Asha Jindal Tutorial - 3 (1)

Sol 1^n - // Linear search with minimum no. of comparison.

To reduced no. of comparison → apply binary search algo.

binary-search (int * a, int n, int target)

```
{
  int low = 0;
  int h = n-1;
  while(low<= h)
  {
    int mid = (low + h)/2
    if ( a[mid] == target)
    {   return mid;
    }
    else if (a[mid] < target)
        l = mid + 1
    else if( a[mid] > target)
        h = mid-1;
  }
}
```

Sol^n:- Iterative / Recursive function for (2)

Iterative Insertion sort

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

Recursive Insertion Sort

```
void insertion_sort (int ind, int a[], int n)
{
    if(ind==n)
        return;
    int j= ind-1
    int temp= a[ind]
    while (j>=0 && a[j]>temp)
```

```
a[j+1] = a[j]
j--;
}
a[j+1] = temp;
insertion sort (ind+1, a, n);
}
```

Online sorting:-

An online algorithm is one that process its input piece by peice in serial fashion. i.e with out knowing the entire i/p available from the beginning.

Insertion sort is known as online sorting algorithm because an online algorithm does not know the whole i/p. It might make decision that later turn out not to be optimal.

where as selection, bubble sort Algorithm repeatedly the minimum and comparing two elements respectively which requires access to entire i/p.

Ans3:- Complexity of All Sorting Algorithm (4)

	Average T.C
Selection	$O(n^2)$
Bubble	$O(n^2)$
Insertion	$O(n^2)$
Merge Sort	$O(n\log n)$
Quick Sort	$O(n\log n)$
Heap Sort	$O(n\log n)$

Ans4:

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

Ans 5 : Iterative binary search

```
void binary_search (int *a, int n)
{
    int low = 0;
    int high = n-1;
    while (low <= high)
    {   int mid = (Low+high) / 2
        if (a[mid] == target)
        {
            return mid;
        }
        else if (a[mid] < target)
            low = mid+1
        else if (a[mid] > target)
            high = mid-1;
    }
}
```

T.C = O(log n)

S.C = O(1)

Recursive Binary Search

```
void binary search( int * a, int low, int h
                          int target)
{
    if( low<= high)
    {
      int  mid = (low+ high) / 2;
    if( a[mid ]= = target)
          return mid ;
   else if (a[mid] < target)
   binary search( a, mid+1, high, target)
   else
   binary-search ( a, l, mid -1, target)
 }
```

T.C = O(n Log n)

S.C = O (n) → because of the recursive calls that are made

Time complexity of linear search

T.C = O(n).

Ans6.- Recurrence Relation for binary search

$$T(n) = T\left(\frac{n}{2}\right) + 1 \quad \text{—} \; ① \qquad T(1) = 1$$

Applying Backward Substitution method

Put $n = \frac{n}{2}$ in eq^n ①

$$T\left(\frac{n}{2}\right) = T\left(\frac{n}{4}\right) + 1 \quad \text{—} \; ②$$

from eq^n ① & ②

$$T(n) = T\left(\frac{n}{4}\right) + 1 + 1 \quad \text{—} \; ③$$

Now put $n = \frac{n}{4}$ in eq^n ①

$$T\left(\frac{n}{4}\right) = T\left(\frac{n}{8}\right) + 1 \quad \text{—} \; ④$$

from eq^n ③ & ④

$$T(n) = T\left(\frac{n}{8}\right) + 1 + 1 + 1$$

$$T(n) = T\left(\frac{n}{2^3}\right) + 3 \Rightarrow T(n) = T\left(\frac{n}{2^k}\right) + k \quad \text{—} \; ⑤$$

Put $\frac{n}{2^k} = 1$

$n = 2^k$

$$\boxed{k = \log_2 n}$$

from eq^n 5 it

$$T(n) = T\left(\frac{n}{2^k}\right) + k$$

$$T(n) = T\left(\frac{n}{2^{\log_2 n}}\right) + k$$

$$T(n) = T\left(\frac{n}{n^{\log_2 2}}\right) + k$$

$$T(n) = T\left(\frac{n}{n}\right) + \log n \quad [\because k = \log_2 n]$$

$$T(n) = T(1) + \log n$$

$$T(n) = 1 + \log n$$

$O(\log n)$ Ans

Ans 7:-

```
findIndex(int a[], int n, int k)
{
    i=0; j=1;
    while (i<n && j<n)
    {
        if(i!=j && a[j]-a[i]==k ||
                 a[i]-a[j]==k)
            printf("%d %d", i, j);
        else if (a[j]-a[i]<k) j++;
        else   i++;
}
```

Ans8:- Quick sort is one of the most effecient sorting algorithms which makes it one of the most used aswell, It is faster as compared to other sorting algorithms. And its time complexity is O(n log n)

But in case of a larger array. Merge sort is preffered.

Ans 9:- Inversion refers to that element how far (or close) the array is from being sorted.

```
int merge (int * a, int * t, int l, int mid,
           int h)
{
    int inver = 0;
    int i = l;
    int j = mid;
    int k = l
  while(i<= mid-1 && j<=h)
    { if(a[i] < a[j]
        t[k++]= a[i++]
      else
        {
        t[k++] = a[j++]
        inver + = (mid -i);
        }
  }
  while (i<= mid - 1)
    { t[k++]= a[i++];
  while ( j<= h)
     t[k++] = a[j++];
```

```
for(i=l; i<=h; i++)
  {
    a[i] = t[i];
  }
  return inver;
}
int merge-sort(int *a, int *t, int l,
               int h)
{
   int inv = 0;
   if(l<h) {
     mid = (l+h)/2
  invt = merge-sort(a, t, l, mid);
  invt = merge-sort(a, t, mid+1, h);
  invt = merge(a, t, l, mid+1, h);
   }
}
```

Ans10:-

Best Case:- $O(n \log n)$

Worst Case:- $O(n^2)$ when the array is sorted or reverse sorted.

Ans 11:- Merge sort

Best case:- $T(n) = 2T\left(\frac{n}{2}\right) + n$

Worst case $T(n) = 2T\left(\frac{n}{2}\right) + n$

Quick sort

Best case $T(n) = 2T\left(\frac{n}{2}\right) + n$

Worst case $T(n) = T(n-1) + n$

Similarities

1. Both Merge and Quick sort are based on Divide and conquer algorithm.
2. Both algorithms are offline.

Difference-

1. Merge sort use extra space of $O(n)$ while quick sort does not require extra space.

[illegible]

(2) Worst case time complexity of merge sort : O(n log n)

Worst case time complexity of quick sort is O(n²).

(3) Merge sort is stable where as Quick sort is not stable.

Ans 12:-

```
void selection sort ( int *a, int n)
{
  for( i = 0 to n-1)
   {
     temp = a[i]
      pos = i
     for( j = i+1 to n-1)
     { if (a[j] < temp)
       { temp = a[j]
         pos = j
       }
     }
   swap(a[pos], a[i]);
}
```

Ans(3):-

```
for(int i=0; i<n; i++)
{ int count = 0;
  for(int j=0; j<n-i; j++)
  { if (a[j] > a[j+1])
    { swap(a[i], a[j+1]);
      count++;
    }
  }
  if (count == 0)
    break;
}
```