UNIT- 2 : DIVIDE AND CONQUER

UNIT STRUCTURE

2.1 LEARNING OBJECTIVE

After going through this unit, you will be able to:

- know the concept divide and conquer
- describe the application of divide and conquer method in binary search, quick sort and merge sort techniques
- elaborate the application of divide and conquer technique in exponentiation

2.2 INTRODUCTION

In the previous unit, you are acquainted with the basic idea about the algorithms, time complexity of algorithms and some other related issues. In this unit, we will introduce you the **‘*divide and***

conquer’ approach that is used in the design of algorithms. This technique is the basis of designing efficient algorithms for all kinds of problems, such as sorting techniques like quick sort, merge sort and in searching techniques like binary search etc.

2.3 DIVIDE AND CONQUER GENERALSTRATEGY

Divide and conquer algorithm is an important algorithm designed paradigm based on multi-branched recursion. A divide and conquer algorithm works by recursively breaking down a problem into two or more independent sub-problems of the same type, until they become simple enough to solved directly. Generally, the sub-problems solved by a divide and conquer is non-overlapping i.e solution to a problem is depend upon only on sub-problems, but is not depend upon sub-sub-problems.
The general methodology applied in the divide and conquer technique is as follows :

Step 1: Divide the problem into two or more independent sub-Problems (not necessarily same type).
Step 2: Solve (conquer) the each sub-problem recursively to the Smallest possible size.
Step 3: Combine these solution of the sub-problems into a solution to the whole problem.

2.4 DIVIDE AND CONQUER ALGORITHM APPLIED IN BINARY SEARCH

Binary search is a well known instance of divide and conquer method. For binary search divide and conquer strategy is applied recursively for a given sorted array is as follows:

Divide: Divide the selected array at the middle. It creates two sub-array, one left sub-array and other right sub-array.

Conquer: Find out the appropriate sub-array.

Combine: Check for the solution to key element.

For a given sorted array of N element and for a given key element (value to be searched in the sorted array), the basic idea of binary search is as fallows –

1. First find the middle element of the array

2. Compare the middle element with the key element.

3. There are three cases

- If it is the key element then search is successful.
- If it is less than key element then search only the lower half of the array.
- If it is greater than key element then search only the upper half of the array.

4. Repeat 1, 2 and 3 until the key element found or sub-array sizes become one.

Algorithm for binary search

```
1.    Set Lower = 0, Upper = N -1
2.    Mid = ( Lower + Upper ) / 2
3.    while ( Lower ≤Upper) and A [ Mid ] != Item repeat
      Steps 4 and 5
4.    if ( Item < A [ Mid ] )  then
5.    Upper = Mid -1
6.    else   Lower = Mid +1
7.    Mid = ( Lower + Upper ) / 2
8.    if ( A [ Mid ] = = Item ) then
9.    Print  "Search successful"
10.   else   Print  "Item is not found"
11.   end
```

Here, **Lower, Upper and Mid** denotes the beginning, ending and middle index of an array A [ ] respectively. Item means the key element to be searched in the given array A [ ]. If size of A [ ] is N then beginning and the ending indices are 0 and N -1 respectively.

In ***step 1*** the algorithm initially sets the value of Lower = 0 and Upper = N -1.

In ***step 2*** it calculate Mid, the index of the middle element, for the array A [ ].

In ***step 3*** while the beginning index (Lower) is less then end index (Upper) and middle element (A [Mid]) is not equal key element (Item) then repeat step 4 and 5.

In ***step 4*** if A [Mid] is less than the Item, then the algorithm searches in the left sub-array. So, the beginning index remain same and the end index of the left sub-array becomes Mid -1. Hence, 'Upper' is set as Mid -1. Else if A [Mid] is greater than Item, then the algorithm searches in the right sub-array. So, the beginning index of the right sub-array becomes Mid+1 and end index remain same. Hence the 'Lower' is set as Mid+1.

In ***step 5*** again middle element (Mid) is calculated for the selected sub-array in step 4.

