Tulsiramji Gaikwad-Patil College of Engineering and Technology
Wardha Road, Nagpur-441 108
NAAC Accredited

Department of Computer Science & Engineering

UNIT 1 Notes

Subject: Data Structures & Program Design **Year/Sem: 2nd yr./4th sem**

Q1. What are the parameters & their roles considered for Analysis of an algorithm? Explain asymptotic notations used for analysis of algorithms. 8M (W-17)

Ans: Efficiency of an algorithm can be analyzed at two different stages, before implementation and after implementation. They are the following −

- **A Priori Analysis** − This is a theoretical analysis of an algorithm. Efficiency of an algorithm is measured by assuming that all other factors, for example, processor speed, are constant and have no effect on the implementation.
- **A Posterior Analysis** − This is an empirical analysis of an algorithm. The selected algorithm is implemented using programming language. This is then executed on target computer machine. In this analysis, actual statistics like running time and space required, are collected.

We shall learn about *a priori* algorithm analysis. Algorithm analysis deals with the execution or running time of various operations involved. The running time of an operation can be defined as the number of computer instructions executed per operation.

Algorithm Complexity

Suppose **X** is an algorithm and **n** is the size of input data, the time and space used by the algorithm X are the two main factors, which decide the efficiency of X.

- **Time Factor** − Time is measured by counting the number of key operations such as comparisons in the sorting algorithm.
- **Space Factor** − Space is measured by counting the maximum memory space required by the algorithm.

The complexity of an algorithm **f(n)** gives the running time and/or the storage space required by the algorithm in terms of **n** as the size of input data.

Tulsiramji Gaikwad-Patil College of Engineering and Technology
Wardha Road, Nagpur-441 108
NAAC Accredited

Department of Computer Science & Engineering

Space Complexity

Space complexity of an algorithm represents the amount of memory space required by the algorithm in its life cycle. The space required by an algorithm is equal to the sum of the following two components −

- A fixed part that is a space required to store certain data and variables, that are independent of the size of the problem. For example, simple variables and constants used, program size, etc.
- A variable part is a space required by variables, whose size depends on the size of the problem. For example, dynamic memory allocation, recursion stack space, etc.

Space complexity S(P) of any algorithm P is S(P) = C + SP(I), where C is the fixed part and S(I) is the variable part of the algorithm, which depends on instance characteristic I. Following is a simple example that tries to explain the concept −

```
Algorithm: SUM(A, B)
Step 1 -  START
Step 2 -  C ← A + B + 10
Step 3 -  Stop
```

Here we have three variables A, B, and C and one constant. Hence S(P) = 1 + 3. Now, space depends on data types of given variables and constant types and it will be multiplied accordingly.

Time Complexity

Time complexity of an algorithm represents the amount of time required by the algorithm to run to completion. Time requirements can be defined as a numerical function T(n), where T(n) can be measured as the number of steps, provided each step consumes constant time.

For example, addition of two n-bit integers takes **n** steps. Consequently, the total computational time is T(n) = c ∗ n, where c is the time taken for the addition of two bits. Here, we observe that T(n) grows linearly as the input size increases.

Tulsiramji Gaikwad-Patil College of Engineering and Technology
Wardha Road, Nagpur-441 108
NAAC Accredited

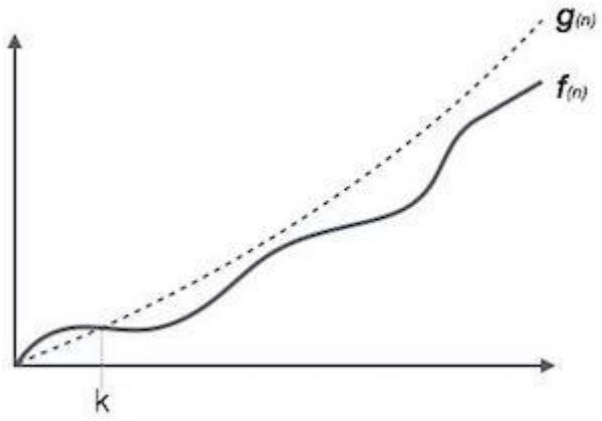
Department of Computer Science & Engineering

Following are the commonly used asymptotic notations to calculate the running time complexity of an algorithm.

