Q1

```
int Linear( int *arr, int n, int key)
    for i >=0 to n-1
        if arr[i] = key
            return arr[i]
    return -1;
```

Q2 iterative

```
void insertion (int *arr, int n)
    int i, temp, j
    for i ← 1 to n
        temp ← arr[i]
        j ← i-1
        while (j>=0 AND arr[j] > temp)
            arr[j+1] ← arr[j]
            j ← j-1
        arr[j+1] ← temp
```

recursive

```
void insertion (int * arr, int n)
    if (n<=1)
        return
    insertion (arr, n-1)
    last= arr[n-1]
    j= n-2
    while (j>=0 && arr[j] > last)
        arr[j+1]=arr[i]
        j--
    arr[j+1] = last.
```

It is called online sorting bcz it does not need to know anything about what values it will sort & the information is requested while the algorithm is running.

Q3

Algo	Best Case	Worst Case	Space Comp
Bubble Sort	$O(n^2)$	$O(n^2)$	$O(1)$
Insertion	$O(n)$	$O(n^2)$	$O(1)$
Selection	$O(n^2)$	$O(n^2)$	$O(1)$
Merge Sort	$O(n \log n)$	$O(n \log n)$	$O(n)$
Quick sort	$O(n \log n)$	$O(n^2)$	$O(n)$
Heap Sort	$O(n \log n)$	$O(n \log n)$	$O(1)$

Q4

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

Q5 Iterative binary

```
int binary ( int arr[], int l, int r, int x)
{   while (l<=r)
    { int m ← l+(r-l)/2
      if (arr[m]=x)
          return m;
      if (arr[m]<x)
          l=m+1
      else r=m-1
    }
    return -1
}
```

T.C
B.C $O(1)$
Ag $O(\log n)$
Worst $O(\log n)$

Recursive Binary

```
int Binary (int arr[], int l, int r, int x)
{  if (r >= l)
    { int mid = l + (r - l)/2 ;
   if (arr[mid] = x)
        return mid
   else if (arr[mid] > x)
      return binary (arr, l, mid-1, x)
   else return binary (arr, mid+1, r, x)
}
```

TC
B.C - O(1)
Avg = O(log n)
Worst = O(log n)

Q 6 $T(n) = T(n/2) + 1$

Q 7 we can use hashing it will compute it in O(n)

```
void findPair (int nums[], int n, target)
{  unordered_map <int, int> map;
   int i
   for (i = 0; i < n; i++)
    { if (map.find (target - nums[i]) != map.end())
      { cout << "found" ;
      }
      maps [num[i]] = i
    }
    cout << "not found";
}
```

Q8 Quick sort is fastest general purpose sort In most practical situation, It is method of choice, if stability is important & space is available then merge sort is good.

Q9 Inversion for an array indicates how far or close the array is from being sorted, If array is already sorted, then inversion count is 0, but if array is sorted in reverse order than inversion count is max.

arr[] = {7, 21, 31, 8, 10, 1, 20, 6, 4, 5}.

There 28 inversions in the above array

Q10 The worst case occurs when picked pivot is on extrem that is when input array is sorted or ~~this~~ reverse sorted or either first or last element is picked.

Best case of Quick sort is when we select pivot as a mean element.

Q11 Merge sort → $T(n) = 2T(n/2) + n$

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

merge sort works faster than Quick sort in case of large array size

worst case time complexity of QS is $O(n^2)$ & merge sort is $O(n \log n)$