In ***step 6*** algorithm is terminated either if middle element (A[Mid]) is equal to Item or beginning index (Lower) is greater than end index (Upper)(i.e when subarray sizes become one) . In first case , terminate when search element is found and in later case terminate when search is not successful.

Example 2.5.1: Suppose A is an array of 6 elements. Search an element 10 in the array using binary search.

0	1	2	3	4	5
3	4	7	10	12	13

Solution:

Step 1: Here Lower = 0, Upper = 5, Item = 10

Step 2: First we have to calculate middle element for the array A

Mid = (Lower + Upper) / 2
= (0 + 5) / 2
=2

Mid *divides* the array into two subarray as follows

Lower= 0	1	Mid = 2	3	4	Upper=5
3	4	7	10	12	13

Step 3: Here *Lower* < *Upper* and *A [Mid] ! =10*. So *continue Step4 and Step 5*

Step 4: Select *(Conquer)* an appropriate subarray.

Here A [Mid] =7

7<10 , select *the right subarray* to search for the element.

Lower = Mid+1
=2 +1
=3

Now the subarray is-

Lower = 3	4	Upper = 5
10	12	13

Step 5: Mid = (Lower + Upper) / 2
= (3 + 5) / 2
= 4

Lower = 3	Mid = 4	Upper = 5
10	12	13

divide the array again in Mid = 4.

Here, *Lower* < *Upper* and *A [Mid] != 10* So, *repeat step 4 and Step 5* again

A [4] =12

12 >10, So search in the *left subarray.*

Upper = Mid -1

= 4 -1

=3

New subarray is-

Lower=3 Upper=3

10

Step 6: Mid = (Lower + Upper) / 2

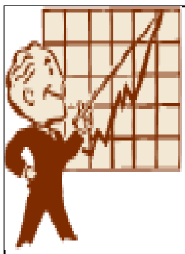
= (3 + 3) / 2

= 3

Here, *Lower* = *Upper* and *A [Mid] = =10*

So, *PRINT "Search successful"*

Step 7: End



CHECK YOUR PROGRESS

1. Fill in the blanks

a) Binary search is applied in already------------array.

b) In binary search each time the algorithm finds out the -------- element.

c) The algorithm divide the array into two halves in ---------.

d) Divide and conquer algorithm is applied in a problem when sub-problems are------.

e) Divide and conquer algorithm divide the problem into -------- sub-problems.

2.5 DIVIDE AND CONQUER ALGORITHM APPLIED IN MERGE SORT

Merge sort is also one of the ‘divide and conquer’ class of algorithms. This is a sorting algorithm to sort an unordered list of element. Merge sort is a recursive algorithm that splits the array into two sub-arrays, sorts each sub-array, and then merges the two sorted arrays into a single sorted array. The base case of the recursion is when a subarray of size 1 (or 0). Merge sort algorithm also closely follow divide and conquer strategy. It is an external sorting algorithm.

Divide : Divide N element array to be sorted into two subarray of N / 2 element each.

Conquer : Sort the subarrays recursively using merge sort.

Combine : Merge the two sorted sub-array to produce final sorted array.

Suppose we have to sort a array of N element, A [p…..r]. Initially p = 1 and r=N

To sort A [p .. r]

