PIYUSH RANA

CST SDL 1

(35)

Tutorial 3

Design & Architecture of Algorithms.

Answer 1:

```
int linear_search ( int A[], int n, int t )
{
  if((abs A[0]-t) > (abs(A[n-1]-t))
   for (i=n-2 to 0 ; i--)
  if (A[i]==t ) { return i;}

 else {
  for (i=0 to n-1; i++)
  if (A[i]==t)
      return i;
 }
```

Answer 2: Iterative Insertion Sort

```
void insertion (int A[8], int n )
  {
    for (i=1 to n)
    { t=A[i];
      j=i;
    while (j>=0 && t<A[j])
    {
      A[j+1] = A[j];
        j-- ;
    }
    A[j+1] = t;
   }
  }
```

Recursive Insertion Sort

```
void insertion (int A[], int n)
{
    if (n<=1)
    return;
    insertion (A, n-1);
    int last = A[n-1];
     int j = n-2;
    while (j>=0 && A[j]>last)
    {
        A[j+1]=A[j];
          j--;
    }
      A[j+1] = last;
}
```

Insertion sort is also called online sorting algorithm because it will work if the elements to be sorted are provided one at a time with the understanding that the algorithm must keep the sequence sorted as more elements are added in.

Other sorting algorithm like bubble sort, insertion sort, heap sort etc are considered external sorting technique as they need the data to be stored in advance.

Answer 3:

Sorting	Best case	Worst case.
(1) Bubble Sort	$O(n^2)$	$O(n^2)$
(2) Selection Sort	$O(n^2)$	$O(n^2)$
(3) Insertion Sort	$O(n)$	$O(n^2)$
(4) Count Sort	$O(n)$	$O(n+k)$
(5) Quick Sort	$O(n \log n)$	$O(n^2)$

Merge sort	O(nlogn)	O(nlogn)
Heap sort	O(nlogn)	O(nlogn)

Answer 4:

Sort	Implace	Stable.	Outline
Bubble	✓	✓	x
Selection	✓	x	x
Insertion	✓	✓	✓
Count	x	✓	x
Quick	✓	x	x
Merge	x	✓	x
Heap.	✓	x	x

Answer 5: Recursive / Iterative pseudo code for binary search:

Iterative:

```
int binary_search (int arr[], int x)
{
  int l=0; r=arr.length-1;
  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;
```

Recursive:

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

Linear Search:

Iterative: Time complexity = O(n)
Space complexity = O(1)

Recursive Time complexity = O(n)
Space complexity = O(n)

Binary Search:

Iterative : Time complexity = $O(n \log_5 n)$
Space complexity = O(1)

Recursive Time complexity = $O(n \log_5 n)$
Space complexity = $O(\log_5 n)$

Answer 6:

T(n)
↓
T(n/2)
↓
T(n/4)
⋮
↓
T(n/2^k)

Recurrence Relation = T(n/2) + O(1)

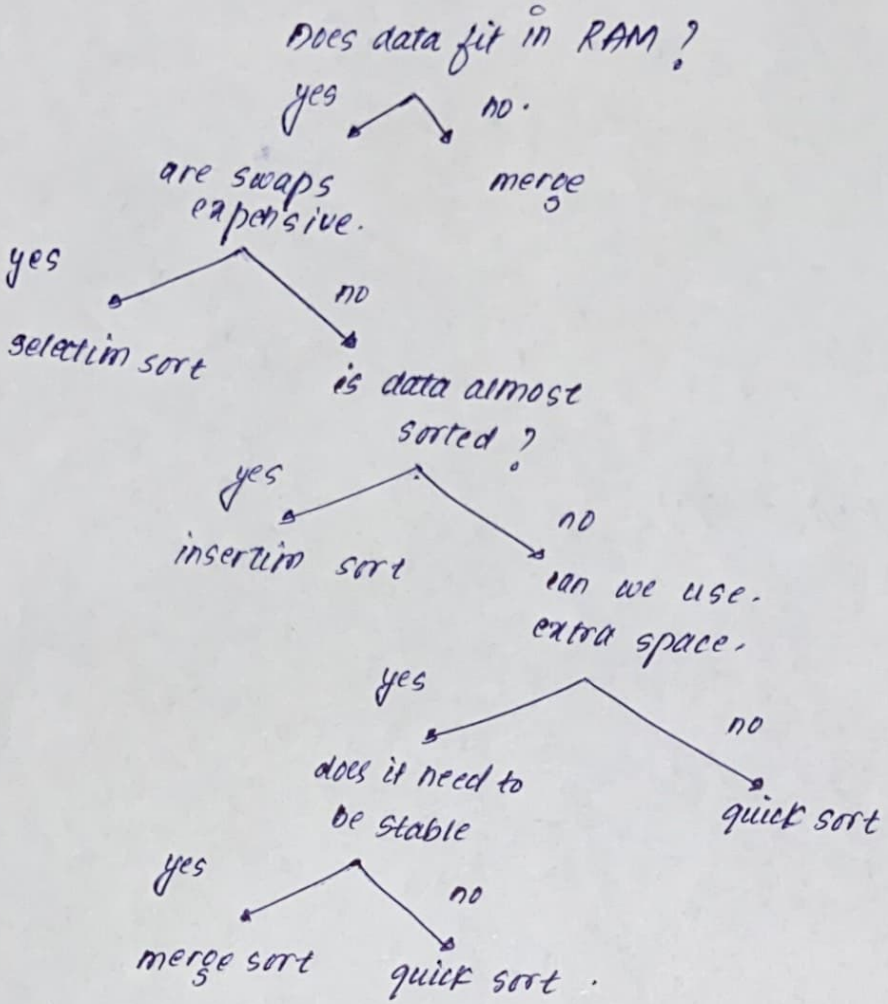
Answer 7:

```
int n;
int A[n];
int key;
int i=0; j=n-1;
while (i<j)
{
if ((A[i]+A[j]) = key
Break;
else if (A[i]+A[j] > key):
   j--;
else
   i++;
}
count << i << " " << j;
```

