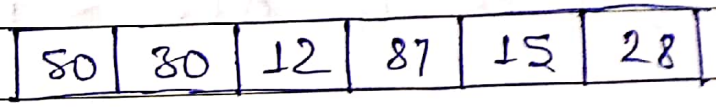
Date : ..... Page No. .....

2. Quick Sort & Bubble Sort

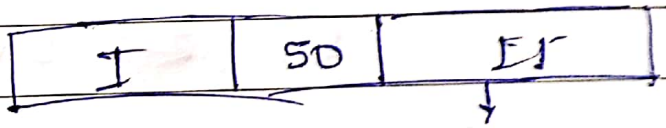
Quick sort is an algorithm of the divide and conquer type. That is, the problem of sorting a set is reduced to the problem of sorting two smaller sets.

↳ It is divide and conquer.

Working logic

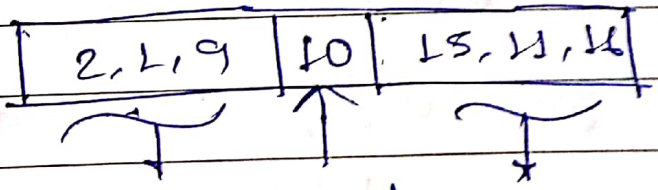


pivot = 50



Value < 50

Values > pivot



Same process

pivot

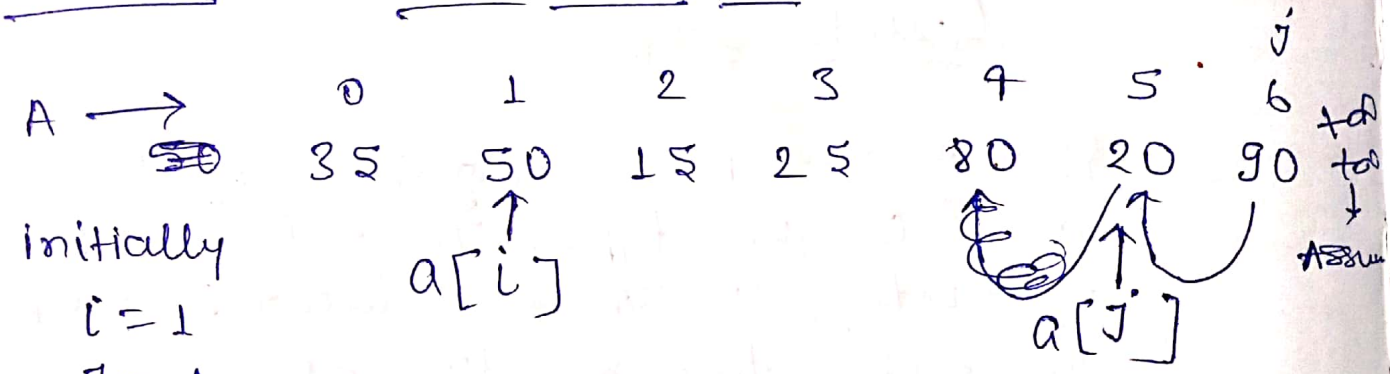
Same process for these data

for general case

TC → $T(n) = T(n/2) + T(n/2) + n$

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

Ex - 2 Explanation



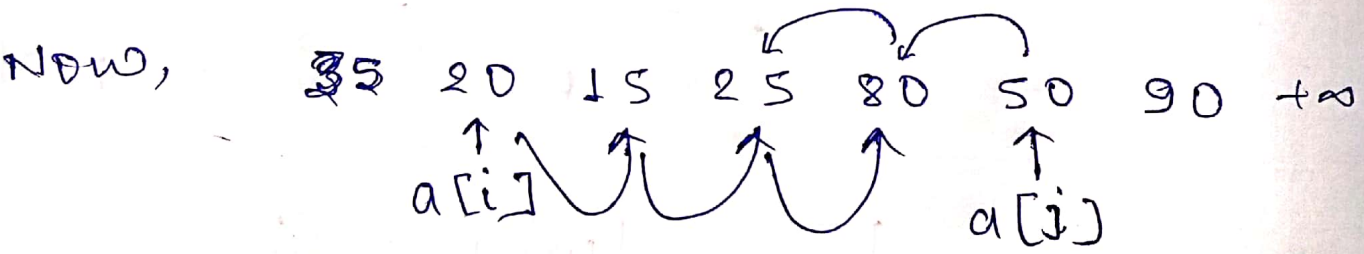
Initially
$i = 1$
$j = 6$

Check if $a[i] >$ pivot and increament of i
here $50 > 35 \rightarrow$ stop

Also check if $A[j] <$ pivot
$j--$;

Now Swap the ~~position~~ data
$20 \rightleftharpoons 50$

Now,



Check again $a[i] > 35$ & $a[j] < 35$
$i++$ $j--$

here, i & j cross then swap pivot with $a[j]$

part - I part - II

do same process for this part

do same process for this part.