- O Notation
- Ω Notation
- θ Notation

Big Oh Notation, O

The notation O(n) is the formal way to express the upper bound of an algorithm's running time. It measures the worst case time complexity or the longest amount of time an algorithm can possibly take to complete.



For example, for a function ***f*(n)**

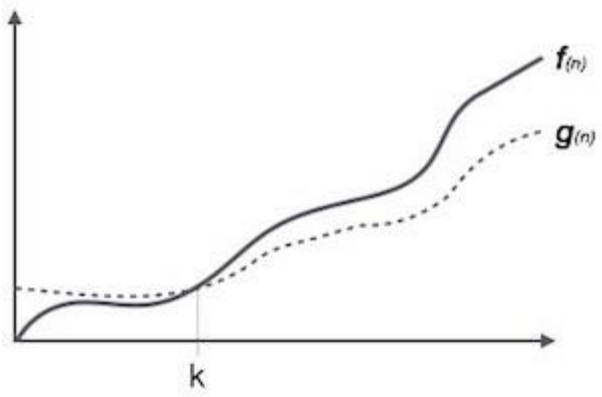
$O(f(n)) = \{ g(n) : \text{there exists } c > 0 \text{ and } n_0 \text{ such that } f(n) \leq c.g(n) \text{ for all } n > n_0. \}$

Omega Notation, Ω

The notation Ω(n) is the formal way to express the lower bound of an algorithm's running time. It measures the best case time complexity or the best amount of time an algorithm can possibly take to complete.

Tulsiramji Gaikwad-Patil College of Engineering and Technology
Wardha Road, Nagpur-441 108
NAAC Accredited

Department of Computer Science & Engineering

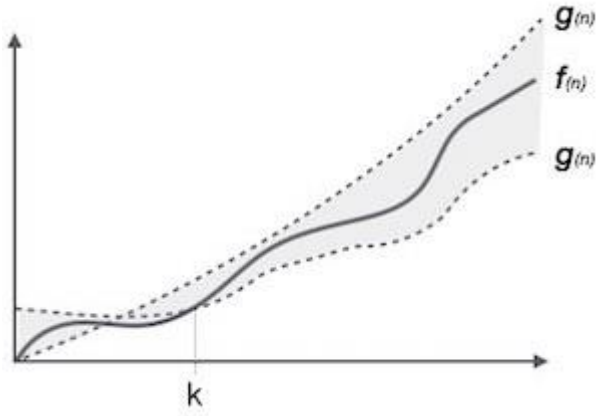


For example, for a function ***f*(n)**

$\Omega(f(n)) \geq \{ g(n)$: there exists $c > 0$ and n_0 such that $g(n) \leq c.f(n)$ for all $n > n_0$. $\}$

Theta Notation, θ

The notation θ(n) is the formal way to express both the lower bound and the upper bound of an algorithm's running time. It is represented as follows −



$\theta(f(n)) = \{ g(n)$ if and only if $g(n) = O(f(n))$ and $g(n) = \Omega(f(n))$ for all $n > n_0$. $\}$

Common Asymptotic Notations

Following is a list of some common asymptotic notations −

constant	–	O(1)
logarithmic	–	O(log n)

Tulsiramji Gaikwad-Patil College of Engineering and Technology
Wardha Road, Nagpur-441 108
NAAC Accredited

Department of Computer Science & Engineering

linear	–	O(n)
n log n	–	O(n log n)
quadratic	–	$O(n^2)$
cubic	–	$O(n^3)$
polynomial	–	$n^{O(1)}$
exponential	–	$2^{O(n)}$

Q2. How to decide performance of an algorithm? Explain big O notation in brief. **5M**
(S-18,S17,W-16)

Ans: Efficiency of an algorithm can be analyzed at two different stages, before implementation and after implementation. They are the following −

- ***A Priori* Analysis** − This is a theoretical analysis of an algorithm. Efficiency of an algorithm is measured by assuming that all other factors, for example, processor speed, are constant and have no effect on the implementation.
- ***A Posterior* Analysis** − This is an empirical analysis of an algorithm. The selected algorithm is implemented using programming language. This is then executed on target computer machine. In this analysis, actual statistics like running time and space required, are collected.