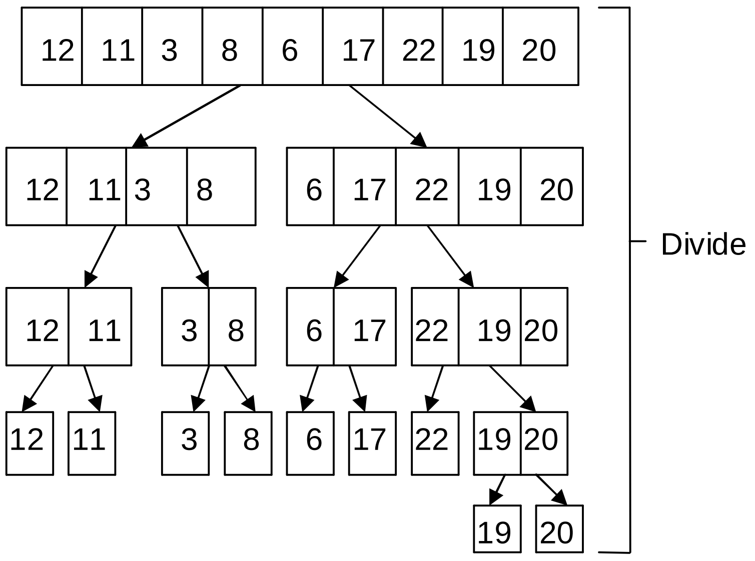
1. ***Divide Step:*** If a given array *A* has zero or one element, simply return; it is already sorted. Otherwise, split *A* [p .. r] into two subarrays *A* [p .. q] and *A* [q + 1 .. r], each containing about half of the elements of *A* [p .. r]. That is, *q* is the halfway point of *A* [p .. r].
2. ***Conquer Step*:** Conquer by recursively sorting the two subarrays *A* [p .. q] and A [q + 1 .. r].
3. ***Combine Step*:** Combine the elements back in A [p .. r] by merging the two sorted subarrays A [p .. q] and *A* [*q* + 1 .. r] into a unique sorted sequence.

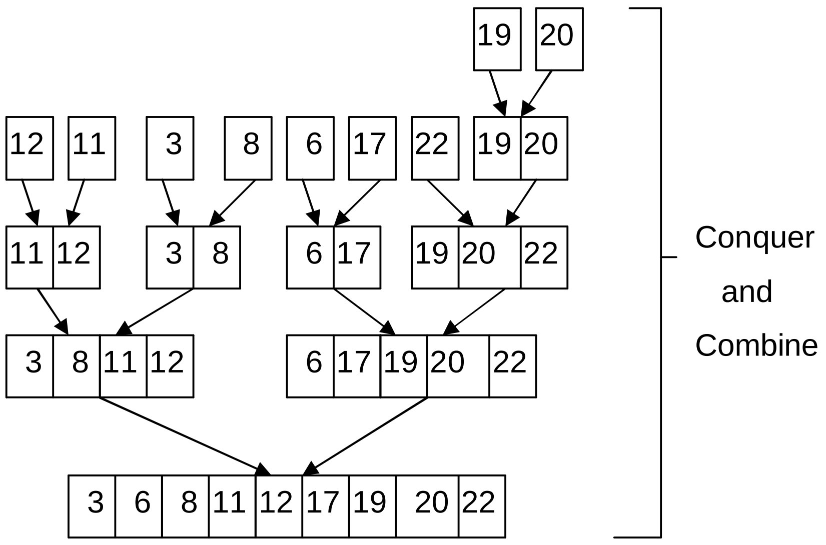
Example 2.6.1: Sort the following data using merge sort

12	11	3	8	6	17	22	19	20

Solution:

The merge sort strategy is applied as fallows-





This is the final sorted array.

2.7 PSEUDO CODE FOR MERGE SORT ALGORITHM

```
MERGE_SORT ( A , p , r)
  1. if ( p < r )
  2. then q = ( p + r ) / 2
  3.           MERGE_SORT ( A , p , q )
  4.           MERGE_SORT ( A , q + 1 , r )
  5.           MERGE ( A , p , q , r )
```

```
MERGE ( A , p , q , r )
  1. $n_1 = q – p + 1$
  2. $n_2 = r - q$
  3. create arrays L [ 1.....$n_1$ + 1 ]  and  R [ 1......$n_2$ + 1 ]
  4. for i=1 to $n_1$
  5.        do   L [ i ] = A [ p + i – 1 ]
  6. for j=1 to $n_2$
  7.        do   R [ j ] = A [ q + j ]
  8. L [ $n_1$ +1 ] = $\infty$
  9.R [ $n_2$ + 1 ] = $\infty$
  10. i =1
  11. j =1
  12. for k = p to r
  13.      do if L [ i ] $\le$ R [ i ]
  14.          then  A [ k ] = L [ i ]
  15.          i = i +1
  16.      else   A [ k ] = R [ j ]
  17.          j = j + 1
```

