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[j]
        j--
    arr[j+1] = last.
```

It is called online sorting because it does not need to know anything about what values it will sort and the information is request while the algorithm is running.

Q3

Algo	Best case	worst case	Space complexity
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	$O(n \log n)$	$O(n \log n)$	$O(1)$

Q 4

Sort	Inplace	Stable	Online
Selection	✓	✗	✗
Insertion	✓	✓	✓
Merge	✗	✓	✗
quick sort	✓	✗	✗
Heaps	✓	✗	✗
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 $O(1)$

Avg C $O(\log n)$

worst.C $O(\log n)$

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)
   }
```

T.C.
B.C. O(1)
Avg C O(log n)
worst C O(log n)

Q6 $T(n) = T(n/2) + 1$

Q7 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"; }
       map [num [i]] = i
  }
  Count << "not found";
}
```

Q8 Quick Sort is fastest general purpose sort in most paractical 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, 1, 4, 5 }.

There 28 inversion in the above array.

Q10 The worst can occur when picked pivot is an extreme that is when input array is sorted or 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 quick sort is $O(n^2)$ & merge sort is $O(n \log n)$