We shall learn about *a priori* algorithm analysis. Algorithm analysis deals with the execution or running time of various operations involved. The running time of an operation can be defined as the number of computer instructions executed per operation.

Algorithm Complexity

Suppose **X** is an algorithm and **n** is the size of input data, the time and space used by the algorithm X are the two main factors, which decide the efficiency of X.

Tulsiramji Gaikwad-Patil College of Engineering and Technology
Wardha Road, Nagpur-441 108
NAAC Accredited

Department of Computer Science & Engineering

- **Time Factor** − Time is measured by counting the number of key operations such as comparisons in the sorting algorithm.
- **Space Factor** − Space is measured by counting the maximum memory space required by the algorithm.

The complexity of an algorithm **f(n)** gives the running time and/or the storage space required by the algorithm in terms of **n** as the size of input data.

Space Complexity

Space complexity of an algorithm represents the amount of memory space required by the algorithm in its life cycle. The space required by an algorithm is equal to the sum of the following two components −

- A fixed part that is a space required to store certain data and variables, that are independent of the size of the problem. For example, simple variables and constants used, program size, etc.
- A variable part is a space required by variables, whose size depends on the size of the problem. For example, dynamic memory allocation, recursion stack space, etc.

Space complexity S(P) of any algorithm P is S(P) = C + SP(I), where C is the fixed part and S(I) is the variable part of the algorithm, which depends on instance characteristic I. Following is a simple example that tries to explain the concept −

```
Algorithm: SUM(A, B)
Step 1 -  START
Step 2 -  C ← A + B + 10
Step 3 -  Stop
```

Here we have three variables A, B, and C and one constant. Hence S(P) = 1 + 3. Now, space depends on data types of given variables and constant types and it will be multiplied accordingly.

Time Complexity

Tulsiramji Gaikwad-Patil College of Engineering and Technology
Wardha Road, Nagpur-441 108
NAAC Accredited

Department of Computer Science & Engineering

Time complexity of an algorithm represents the amount of time required by the algorithm to run to completion. Time requirements can be defined as a numerical function T(n), where T(n) can be measured as the number of steps, provided each step consumes constant time.

For example, addition of two n-bit integers takes **n** steps. Consequently, the total computational time is T(n) = c ∗ n, where c is the time taken for the addition of two bits. Here, we observe that T(n) grows linearly as the input size increases.

Big O Notation: The Big O notation defines an upper bound of an algorithm, it bounds a function only from above. For example, consider the case of Insertion Sort. It takes linear time in best case and quadratic time in worst case. We can safely say that the time complexity of Insertion sort is O(n^2). Note that O(n^2) also covers linear time.
If we use Θ notation to represent time complexity of Insertion sort, we have to use two statements for best and worst cases:
1. The worst case time complexity of Insertion Sort is Θ(n^2).
2. The best case time complexity of Insertion Sort is Θ(n).
The Big O notation is useful when we only have upper bound on time complexity of an algorithm. Many times we easily find an upper bound by simply looking at the algorithm.

O(g(n)) = { f(n): there exist positive constants c and

n0 such that 0 <= f(n) <= c*g(n) for

all n >= n0}

Q3. Explain the Stability of an Algorithm for sorting. Give an example. **4M (S-18)**
Ans: A sorting algorithm is said to be stable if two objects with equal keys appear in the same order in sorted output as they appear in the input array to be sorted.

Formally stability may be defined as,
Let be an array, and let be a strict weak ordering on the elements of A sorting algorithm is stable if- where is the sorting permutation (sorting moves to position)
Informally, stability means that equivalent elements retain their relative positions, after sorting.