Here the procedure MERGE_SORT (A , p , r) sorts the element in the subarray A [p….r] . i.e p is the first element index and r is the last element index of the subarray. If $p \ge r$, the subarray is atmost one element and is already sorted. Otherwise, the divide

step (step 2 in MERGE_SORT (A, p, r) procedure) simply computes an index q that partition A [p….r] into two subarray A [p …..q] and A [q + 1…..r] containing n/2 elements in each subarray
To sort a sequence A of N element , the initial call is MERGE_SORT(A,1, N).
Next, we have to merge the sorted subarrays obtain from the MERGE_SORT (A , p , r) procedure using MERGE(A , p , q , r), where A is an sorted array, p , q and r indices of the element such that $p \le q < r$. This procedure merge two sorted sub-array A[p..q] and A [q + 1….r] and form a single sorted subarray and replaces the current subarray A [p..r].

In details the MERGE procedure is work as follows-

Line 1 compute the length n_1 of the subarray A [p…q] .
Line 2 compute the length n_2 of the subarray A [q + 1…r].
Line 3 create array 'L' (left) and 'R' (right) of length n_1+1 and n_2+1 respectively.
Line 4-5 the for loop copies the subarray A [p…q] into L [1..n_1].
Line 6-7 it copies the subarray A [q + 1…r] into R [1..n_2].
Line 8-9 put '∞' at the end of the array L (i.e in n_1+1)and R (i.e in n_2 + 1).
Line12-17 find the smallest element between L[i] and R [j], where i =1….n_1 and j =1….n_2.

- If L [i] ≤ R [j] then it copies L [i] to A and increase i to i +1,
- Otherwise, copies R [j] to A and increase j to j +1.

Example:

Let us see, how merging is done between two sorted subarrays using the MERGE procedure. After merging is done the subarrays are combined to one sorted array.

In the following sequence the procedure calls MERGE(A,13,15,17) works as below .

Here p = 13
q = 15
r = 17
L= left subarray
R= right subarray
i = index of element's of left subarray L
j = index of element's of right subarray R

A	k=13	14	15	16	17	
....	10	11	14	9	13	…

L i=1	2	3	4
10	11	14	∞

R j=1	2	3
9	13	∞

Here, k = 13, i =1 and j = 1
R [1] < L [1] i.e 9 < 10.
So, A [1] = R [1]
= 9
j = j + 1 =1 + 1 = 2

A	k=13	14	15	16	17	
…	9	11	14	9	13	…

Next, k = 14, i = 1, j = 2

L [1] < R [2] i.e 10 < 13

So, A [14] = L [1]

=10

i = i + 1 = 1 + 1 = 2

A	13	k=14	15	16	17	
…	9	10	14	9	13	…

Next, k = 15, i = 2, j = 2

L [2] < R [2], i.e 11 < 13

So, A [15] = L [2]

=11

i = i + 1 = 2 + 1 = 3

A	13	14	k=15	16	17	
…	9	10	11	9	13	…

Next, k = 16 , i = 3, j = 2

R [2] < L [3] , i.e 13 < 14

So, A [16] = R [2]

=13

j = j + 1 = 2 + 1 = 3

A	13	14	15	k = 16	17	
…	9	10	11	13	13	…

Next, k = 17, i = 3, j = 3

L [3] < R [3], i.e 14 < ∞

So, A [17] = L [3]

=14

i = i + 1 = 3 + 1 = 4

A	13	14	15	16	k = 17	
…	9	10	11	13	14	…

Next, k=18 which is greater then r(i.e 17). So the MERGE procedure is terminated here.