Time complexity = O(n log n)

Answer 8: (i.) run time
(ii.) space
(iii.) stable
(iv.) no. of swaps
(v.) will the data fit in the RAM.

There is no best sorting algorithm. It depends on the situation or the type of array provided.



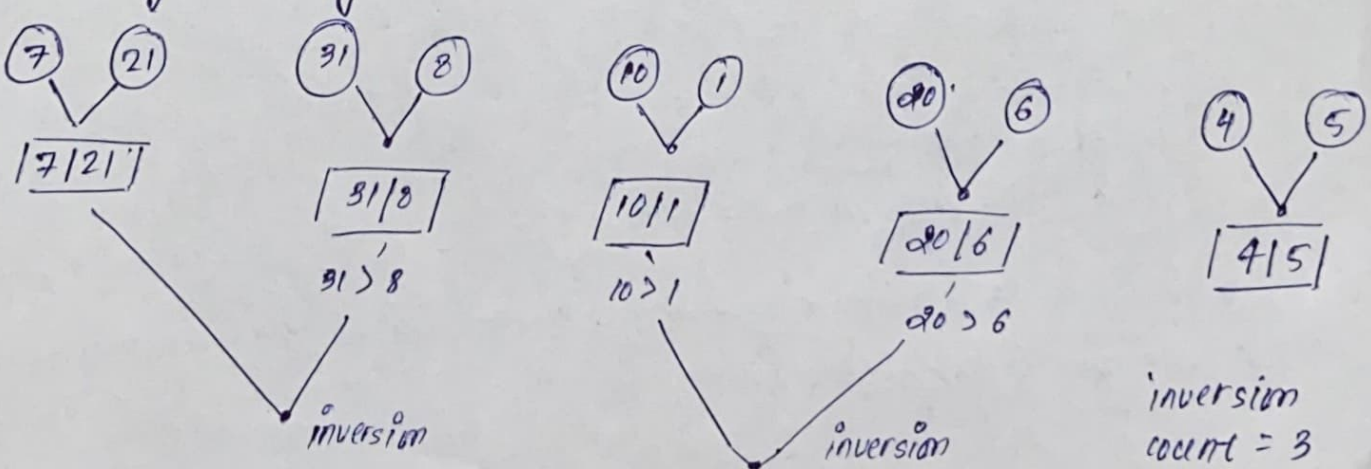
Answer 9: Inversion in an array indicates how the array is sorted. If the array is already sorted, the inversion count is 0, but if the array is sorted in reverse order, then the inversion count is maximum.