When equal elements are indistinguishable, such as with integers, or more generally, any data where the entire element is the key, stability is not an issue. Stability is also not an issue if all keys are different.

Tulsiramji Gaikwad-Patil College of Engineering and Technology
Wardha Road, Nagpur-441 108
NAAC Accredited

Department of Computer Science & Engineering

An example where it is useful
Consider the following dataset of Student Names and their respective class sections.

If we sort this data according to name only, then it is highly unlikely that the resulting dataset will be grouped according to sections as well.

Q4. Write an function to implement heap sort. 6M (S-18,W-16)
Ans: Heaps can be used in sorting an array. In max-heaps, maximum element will always be at the root. Heap Sort uses this property of heap to sort the array.
Consider an array *Arr* which is to be sorted using Heap Sort.

- Initially build a max heap of elements in *Arr*.
- The root element, that is *Arr*[1], will contain maximum element of *Arr*. After that, swap this element with the last element of *Arr* and heapify the max heap excluding the last element which is already in its correct position and then decrease the length of heap by one.
- Repeat the step 2, until all the elements are in their correct position

```
void heap_sort(int Arr[ ])

    {
        int heap_size = N;
        build_maxheap(Arr);
        for(int i = N; i >= 2 ; i-- )
        {
            swap|(Arr[ 1 ], Arr[ i ]);
            heap_size = heap_size - 1;
            max_heapify(Arr, 1, heap_size);
        }
    }
```

Complexity Analysis of Heap Sort
Worst Case Time Complexity: O(n*log n)
Best Case Time Complexity: O(n*log n)
Average Time Complexity: O(n*log n)
Space Complexity : O(1)

- Heap sort is not a Stable sort, and requires a constant space for sorting a list.
- Heap Sort is very fast and is widely used for sorting.

Ex: Input data: 4, 10, 3, 5, 1

```
       4(0)
      /   \
   10(1)   3(2)
  /   \
5(3)   1(4)
```

Tulsiramji Gaikwad-Patil College of Engineering and Technology
Wardha Road, Nagpur-441 108
NAAC Accredited

Department of Computer Science & Engineering

The numbers in bracket represent the indices in the array representation of data.
Applying heapify procedure to index 1:

```
        4(0)
       /   \
    10(1)    3(2)
   /   \
5(3)    1(4)
```

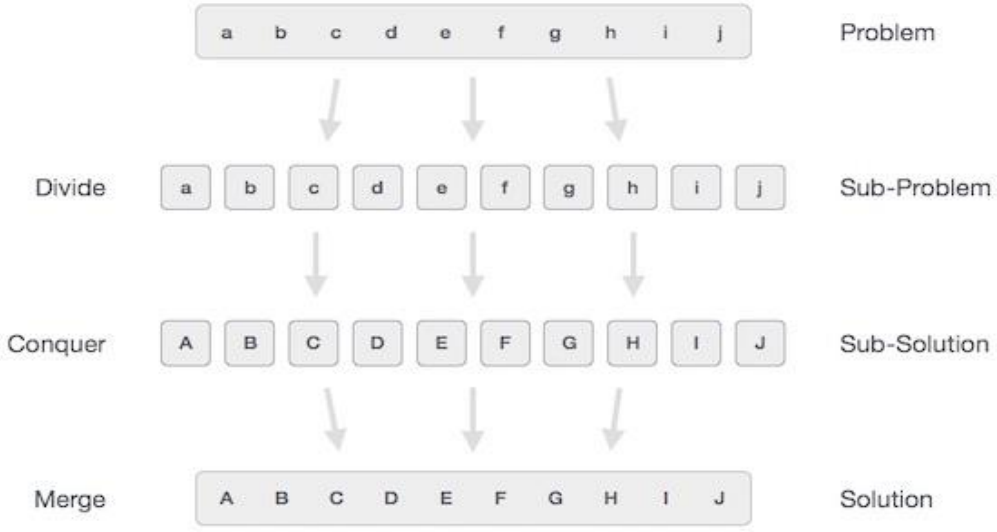
Applying heapify procedure to index 0:

```
        10(0)
       /  \
     5(1)  3(2)
    /   \
 4(3)    1(4)
```

The heapify procedure calls itself recursively to build heap in top down manner.

