DS . ASSIGNMENT (Sorting And BST)

(1.) Insertion sort algorithm is

```
Insert ( int a[ ],n)
{
    for ( int i=1; i<n; i++)
    {
        int t = a[i];
        int j = i-1;
        while ( j ≥ 0 && a[j] > a[i])
        {
            a[j+1] = a[j];
            j = j-1;
        }
    a[j+1] = t;
    }
}
```

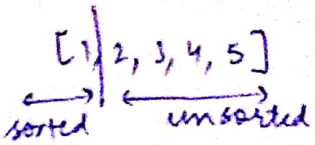
The time complexity of this algorithm is $O(n^2)$ for worst case.

consider an array a = [1, 2, 3, 4]

here, n = 4

For the best case analysis,

$a = [1, 2, 3, 4, 5]$, $n = 5$



			Time required	Array
$i = 1$	$j = 0$, $t = 2$ $a[j] = 1 \not> 2$	$a[j+1] = t$ $a[1] = 2$	c	[1, 2 | 3, 4, 5]
$i = 2$	$j = 1$, $t = 3$ $a[j] = 2 \not> 3$	$a[j+1] = t$ $a[2] = 3$	c	[1, 2, 3 | 4, 5]
$i = 3$	$j = 2$, $t = 4$ $a[j] = 3 \not> 4$	$a[j+1] = t$ $a[3] = 4$	c	[1, 2, 3, 4 | 5]
$i = 4$	$j = 3$, $t = 5$ $a[j] = 4 \not> 5$	$a[j+1] = t$ $a[4] = 5$	c	[1, 2, 3, 4, 5 |] sorted
⋮				
$i = n-1$			c	
$i = n$	x			

Total time taken = c X (n-1) times

= $\boxed{O(n)}$

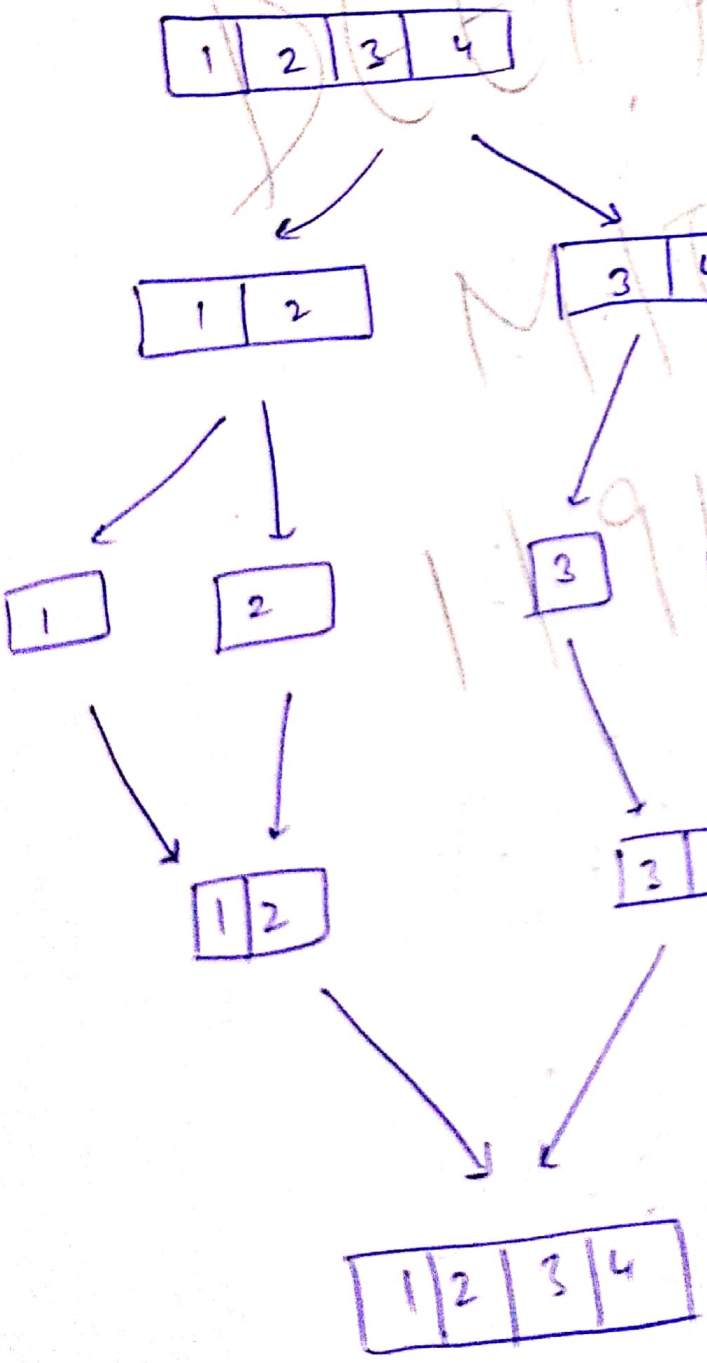
For insertion sort, the best case will be a sorted list with time complexity