CHECK YOUR PROGRESS

2. Fill in the blanks

a. In divide step merge sort algorithm divides the array elements to be sorted up to array size becomes ---- or -----.

b. Merge sort algorithm merges two ----- subarrays.

c. For merging two subarrays merge sort algorithm uses another ----- subarray.

d. Merge sort algorithm is an-------sorting algorithm.

2.7 DIVIDE AND CONQUER ALGORITHM APPLIED IN QUICK SORT

It is one of the widely used internal sorting algorithm. In its basic form it was developed by C.A.R Hoare in 1960. The basis of quick sort is divide and conquer strategy, i.e divide the problem (list to be sorted) into sub-problems (sub-lists) , until solved sub-problems (sorted sub-list) are found. The divide and conquer approach can be used in quick sort differently from merge sort. In merge sort , the list to be sorted is divided at its midpoint into subarrays which are independently sorted and later merged. In quick sort, the division to the sorted subarrays is made, so that the sorted subarrays do not need to merge later.

Divide and conquer strategy for quick sort to sort an array A [p…r] is as follows:

Divide: partition the array A [p…r] into two sub-arrays A[p…q -1] and A [q + 1…..r] such that each element of A [p…q -1] is less than or equal to A [q], which in turn, less than or equal to each element of A [q + 1…r]. Compute the index q as part of this partitioning procedure.

Conquer: Sort the two subarrays A[p…q-1] and A[q+1…r] by recursive calls to quick sort .

Combine: Since the subarrays are sorted in place, no work is needed to combine them. The entire array A [p…r] is now sorted.

2.7.1 ALGORITHM FOR QUICK SORT

Quick_Sort **:**

```
Step 1 : If First < Last then begin      /* here First and Last are the
                                            index of the first and last
```

elements in the array*/

Step 2 : Partition the elements in the subarray *First….Last*

Step 3 : Apply ***Quick_Sort*** to the first subarray.

Step 4 : Apply ***Quick_Sort*** to the second subarray.

end

For this algorithm two stopping cases are-

- If *First* = *Last* . i.e. only one element in the subarray to be sorted.
- if *First* > *Last*. i.e. no element in the subarray to be sorted.

Partition algorithm**:**

To partition an array A[ ] partition algorithm is

Step 1: Define the *Pivot* value as the contents of the Array, A [*First*].

Step 2: Initialize *Up* to the *First* and *Down* to the *Last*

Step 3: Repeat step 4,5,6 until *Up* ≥ *Down*

Step 4: Increment Up until Up selects the first element greater than the *Pivot* value.

Step 5: Decrement *Down* until it selects the first element less than or equal to the *Pivot* value.

Step 6: If *Up* < *Down* exchange their values.

Step 7: Exchange A [*First*] and A [*Down*].

Step 8: Define *PivotIndex* as *Down*

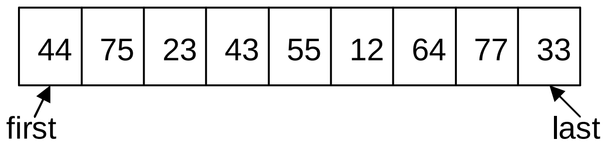
Example :

Sort the following data of array A using quick sort.

A = 44 75 23 43 55 12 64 77 33

Solution:

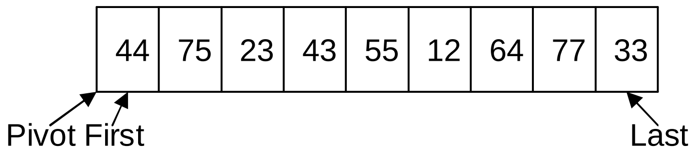
1. Assign *First* to first element and *Last* to last element.



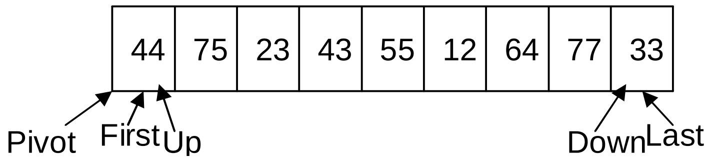
2. First < Last,

   So, assign Pivot to First .