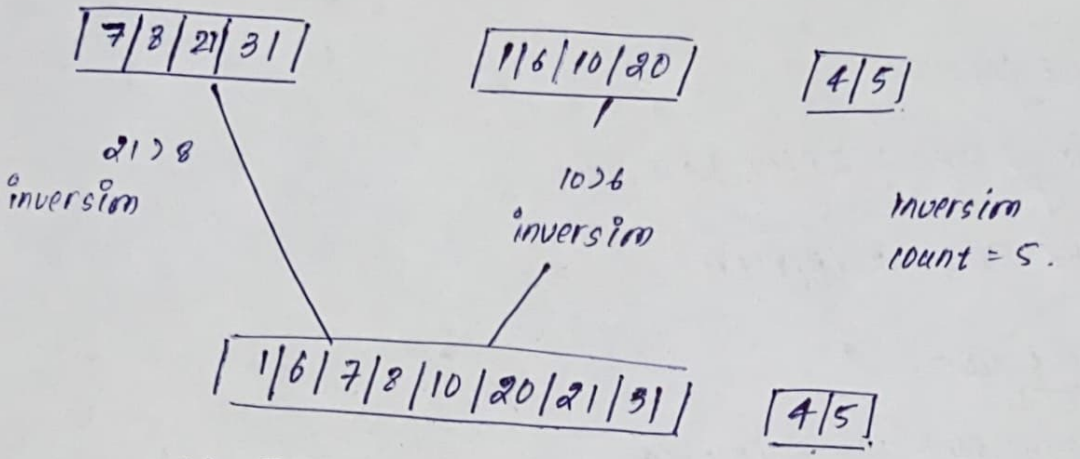
condn for inversion :

a[i] > a[j] & i<j

| 7 | 21 | 13 | 8 | 10 | 1 | 20 | 6 | 4 | 5 |

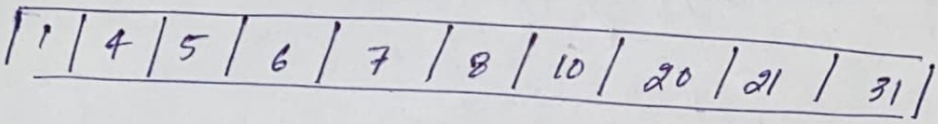
Dividing the array :





7>1, 7>6, 8>1, 8>6, 21>10, 21>20, 31>1, 31>6,
31>7, 31>20, 31>1, 21>6.

Total no. of inversion in this step = 12.



6>4, 6>5, 7>4, 7>5; 8>4, 8>5, 10>4, 10>5, 20>4,
20>5, 21>4, 21>5, 31>4, 31>5.

Total inversion in this step = 14

Inversion count = 31 = (5+14+12)

Answer 10:

Best case:

Time complexity = O(nlogn)

The best case occurs when the partition process always picks the middle element as pivot.

Worst case:

Time complexity = $O(n^2)$

when the array is stored in ascending or descending order.

Answer 11: Best case:

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

Quick Sort : $2T(n/2)+n$

Worst case:

Merge sort : $2T(n/2)+n$

Quick Sort : $T(n-1)+n$

Similarities:

They both work on the concept of divide & conquer algorithm. Both have best case complexity of $O(n\log n)$.

Difference:

Merge Sort	Quick Sort
(i.) the array is divided into just a half.	(i.) the array is divided in any ratio.
(ii.) Worst case complexity is $O(n\log n)$.	(ii.) Worst case complexity is $O(n^2)$.
(iii.) It requires extra space i.e. NOT inplace.	(iii.) It does not require any extra space i.e. inplace.
(iv.) It is external sorting algorithm and it is stable.	(iv.) It is internal sorting algorithm and is not stable.
(v.) Works consistently on any size of data set.	(v.) Works fast on small dataset.

Answer 12: Selection sort is not stable by default but you can write a version of stable selection sort.

```
void selection ( int A[], int n )
{
 for (int i=0; i<n-1; i++)
  {
   int num=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;
   }
  }
```

Answer 13:

```
void bubble sort (int A[], int n]
 {
    int i,j ;
    int j=0;
  for (i=0; i<n; i++)
   {
    for (j=0; j<n-1; j++)
    {
     if ( A[j] > A[j+1] )
     {
      swap(A[j], A[j+1])
       j=1;
     }
    }
    if (j==0):
     break;
   }
```

Answer 14: When dataset is large enough to fit inside RAM, we ought to use merge sort. because it uses the divide and conquer approach in which it keeps dividing the array into smaller parts until it can no ~~tog~~ longer be splitted it then merge the array divided in n parts. Therefore at the time only a part of array is taken on RAM.

External Sorting:-

It is used to sort massive amount of data. It is required when the data doesn't fit inside the RAM & instead they must reside in the slower external memory.

During sorting, chunks of small data that can fit in main memory are read, sort and written out to a temperory file.

During merging, the sorted subfiles are combine into a single large file.

Internal Sorting:-

It is a type of sorting which is used when the entire collection of data is small enough to reside within RAM. Than there is no need of external memory for program execution. It is used when input is small.

Example: Insertion Sort, quick sort, heap sort etc.