= $\boxed{\omega(n)}$

omega (n)

To increase time efficiency of insertion sort,

① We can use recursive version of ~~insertion~~ merge sort in which we break an array recursively into smaller parts and then, decreasing time complexity from $O(n^2)$ to $O(n \log n)$.

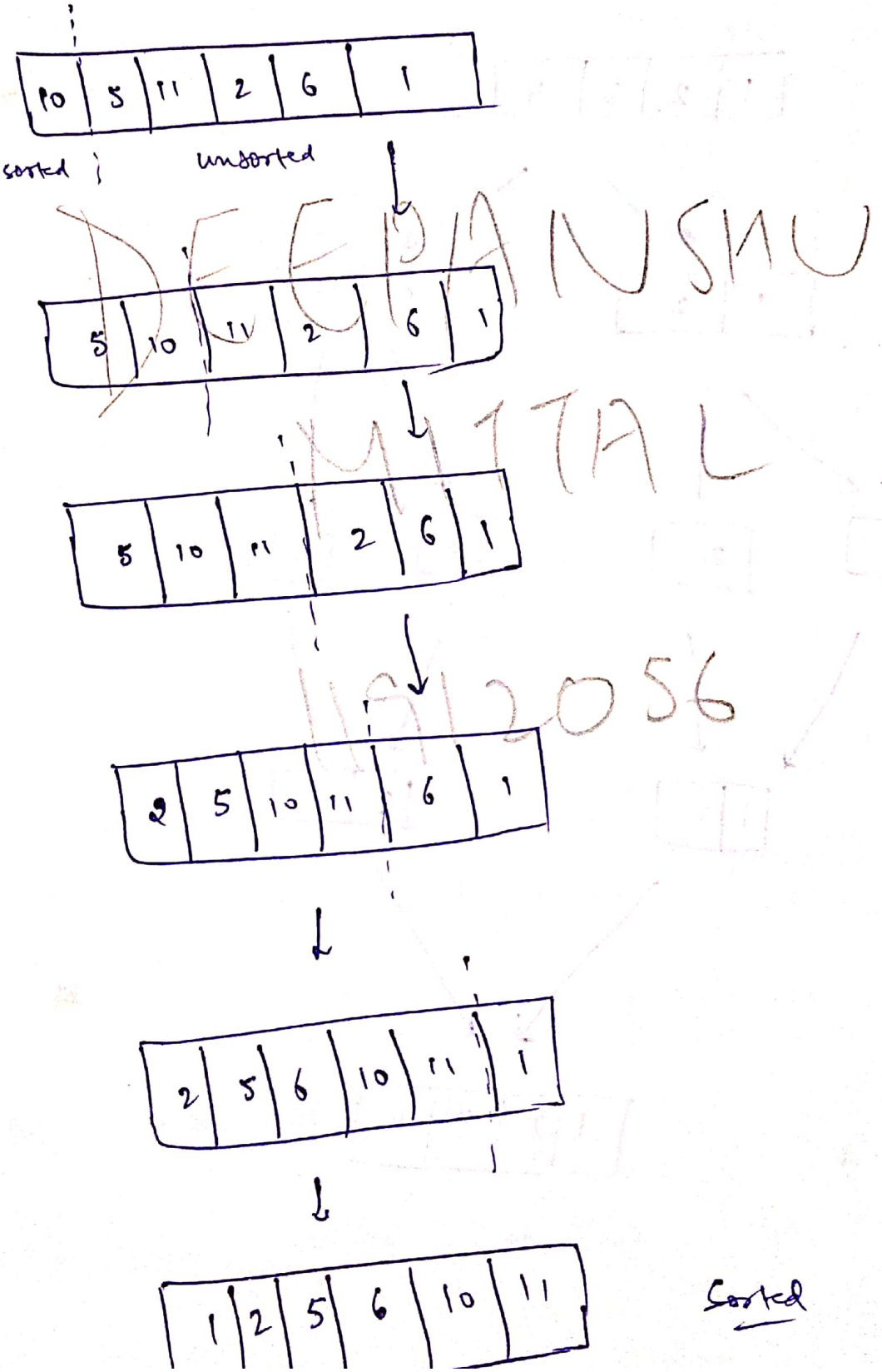


(2) Let us take an example of an array

A:

10	5	11	2	6	1

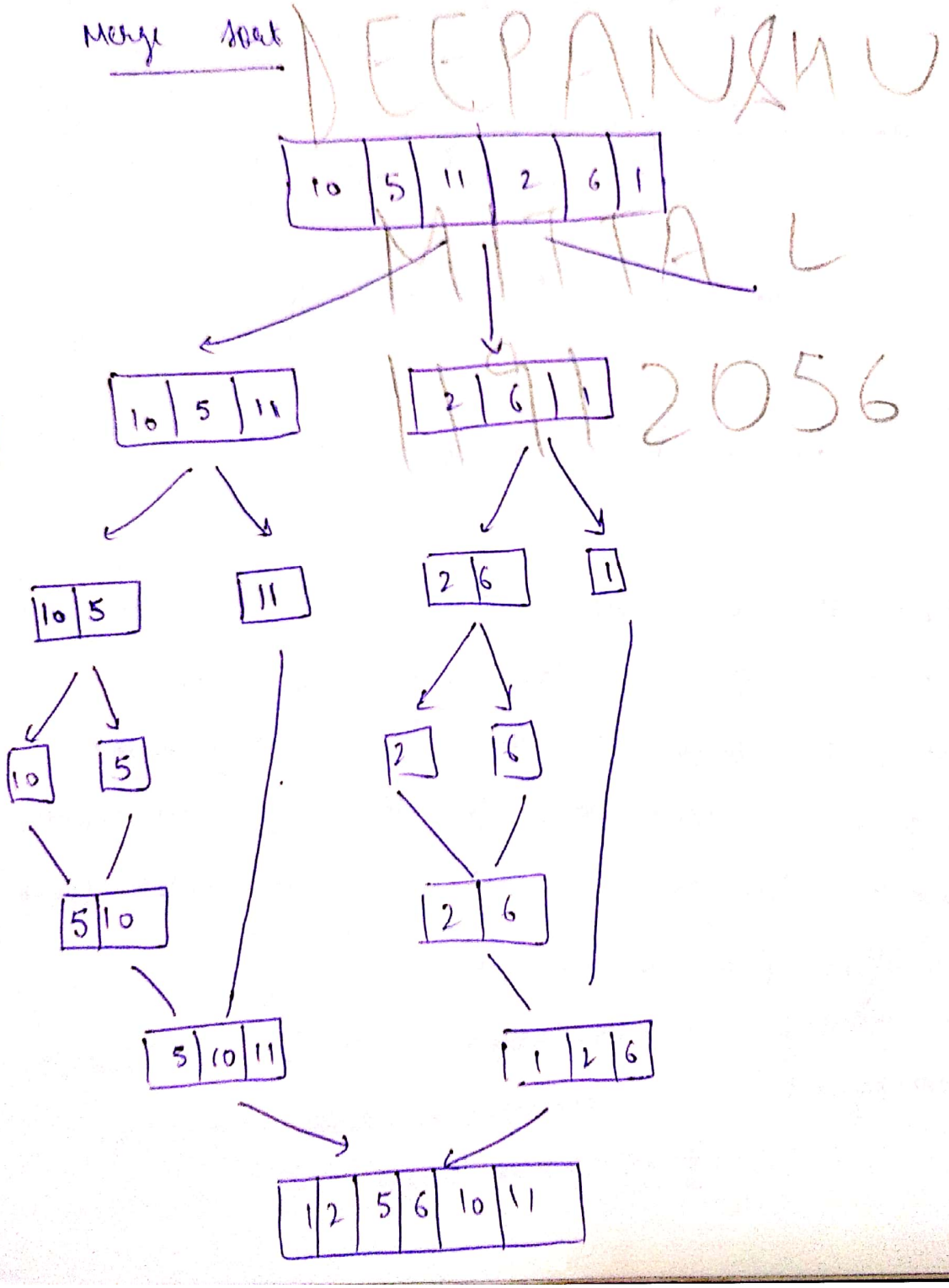
using Insertion sort :