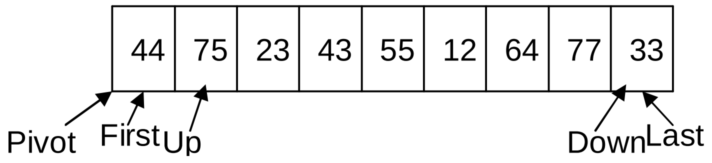
   Pivot = First



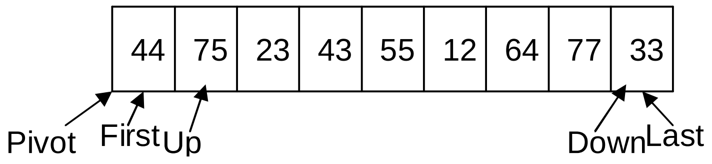
3. Assign Up to first element and Down to last element.



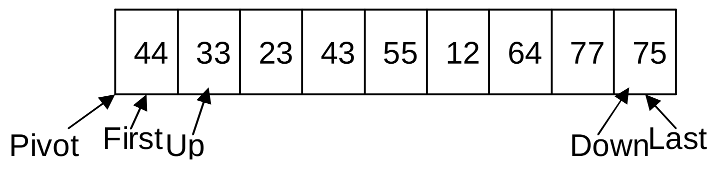
a) if (A [Pivot] ≥ A [Up]) then Up++



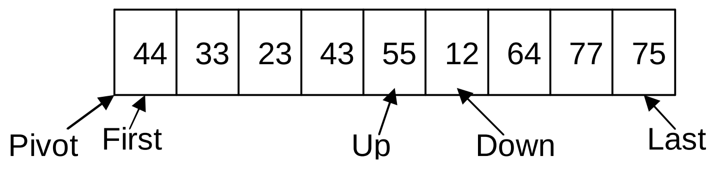
b) if (A [Pivot] < A [Down]) Down --

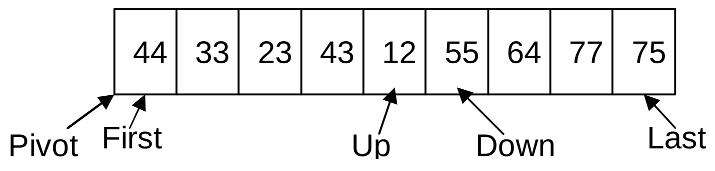


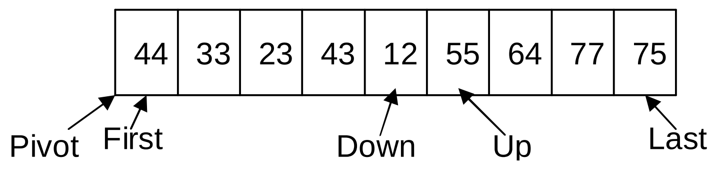
c) if (Up < Down) exchange A [Up] and A [Down]