~~1.~~ Program. Quick - Short

```
/* end --> ending index, start → starting index */
quick_sort (int a[], int start, int end)
{
   if (low < high)
    {
      pi = partition(a, start, end);
          // pi → partition index
     quick_sort (a, low, pi-1); // Before pi
     quick_sort (a, pi+1, end); // after pi
    }
}
partition (a[], start, end)
 {
 int pivot = a[start]
int i = start + 1 ; j = end ;
~~for ( i = start+1 ; i < end ; i++ )~~
while (i < j)
   {
      while (a[i] > a[pivot])
       i++ ;
      while (a[j] < a[pivot])
       j--;
     if (i < j) { swap (&a[i], &a[j]); }
} swap (&a[pivot], &a[j]); return j; }
```

Date ..... Page No. .....

Analysis of QuickSort

Time Complexity

* Worst case :- It occurs when the pivot element picked is either the greatest or the smallest element.

$$O(n^2)$$

* Best case :- It occurs when the pivot element is always the middle element or near to the middle element.

$$O(n \log n)$$

* Average case :- It occurs when above condition do not occur.

$$O(n \log n).$$

Space Complexity → $O(\log n)$.

Sorting

Bubble Sort

Ex:-

	0	1	2	3
A	35	15	29	8

↳ Logic starts with comparison of first two elements and if the left element is greater than right element, they swap their position. Comparison proceeds till the end of array.

Algorithm

BUBBLE_SORT (A, N) : A is array of values and N is the no. of elements.

1. Repeat for round = 1, 2, 3, ... N-1
2. Repeat for i = 0, 1, 2, ... N-1-round
3. If A[i] > A[i+1] then swap A[i] and A[i+1]
4. Return

Program Bubble — Sort.

```
int main ( )
{
   int A[ ] = { 34, 15, 29, 8 } ;
   int i ;
   bubble_sort (A, 4 ) ;
  for ( i = 0 ; i <= 3 ; i++ )

    printf ( " %d", A [i] ) ;
  return 0 ;
 }
Void bubble_Sort (int A[ ], int n)
 {
   int round , i, temp ;
  for (round = 1 ; round <= n-1 ;
                          round++ )
   for ( i = 0 ; i <= n-1-round ; i++ )
if( A[i] > A[i+1] )
 {
     temp = A[i ] ;
      A [i] = A [i+1] ;
      A[i+1] = temp ;
  }
}
```

Date : .....

Page No. .....

Time Complexity of bubble short.

Time Complexity = $O(n^2)$

How to reduce to doing some modification time Complexity.

↳ We want to skip the round if it already sorted.

Algorithm

MBUBBLE_SORT(A, N) : A is array of values and N is the no. of elements.

1. Repeat step 2,3,4 for round = 1, 2, 3, ... N-1.
2. flag = 0
3. Repeat for i = 0, 1, 2, ... N-1-round

   if A[i] > A[i+1] then swap A[i] and A[i+1], also set flag = 1
4. If flag == 0 return
5. Return.

Program

```
int main( )
{
  int A[ ] = {34, 11, 4, 56, 9};
  int i;
  bubble_sort(A, 9);
  for ( i=0; i<=8; i++ )
    printf ("%d", A[i]);
  return 0;
}
Void bubble_sort ( int A[ ], int N )
{
  int round, i, temp, flag;
  for( round = 1; round <= N-1; round++ )
  {
    flag = 0;
      for ( i=0; i<= N-1-round; i++)
    if ( A[i] > A[i+1])
    {
      temp = A[i];
      flag = 1;
      temp = A[i];
      A[i] = A[i+1];
      A[i+1] = temp;
    }
```

Date : .....

Page No. .....

```
if (flag == 0)
{
    printf ("Round = %d \n" round);
    return ;
}
}
```

∴ for best case, round = 0

for worst case,

↓

when array is reverse sorted.

$O(n^2)$

Best case :- $O(n)$

↳ when array is already sorted.

Average case :- $O(n^2)$.