This is inplace sorting algorithm, as the changes are made in the same array or no extra use is made.

Time complexity of insertion sort is $O(n^2)$ in worst case and $O(n)$ is best case

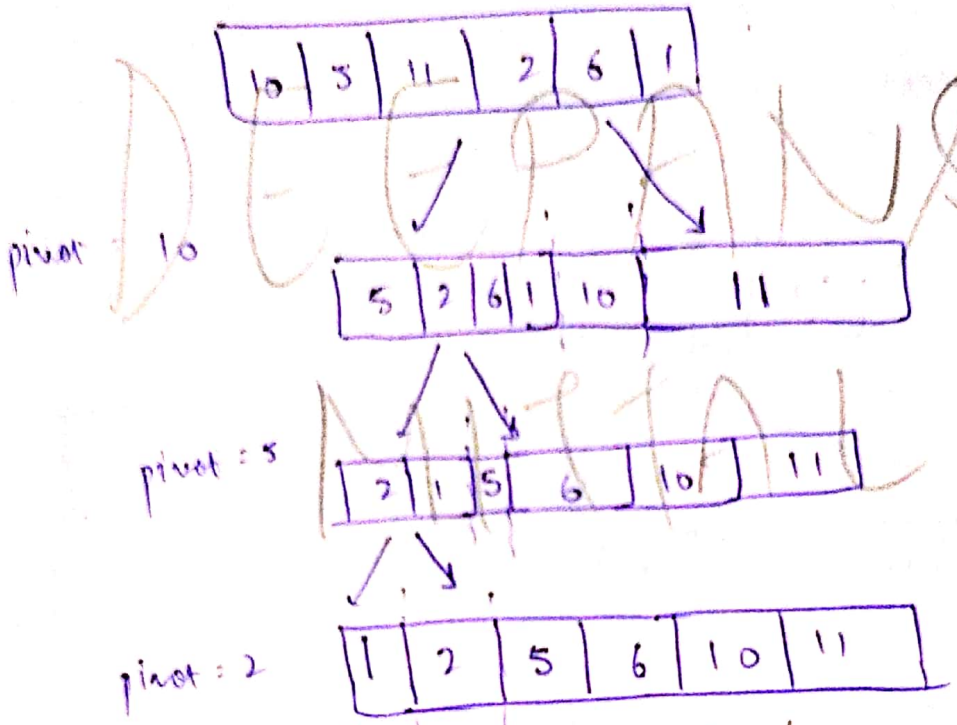
Merge sort



Merge sort is not - in place sorting algorithm

Time complexity is O(n log n)

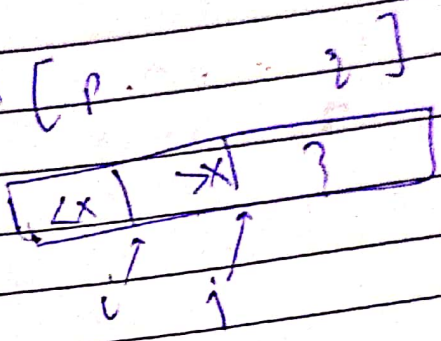
Quick sort



Quick sort is in place sorting algorithm as it does not need an extra space and produces an output in same memory that contains the data by transforming the input 'in-place'. However, a small constant extra space used for pivot is allowed.

