_/_/_

Tutorial → 3

①
```
int (linearSearch (int arr [], int n, int key)
{
    for (i = 0 to n-1)
    {
        if (arr [i] == key)
            return i;
    }
    return -1;
```

② Iterative Insertion Sort

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

③ Recursive Insertion Sort :

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

In Insertion Sort, we give i/p one by one and place each one at right order with comparison from already placed element. We need not the whole array simultaneous to operate algorithm. ~~so~~ ~~is~~

Let A[] = {11, 3, 4, 9, 7}

Step:

1: A[] = {11, 3, 4, 9, 7}

2: A[] = {3, 11, 4, 9, 7}

.... and so on

Insertion is online sorting rest of the sortinge are offline.

__/__/__

Ans.)
3)
4)

	T.C	S.C	Stable	Inplace	Online
i) Bubble sort-	n^2	1	✓	✓	✗
Selection sort-	n^2	1	✗	✓	✗
Insertion sort-	n	1	✓	✓	✓
Merge sort-	nlogn	n	✓	✗	✗
Quick sort-	nlogn	n	✗	✗	✗
Heap sort-	nlogn	~~✗~~ 1	✗	✓	✗

Ans.5) <u>Iterative Binary search:</u>

```
int   Binary_search(int A[], int n)
  {
      int low=0, high = A.length -1;
     while (low <= high)
        { int mid = low+high)/2;
      if (n== A[mid]) {
              return mid;
      }
   else if ( n < A [mid]) {
           haigh = mid -1;
       }
  else {
        low =mid +1;
   }
```

<u>Recursive Binary search:</u>

```
bool  Binary search (int arr[], int l,
                  int r, int key)
```

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

}
```

	Linear	Binary
Recursive T.C	O(n)	O(log n)
Iterative T.C	O(n)	O(log n)
Iterative S.C	O(1)	O(1)
Recursive S.C	O(log n)	O(log n)

__/__/__

Qus 7)
```
bool checkPair (int A[], int n, int k)
  {
    Take Hash Table H of size O(n)
    for (i=0; to n-1)
      {
        int x = K - A[i]
        if (H.Search (x) is true)
          return 1
        H.insert (A[i])
      }
    return -1
  }
```

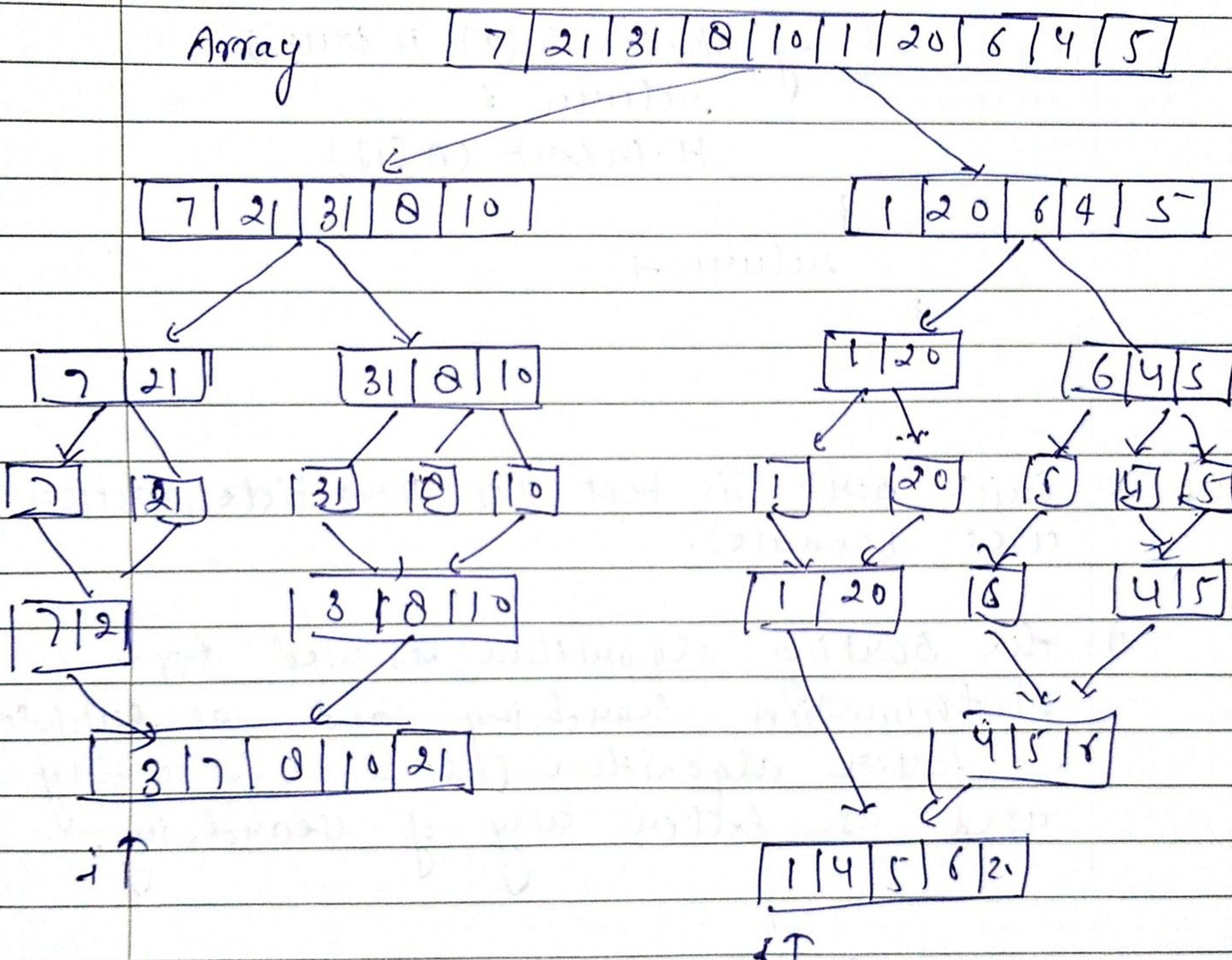
Qus. 8) Quick sort is best for ~~practide~~ practical cases because:-

1) the sorting algorithm is used for information searching and as Quicksort is faster algorithm so it is widely used as better way of searching.

ii) It is used everywhere where a stable sort is not needed.

(4) Quick sort is cache-friendly algorithm as it has a good locality of reference when used for array.

Ques. 9) Inversion ~~sort~~ count for an array indicates how far (or close) the array is already sorted, then the inversion count is 0.
but if the array is sorted in reverse order, the inversion count is maximum.



inversion pair → (3,1) (1,1) (8,1) (10,1) (21,1)

| 3 | 7 | 8 | 10 | 21 |

| 4 | 5 | 6 | 20 |

inversion pair → (7,5), (8,8), (19,5) (21,5)

| x | 7 | 8 | 10 | 21 |

| x | $\overset{5}{\cancel{x}}$ | x | 6 | 20 |

~~invrsion Pair → (7,4), (8,4), (10,4), (21,4)~~

inversion pair → (7,5) (8,5) ,(19,5) (21,5)

| x | 7 | 8 | 10 | 21 |

| x | x | x | 6 | 20 |

inversion pair → (7,6) (8,6) (16,6) (21,6)

| x | 7 | 8 | 19 | 21 |

| x | x | x | ~~6~~ 20 |

inversion pair → (21,20)

Ques. 10) The best case occure when the partition process always picks the middle element as pivot.

when the array is reverse sorted or already sorted quick sorted becomes worst.