Q5. What do you mean by Divide and conquer strategy? Give suitable example for the same. 4M (S-18,W-16)

Ans: In divide and conquer approach, the problem is divided into sub-problems and each sub-problem is independently solved. The problems can be divided into sub-problems and sub-problems to even smaller sub-problems, but to a stage where division is not possible. Now the smallest possible sub-problems of the sub-problems are solved. Finally, the solution of all the sub-problems is merged to obtain solution of original problem.



The concept of **divide-and-conquer approach** is explained in a three-step process.

Divide/Break

Tulsiramji Gaikwad-Patil College of Engineering and Technology
Wardha Road, Nagpur-441 108
NAAC Accredited

Department of Computer Science & Engineering

In this step, the problem is broken into smaller sub-problems such that each sub-part should represent a part of the original problem. Recursive approach is used in this step to divide the problem till further sub-division of the problem is not possible. In this approach, the problems turn to be atomic in nature but even then represent some part of the original problem.

Conquer/Solve

In this step, the sub-problems are solved. At this level, usually the problems are considered 'solved' on their own.

Merge/Combine

After the sub-problems being solved, this stage enables in combining the solution to formulate a solution for the original problem. This algorithmic approach works recursively and conquers & merge steps works so close that they appear as one.

The computer algorithms which are based on **divide-and-conquer** programming approach are:

- Merge Sort
- Quick Sort
- Binary Search
- Strassen's Matrix Multiplication
- Closest pair (points)

Q6. Explain the concept of data structure in detail. Also explain abstract data type. **7M**
(S-17,W-16)

Ans: Data Structure is a way of collecting and organising data in such a way that we can perform operations on these data in an effective way. Data Structures is about rendering data elements in terms of some relationship, for better organization and storage.

Data Structures are structures programmed to store ordered data, so that various operations can be performed on it easily. It represents the knowledge of data to be organized in memory. It should be designed and implemented in such a way that it reduces the complexity and increases the efficiency.

Some example of Abstract Data Structure are :

- Linked List
- Tree
- Graph
- Stack, Queue etc.

All these data structures allow us to perform different operations on data.

Abstract Data type (ADT) is a type (or class) for objects whose behavior is defined by a set of value and a set of operations. The definition of ADT only mentions what operations are

Tulsiramji Gaikwad-Patil College of Engineering and Technology
Wardha Road, Nagpur-441 108
NAAC Accredited

Department of Computer Science & Engineering

to be performed but not how these operations will be implemented. It does not specify how data will be organized in memory and what algorithms will be used for implementing the operations. It is called "abstract" because it gives an implementation independent view.
The process of providing only the essentials and hiding the details is known as abstraction. The user of data type need not know that data type is implemented, for example, we have been using int, float, char data types only with the knowledge with values that can take and operations that can be performed on them without any idea of how these types are implemented. So a user only needs to know what a data type can do but not how it will do it. We can think of ADT as a black box which hides the inner structure and design of the data type. Now we'll define three ADTs namely List ADT, Stack ADT, Queue ADT.

Q7. Write a C program to sort the elements of matrix row wise Assume that the matrix is represented by two dimensional array. **6M (S-17,S-16)**

Ans:

```
#include <stdio.h>
#include<conio.h>
void main()
{
   static int array1[10][10], array2[10][10];
   int i, j, k, a, m, n;

   printf("Enter the order of the matrix \n");
   scanf("%d %d", &m, &n);
   printf("Enter co-efficients of the matrix \n");
   for (i = 0; i < m; ++i)
   {
      for (j = 0; j < n; ++j)
      {
         scanf("%d", &array1[i][j]);
         array2[i][j] = array1[i][j];
      }
   }
   printf("The given matrix is \n");
   for (i = 0; i < m; ++i)
   {
      for (j = 0; j < n; ++j)
      {
            printf(" %d", array1[i][j]);
      }
      printf("\n");
   }
```

