Page :
Date: / /

Ans1- int L_search (int *ar , int n, int key)

```
{
    for (int i =0 ; i<n ; i++)
    {
        if (ar[i]== key)
            return i;
    }
    return -1;
}
```

Ans 2 : void insertion_sort(int ar[], int n)

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

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

Page:
Date: / /

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

Ans 2: Insertion sort is called online sort because it does not need to know anything about what values it will sort and the information is requested while the algorithm is running

Ans 3

			Best	Worst
(i)	Selection Sort -	Time Complexity	$O(n^2)$	$O(n^2)$
		Space Complexity	$O(1)$	
(ii)	Insertion Sort -	Time Complexity	$O(n)$	$O(n^2)$
		Space Complexity	$O(1)$	
(iii)	Merge Sort	Time Complexity	$O(n\log n)$	$O(n\log n)$
		Space Complexity	$O(n)$	
(iv)	Quick Sort	Time Complexity	$O(n\log n)$	$O(n^2)$
		Space Complexity	$O(n)$	
(v)	Heap Sort	Time Complexity	$O(n\log n)$	$O(n\log n)$
		Space Complexity	$O(1)$	
(vi)	Bubble Sort	Time Complexity	$O(n^2)$	$O(n^2)$
		Space Complexity	$O(1)$	

1)

because it does not
values it will sort and
algorithm is running

est	Worst
(n^2)	$O(n^2)$
(1)	
n)	$O(n^2)$
1)	
gn)	$O(n\log n)$
n)	$O(n^2)$
n)	$O(n\log n)$
	$O(n^2)$
)	

Page :
Date: / /

Ans4.

Sorting	Inplace	Stable	Online
Selection	✓		
Insertion	✓	✓	✓
Merge		✓	
Quick	✓		
Heap	✓		
Bubble	✓	✓	

Ans5:

```
int b_search ( vector<int> ar, int key)
{
        int lo=0 , hi=ar.size()-1;
        int mid;
        while(lo<=hi)
        {
                mid = lo + (hi-low)/2 ;
                if(ar[mid] == key)
                        return mid;
                else if (ar[mid] > key)
                        hi = mid-1;
                else
                        lo = mid+1;
        }
        return -1;
}
int b-search (int ar[], int l, int h, int key)
{
        if(h >= l)
        {
                int mid = bl+(hi-l)/2;
                if (ar[mid]==key)
                        return mid;
                else if (ar[mid] > key)
```

Page :
Date: / /

```
        breturn b_search(arr, l, mid-1, Key);
    else
        return b_search (arr, mid+1, h, Key);
    }
    return -1;
}
```

Time Complexity - Best - ~~$\Theta(n)$~~ $O(1)$
Avg. - $O(\log n)$
Worst - $O(\log n)$

Ans 6 : Reccurence Relation for Binary Search.

$T(n) = T(n/2) + 1$

Ans 8 : Quick sort is the fastest general purpose sort. In most practic situation Quick sort is the method of choice. If stability is concern and space is available, Merge sort can be the best ~~altern~~ option.

Ans 9 : Howfar or close the array is from being sorted if the array is alerady sorted the then the insertion count is 0 but if array is sorted in reverse order the insertion count is max.

Merge Sort

```
int main()
{
    int arr[] = {7, 21, 31, 8, 10, 1, 20, 6, 4, 5};
    int n = sizeof(arr) / sizeof(arr[0]);
    int ans = mergesort(arr, n);
```

```
,l,mid-1, Key);
mid+1, h,Key);

earch.

purpose sort. In
method of choice
s available,
option.

being sorted if
the insertion
n reverse order

, 4,5};
));
```

```
        cout << "No. of inversion " << ans <<endl;
        return 0;
}
int mergesort (int arr[], int n)
{
        int   temp[n];
        return _merge-sort (arr, temp, 0, n-1);
}
int _merge-sort (int arr[], int temp[], int left, int right)
{
        int mid ,inv_count =0;
        if (right > left)
        {
            mid = (right + left)/2;
            inv_count += _merge sort(arr, temp, left, mid);
            inv_count += merge_sort(arr, temp, mid+1, right);
            inv_count += merge sort(arr, temp, left, mid+1, right);
        }
        return inv_count;
}
int  merge_sort (int arr[], int temp[], int left, int mid, int right)
{
      int inv_count =0;
      int i= left, j= mid , k = left ;
      while (i<= mid - 1 && j<= right)
      {
           if(arr[i] < arr[j])
                temp[k++] = arr[i++]
           else
           {
                temp[k++] = arr[j++];
                inv_count = inv_count + (mid-i);
           }
      }
```

Page:
Date: / /

```
while (i<= mid-1)
        temp[k++] = ar(i++];
while [j<= right)
        temp[k++] = ar[j++];
for (i=left ; i<= right ; i++]
{
        ar [i] = temp [i];

return inv_count;
}
```

Q10. The worst case time Complexity of Quick Sort is $O(n^2)$ the worst case occurs when the pivot is always on extreme (smallest or largest) element. This happens when input array is sorted or reverse sorted and either first or last element is picked as pivot.
The best case of quick sort is when we select pivot element as a mean element.

Ans 11: Reccurence Relation.

(1) Merge Sort : $T(n) = 2T(n/2) + n$

(2) Quick sort $T(n) = 2T(n/2) + n$

Merge Sort is more efficient and works ~~mas~~ faster than quick sort in case of larger array size or data sets.
The worst case complexity for quick sort is $O(n^2)$ where $O(n\log n)$ for merge sort.

Ans 12: Stable Selection Sort

```
void stableselSort(int ar[], int n)
{
        for (int i=0 ; i<n-1 ; i++)
        {
```

Page :
Date : / /

```
    int min = i;
    for( int j = i+1 ; j< n ;j++)
    {
        if (a[min] > a[j])
            min = j;
    }
    int key = a[min];
    while (min >= i)
    {   a[min] = a[min-1];
        min--;
    }
    a[i]= key;
    }
}
int main ()
{
    int a[] = {4,5,3,2,4,1};
    int n= sizeof(a) / sizeof (a[0]);
    stable_sel_sort (ar, n);
    for (int i= 0; i<n ;i++)
    {
        cout << a[i]<<" \t";
    }
    Cout<< endl;
    return 0;
}
```

A Ans13:- The easiest way to do this is to use external sorting. we divide our source file into temporary files of size equals to the size of RAM & first sort these files.

→ External Sorting

If the input data is such that it can't adjust in the memory entirely at once, it needs to be stored in a hard disk, floppy disk or any other storage device. This is external sorting

Page :
Date: / /

Internal Sorting - If the input data is such that it can be adjust in the main memory at once then it is called internal sorting.