It can be used where data is uniformly distributed and more amphasis on time. Use less memory and can be used in database.

(iii) Heap sort → No additional memory is required but is little slower than above two. Can be used for databases.

Date.
Page No.

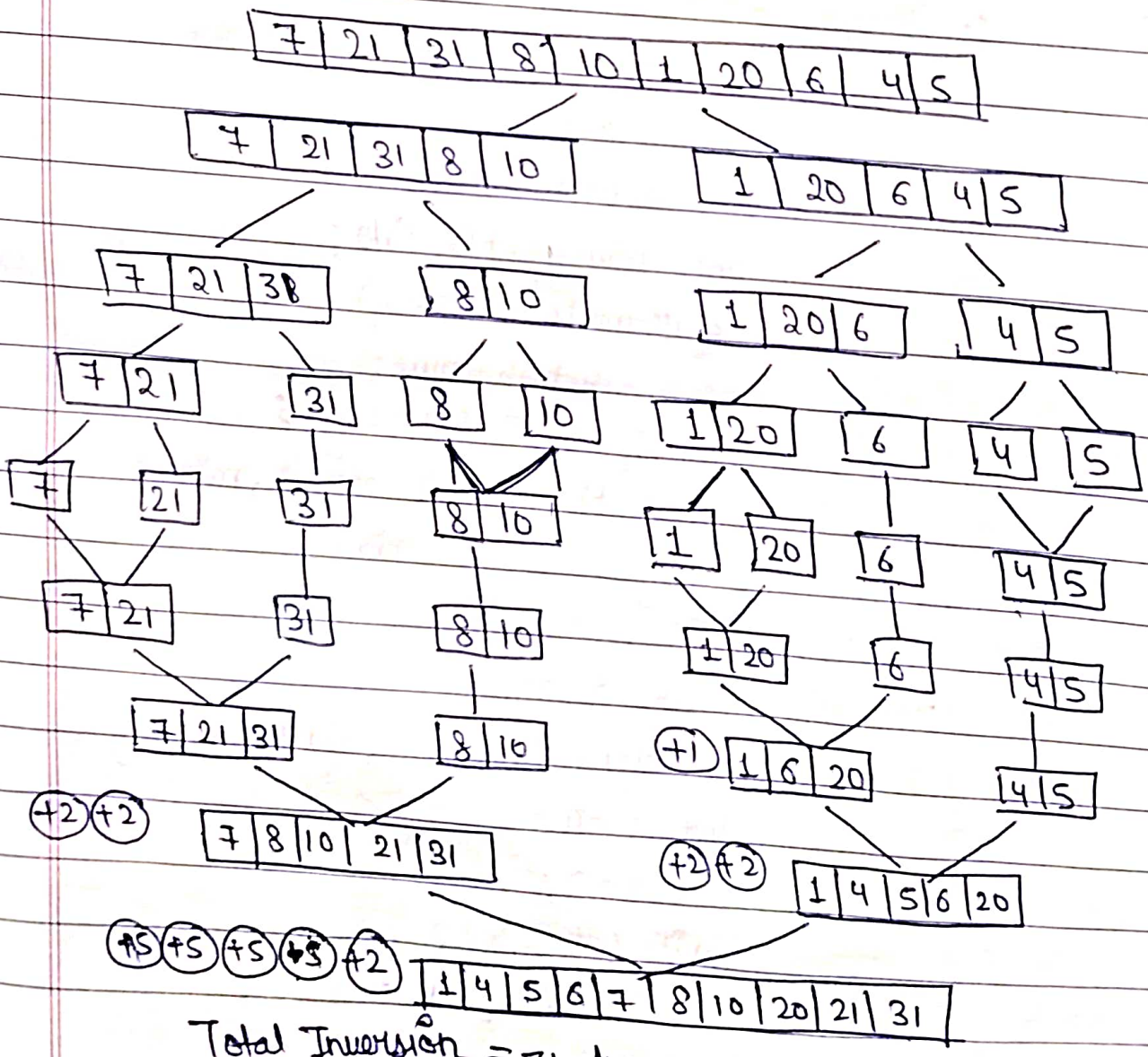
(iv) Insertion sort → used in smaller database or questions where some part needs to be sorted.

(v) Selection sort → T.C being heigh in partically not used as much but can be implied in smaller sits.

(vi) Bubble sort → Rarely used.

9) Inversions → Number of inversions means how many elements is shifted before placing new elements its prescribed location.

Merge sort →



Total Inversion = 31 Ans

Date.
Page No.

10) Quick Sort

Best = O(nlogn) when pivot is exactly the middle element.

Worst = $O(n^2)$ when pivot is either first or last element.

Average = O(nlogn) Array is roughly divided in equally half subarray.

11) Merge Sort.

Best & Worst :- $T(n) = 2T(n/2) + O(n)$

Quick Sort

Best :- $T(n) = T(n/2) + T(n/2) + O(n)$

Worst :- $T(n) = T(n-1) + O(n)$

Similarities :-

① both uses divide and conquere approach.

② TC = O(nlogn)

Difference :-

In Quick Sort, T.C varies.

12)

```
void Selection sort (int a[], int n) {
    for (int i=0; i<n; i++) {
        int min = i;
        for (int j=i+1; j<n; j++)
        {
            if a[j] < a[min]
                min = j;
        }
        minv = a[min];
```

Date.
Page No.

```
        while(min > i){
            a[min] = a[min - 1];
            min = min - 1;
        }
        a[i] = minv;
    }
}
```

13) void bubblesort(int a[], int n){
    bool k;
    for(int i=0; i<n-1; i++)
    {   k = false;
        for(int j=0; j<n-1; j++){
            if(a[j] > a[j+1]){
                swap(a[j], a[j+1]);
                k = true;
            }
        }
        if(!k){ break; }
    }
}

14) Merging of data greater than available memory
We can sort data of 4GB in 2GB memory to do so we have to use external sorting methods like External merge sort or External Quick sort.

External Sorting :- It is sorting in which data is sorted and stored in slower memory for a while. Then it is retrived from file to sort with other data. This chuncking of data allows us to sort large data with less resource usage.

Date.
Page No.

Internal Sorting:- It is sorting in which memory is directly used and no secondary memory is used. It can only accomodate chunck smaller than available memory.

Speed comparison:- Internal > External

Storage can be sorted:- External > Internal