Name : Gharinath Hapati Class : IST-SP1 Roll No : 33

Tutorial - 3

Que 1

```
int l-search ( int A[], int n, int t)
{  if (abs(A[0]-t) > abs(A[n-i]-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;
   }
}
```

Que 2) Iterative insertion sort

```
void insertion-sort ( int A[], 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;
   }
}
```

Jhansinath

Recursive insertion sort

```
void i_sort (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 algo 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.

Other sorting algorithms like bubble sort insert sort, heap sort etc are considered external sorting algorithms as they need the data to be sorted in advance.

Shainath

Que 3 Complexity of sorting algorithms:

Sorting	Best Case	Worst Case
Bubble sort	$O(n^2)$	$O(n^2)$
selection sort	$O(n^2)$	$O(n^2)$
Insertion sort	$O(n)$	$O(n^2)$
count sort	$O(n+r)$	$O(n^2)$
quick sort	$O(n \log n)$	$O(n^2)$
merge sort	$O(\log n)$	$O(n \log n)$
heap sort	$O(n \log n)$	$O(n \log n)$

Que 4

Sort	Inplace	stable	Online
Bubble	✓	✓	×
selection	✓	×	×
Insertion	✓	✓	✓
count	×	✓	×
quick	✓	×	×
merge	×	✓	×
heap	✓	×	×

Que 5 Recursive / Iterative pseudo code for binary search.

Iterative :

```
int bs(int a[], int x)
{
    int l=0, r = arr.length - 1;
    while (l<=r)
    {
        int m = l+(r-l)/2;
```

ghainath

```
if (a[m] == x);
    return m;
if (a[m] < x)
    lb = m+1;
else
    ub = m-1;
}
return -1;
}
```

Recursive

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

Linear search

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

Shrinath

Recursion : Time = O(n)
Space = O(n)

Binary search Iterative Time complexity : O(n log n)
Space complexity : O(1)

Recursion Time : O(n log n)
Space : O(log n)

Que 6

T(n)

↓

T(n/2)

↓

T(n/4)

↓

⋮

T(n/2^k)

recurrence relation = T(n/2) + (0/1)

Que 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++;
```

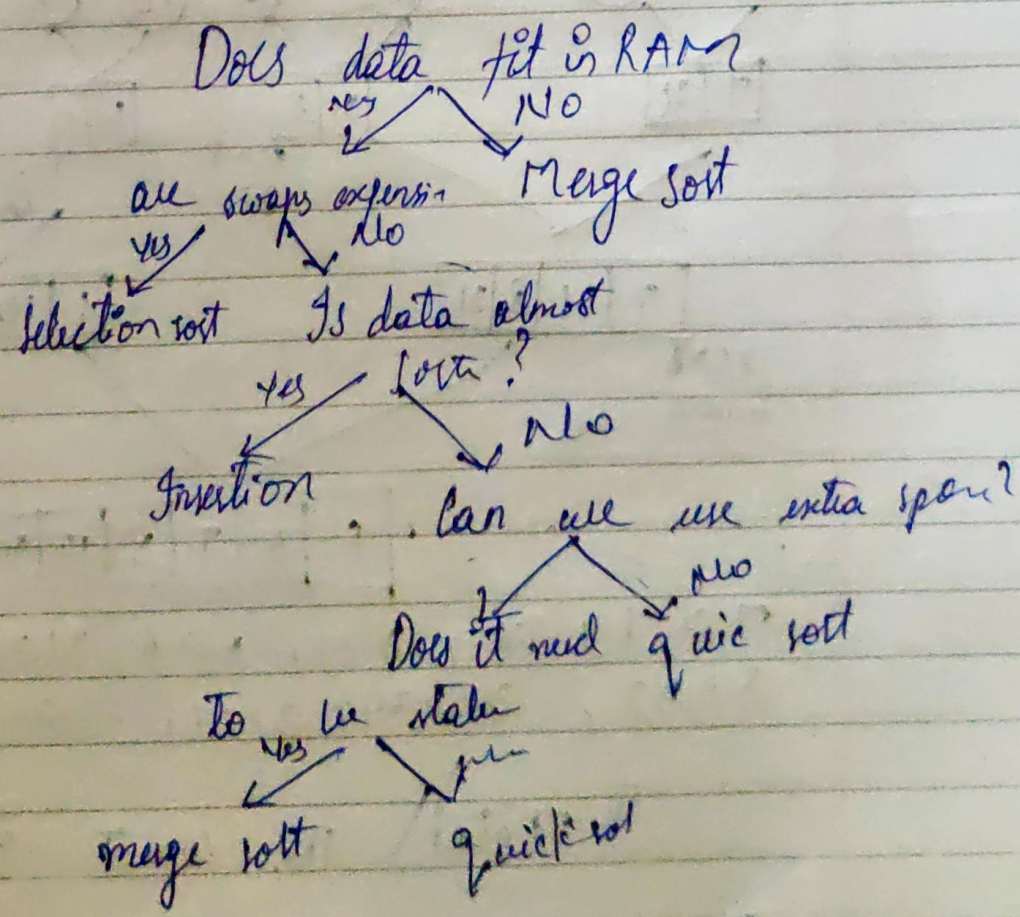
ghainath

cout << i << " " << j

Time complexity : O(nlogn)