d) Repeat a), b) and c) if Up < Down

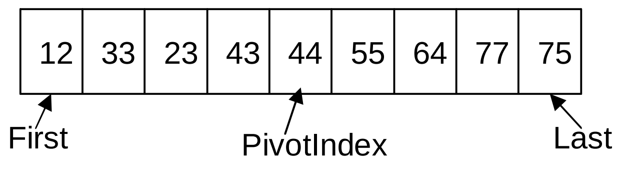




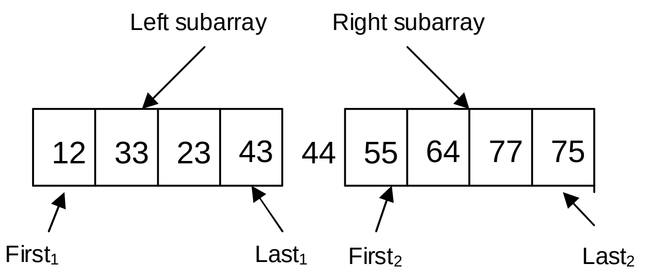


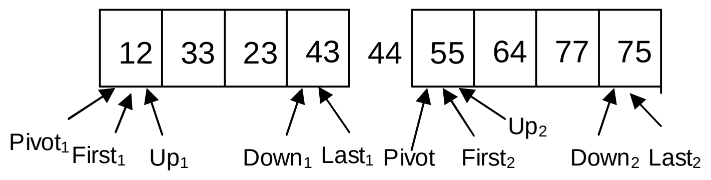
e) Here, Up > Down

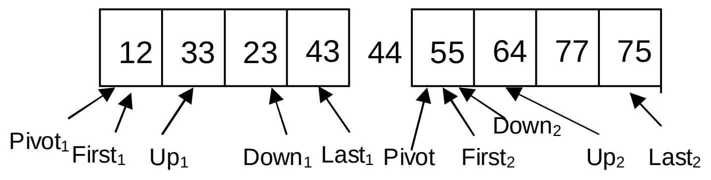
Exchange A [Pivot] and A [Down] and assign Down as PivotIndex

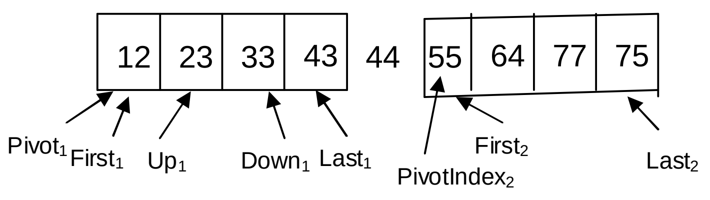


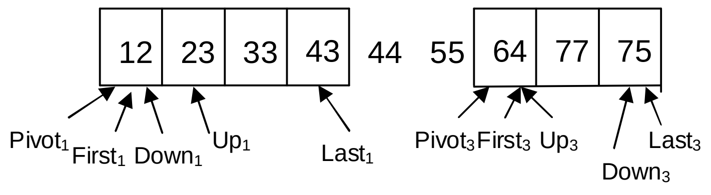
4. This gives two subarrays , Left subarray and Right subarray. Again we have to apply Quick Sort procedure in these sub arrays to sort it.