Page No.
Date:

QUICK SORT




```
partition ( A[p ... q])
{
  x ← A[p]
  i ← p
  for j ← p+1 to q
      do if A[j] < x
          i ← i+1
          exchange A[i] ↔ A[j]
  A[p] ↔ A[i]
  return(i);
}
```

→ t.c = $\theta(n)$

```
Quicksort ( A[p ... q])
{
  if (p == q) then done
  else r ← partition (A[p ... q])
  recursively
      Quicksort (A[p ... r-1])
      Quicksort (A[r+1 ... q])
}
```

Page No.
Date:

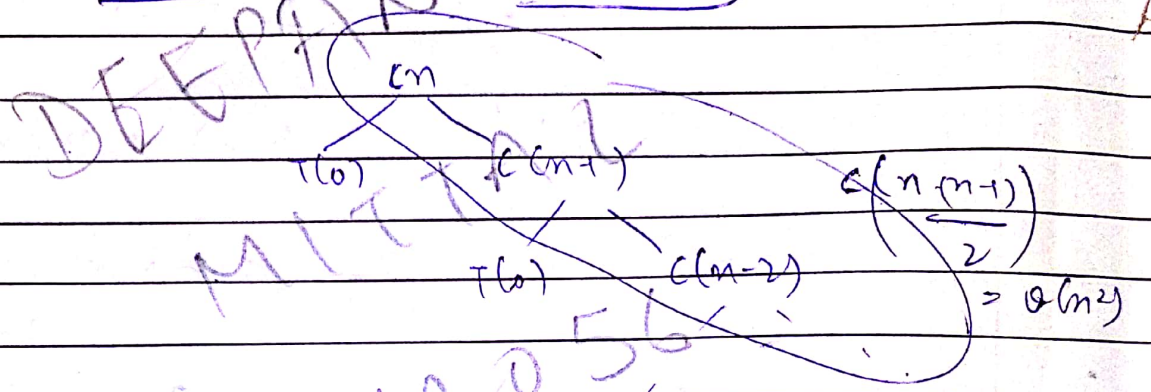
$T(n) = \theta(1) + \theta(n) + T(0) + T(n-1)$

$= \theta(n) + T(n-1)$

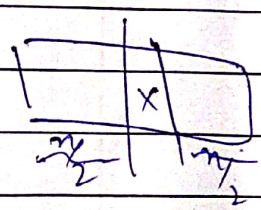
$$T(n) = T(n-1) + \theta(n)$$

$T(n) = T(0) + T(n-1) + cn$

in worst case $\Rightarrow \boxed{\theta(n^2)}$ sorted array



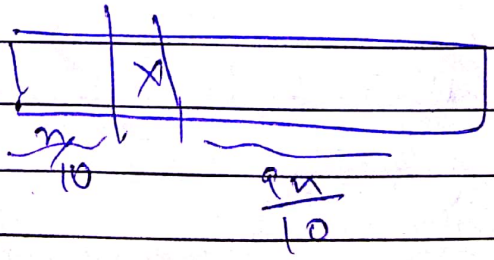
in best case $\rightarrow \boxed{\theta(n \log n)}$



$T(n) = 2T(n/2) + \theta(n)$

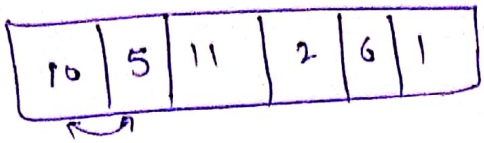
$= \theta(n \log n)$

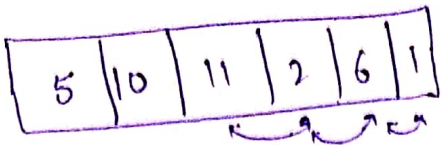
Almost best case $\rightarrow \boxed{\theta(n \log n)}$ Lucky case