Ques i) run time
ii) space
iii) stable
iv) no of swaps
v) will the data fit in the RAM

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

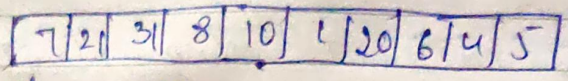


Ans 9

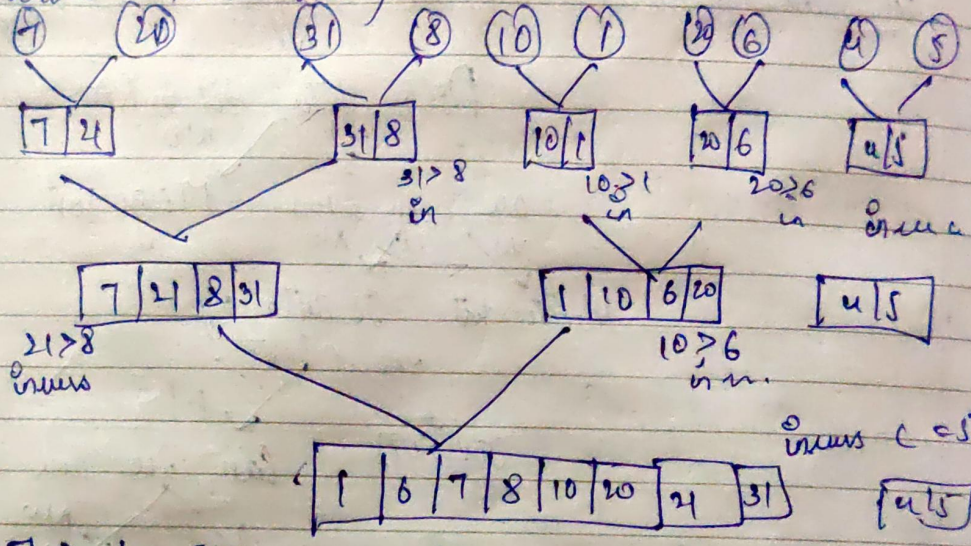
Inversion is an array indicates how far the array in form using sorted if the array is already sorted the inversion count is 0, but if the array is sorted in reverse order, then the inversion count is maximum.

Condition for inversion

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

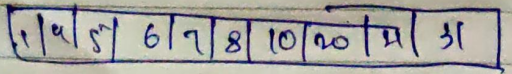


Divide the array



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

Total in, count 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

ghaiwats

Ques 10 Best case
Time complexity = $O(n \log n)$

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 sorted in asending or desending order.

Ques 11 Best case
merge sort :- $2T(n/2) + n$
quick sort : $2T(n/2) + n$

Worst case

merge sort : $2T(n/2) + n$
quick " : $T(n-1) + n$

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

Differences

Merge	Quick
i) The array is divided into 2	i) The array is divided in any ratio.
ii) Worst case time complexity is $O(n \log n)$	ii) Worst case is $O(n^2)$

shravanth

ii) It uses extra space i.e not inplace	iii) It is inplace
iv) It is external & stable sorting algo.	iv) It is internal & not stable algo.
v) It works consistently on any size of data	v) It works fast on small data set.

Ans 12 selection sort is inplace not stable but you can write a version of stable selection sort.

```
void S-S. (int A[], int n)
    for (int i=0; i<n-1; i+1)
    {
        int num = i;
        for (int j = i+1; i<n; j++)
        {
            if (A[num] > A[j])
                min = j;
            int key = A[min];
            while (min > i)
            {
                A[min] = A[min-1]
                min -= i
            }
            A[i] = key;
        }
```

Jharinath

Que 13

```
void bubble sort (int a[], int n)
{
    int i, j;
    int f = 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])
                f = 1;
            };
        }
        if (f = = 0)
            break;
    }
```

Que 14 When the data is large external enough to fit in RAM, we ought to use external sort(merge) because it uses the divide & conquer approach in which it keeps dividing the array into smaller parts until it can no longer be split. it is the merge the array divided into n parts. Therefore at the time only a part of array is taken into 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.

Jhairath

During sorting chunks of small data that can fit in main memory are read, sort & written out to a temporary file. During merging sorted sub files are combined into a single large file.

Internal Sorting

It is a type of sorting which is used when the total collection of data is small enough to reside in RAM. Then there is no need of external method for program. It is used for small input.

Eg: Insertion, quick, heap sort etc.