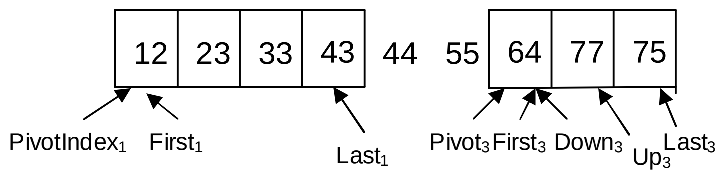


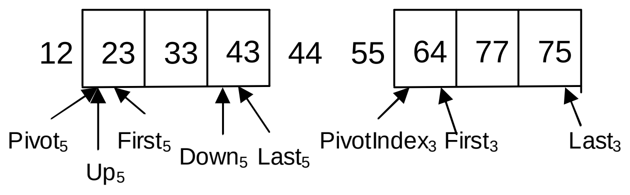


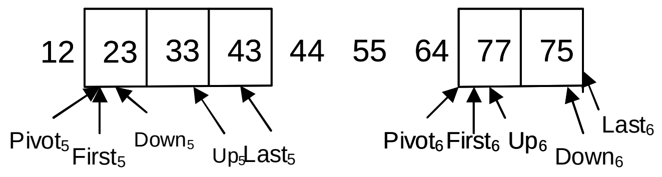


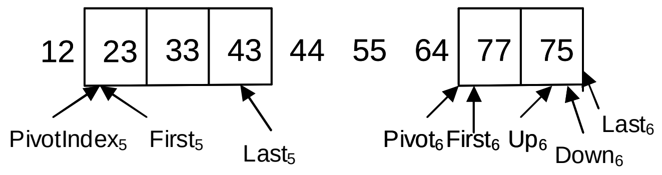


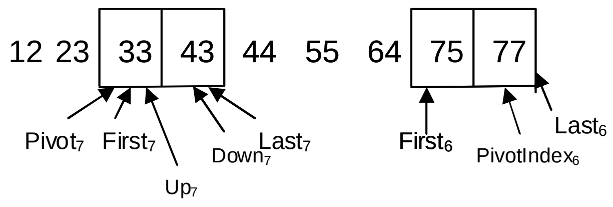


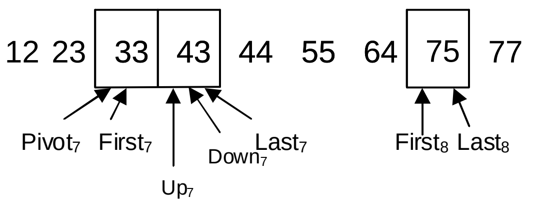

12 23 33 43 44 55 64 77 75
PivotIndex₁
First₁
Last₁
Pivot₃
First₃
Down₃
Up₃
Last₃

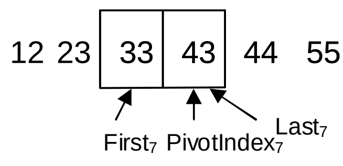

12 23 33 43 44 55 64 77 75
Pivot₅
Up₅
First₅
Down₅
Last₅
PivotIndex₃
First₃
Last₃

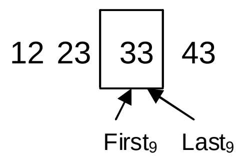

12 23 33 43 44 55 64 77 75
Pivot₅
First₅
Down₅
Up₅
Last₅
Pivot₆
First₆
Up₆
Down₆
Last₆

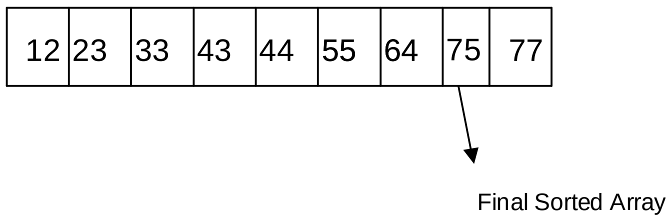

12 23 33 43 44 55 64 77 75
PivotIndex₅
First₅
Last₅
Pivot₆
First₆
Up₆
Down₆
Last₆


12 23 33 43 44 55 64 75 77
Pivot₇
First₇
Up₇
Down₇
Last₇
First₆
PivotIndex₆
Last₆


12 23 33 43 44 55 64 75 77
Pivot₇
First₇
Up₇
Down₇
Last₇
First₈
Last₈







2.7.2 PSEUDO CODE FOR QUICK SORT

The pseudo code for the quick sort algorithm is given below :

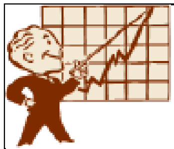
```
QUICKSORT ( A , p , r)
1.    if p < r
2.     then q = PARTITION ( A , p , r )
3.         QUICKSORT  ( A , p , q – 1 )
4.            QUICKSORT ( A ,q + 1 , r )
```

To sort an entire array A, the initial call is
QUICKSORT (A ,1, length [A])

Next, the PARTITIONING procedure which rearranges the subarray A [p….r] in place.

```
PARTITION (A , p , r)
```

```
1.    x = A [ r ]
2.    i = p -1
3.    for j = p to r -1
4.      do if A [ j ] ≤ x
5.          then i = i +1
6.              exchange A [ i ] = A [ j ]
7.    exchange A [ i + 1 ] = A [ r ]
8.    return i + 1
```



CHECK YOUR PROGRESS

3. What is the type of quick sort algorithm? External or internal?
4. Why does not quick sort algorithm need to combine the sorted subarrays later?