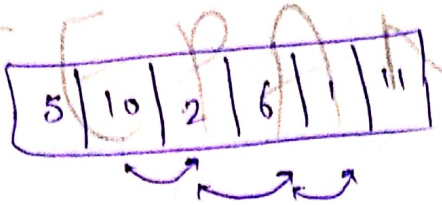


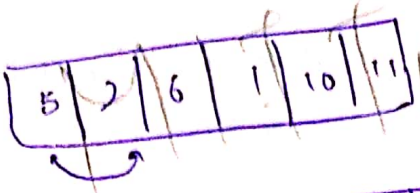
$T(n) = T(n/10) + T(9n/10) + cn$

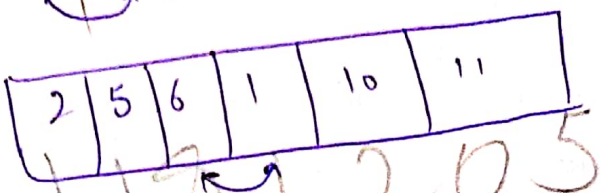
Bubble Sort

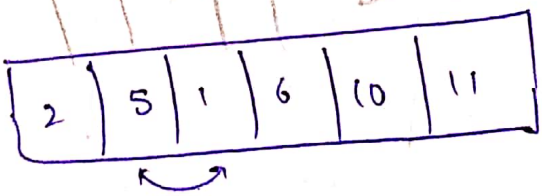


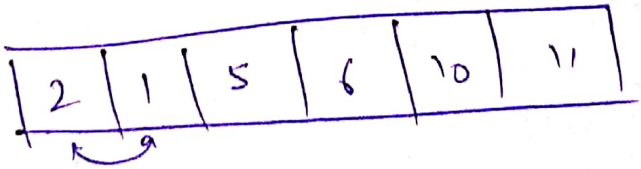


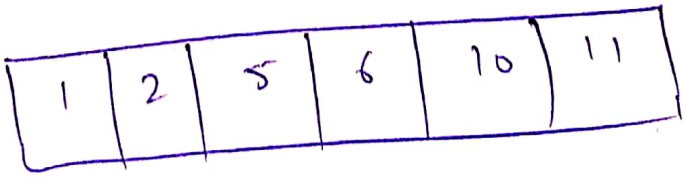












Sorted

Bubble sort is also inplace sorting algorithm since no extra space is required and changes are made in the input array itself.

	Quick sort	Bubble sort	Merge sort	Insertion sort
Category	In place	In place	not in place	In place
Time complexity				
Best case	$O(n \log n)$	$O(n)$	$O(n \log n)$	$O(n)$
Avg case	$O(n \log n)$	$O(n^2)$	$O(n \log n)$	$O(n^2)$
Worst case	$O(n^2)$	$O(n^2)$	$O(n \log n)$	$O(n^2)$
Preference	generally preferred for big data	not preferred due to excess time needed in it	Preferred for average size n	Preferred for small n or smaller data

To make a program to sort a given string using mergesort in increasing order,

make two functions

↳ merge (char A, int p, int q, int r)

↳ and mergeSort (char* A, int p, int r)

```
merge (char A[10], int p, int q, int r)
{
  // make two character arrays (strings) from the
  given array A,

     L[10]  of A[p] to A[q],  n1 = q-p+1

     R[10]  of A[q+1] to A[r],  n2 = r-q

  Now,  i=0,  j=0,  k=0;

  while ( i < n1  && j < n2)
  {
      if ( L[i] < R[j])
        A[k++] = L[i++];

      else
         A[k++] = R[j++];

  }

  while ( i < n1)                    // remaining elements of L
   A[k++] = L[i++]

  while (j < n2)                     // remaining elements of R
   A[k++] = R[j++]

}
```

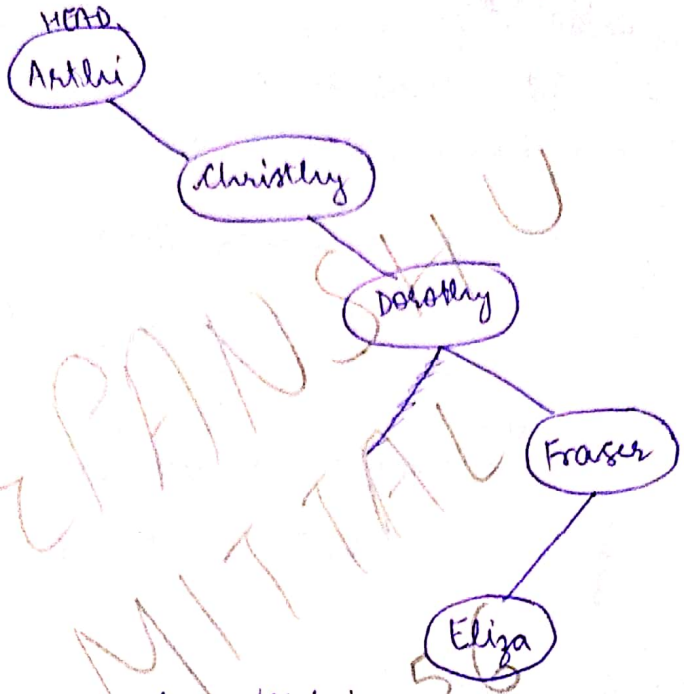
```
mergesort( char A[50], int p, int r)
{
    int q = (p+r)/2;
    //finding middle index.
  // Now, recursively call.
    mergesort(A, p, q);
    mergesort(A, q+1, r);

    merge(A, p, q, r);
}
```

Now in main function, call mergesort function with a string input and its size as arguments.

```
main()
{
   mergesort("Polynomial", 0, 10);
                A          p  r
   pf();
}
```

(3.) To make a program to construct a BST and insert a name into it.

BST will be made as :



The structure of the node will be :

```
struct node {
    char data[];
    struct node* left;
    struct node* right;
};
```

Write a function to make a new node

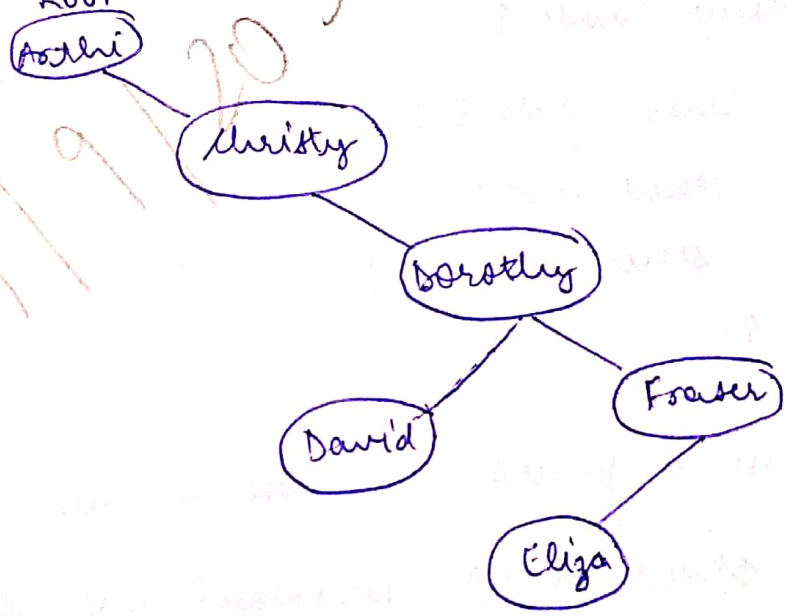
```
struct node * NewNode(char data[])
{
    struct node* new = (struct node*) malloc(sizeof(struct node));
    strcpy(new -> data, data);
        new -> left = NULL;
        new -> right = NULL;
    return new;
}
```

To insert 'David' in the given BST,

```
void insert ( struct node * t, char A[20] )
{
    if (!t)
    {t = New_Node (A);
     return; }
    else if ( t → data > A)
        insert ( t → left, A);
    else
        insert ( t → right, A);
}
```

// Recursively call insert func. to find and insert the new node as the required place.

The BST will become,



Preorder traversal will result in

Arthi Christy Dorothy David Fraser Eliza