Tulsiramji Gaikwad-Patil College of Engineering and Technology
Wardha Road, Nagpur-441 108
NAAC Accredited

Department of Computer Science & Engineering

```
printf("After arranging rows in ascending order\n");
for (i = 0; i < m; ++i)
{
   for (j = 0; j < n; ++j)
   {
      for (k =(j + 1); k < n; ++k)
      {
         if (array1[i][j] > array1[i][k])
         {
            a = array1[i][j];
            array1[i][j] = array1[i][k];
            array1[i][k] = a;
         }
      }
   }
}
for (i = 0; i < m; ++i)
{
   for (j = 0; j < n; ++j)
   {
      printf(" %d", array1[i][j]);
   }
   printf("\n");
}
printf("After arranging the columns in descending order \n");
for (j = 0; j < n; ++j)
{
   for (i = 0; i < m; ++i)
   {
      for (k = i + 1; k < m; ++k)
      {
         if (array2[i][j] < array2[k][j])
         {
            a = array2[i][j];
            array2[i][j] = array2[k][j];
            array2[k][j] = a;
         }
      }
   }
}
for (i = 0; i < m; ++i)
{
```

Tulsiramji Gaikwad-Patil College of Engineering and Technology
Wardha Road, Nagpur-441 108
NAAC Accredited

Department of Computer Science & Engineering

```
        for (j = 0; j < n; ++j)
        {
            printf(" %d", array2[i][j]);
        }
        printf("\n");
    }
}
```

Tulsiramji Gaikwad-Patil College of Engineering and Technology
Wardha Road, Nagpur-441 108
NAAC Accredited

Department of Computer Science & Engineering

Q: Give the snapshots an element using linear and binary search in the given array 20 91 32 81 75 48 05 19 08 Also comment on time complexity. 7M (S-14)

Ans: Linear Search :-

Array elements along with Array index are given below. Let us search an element '48' using linear search.

Array Index	0	1	2	3	4	5	6	7	8
Array Elements	20	91	32	81	75	48	05	19	08

Element = 48?

Is a[0] == 48? yes/No → ie. 20 ≠ 48

∴ Increment array index one position right

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]
20	91	32	81	75	48	05	19	08

Is a[1] == 48? yes/No → ie. 91 ≠ 48 ∴ Increment array index by 1

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]
20	91	32	81	75	48	05	19	08

IS a[2] == 48? yes/No → No ie. 32 ≠ 48 ∴ Increment array index by 1

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]
20	91	32	81	75	48	05	19	08

Is a[3] == 48? yes/No → No ie. 81 ≠ 48 ∴ Increment array index by 1

Tulsiramji Gaikwad-Patil College of Engineering and Technology
Wardha Road, Nagpur-441 108
NAAC Accredited

Department of Computer Science & Engineering

DATE :

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]
20	91	32	81	75	48	05	19	08

↑

Is a[4] == 48 ? Yes/No → No ie. 75 ≠ 48 ∴ Increment array index by 1

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]
20	91	32	81	75	48	05	19	08

↑

Is a[5] == 48? Yes/No → Yes ie. 48 == 48 ∴ Break.
Hence, element 48 is found at index position 5.

Binary search :- We have given total 9 elements of array.
precondition for Binary search is elements must be in sorted form either in ascending or descending order. So, here we assume elements are sorted in ascending order. And we search element '81'.
Sorted array elements along with array index are given below :-

Array Index									
Array Elements	20	91	32	81	75	48			

Array Index	0	1	2	3	4	5	6	7	8
Array Elements	05	08	19	20	32	48	75	81	91

Step 1 :- The given array is in Ascending order and element to be search is '81'.

Tulsiramji Gaikwad-Patil College of Engineering and Technology
Wardha Road, Nagpur-441 108
NAAC Accredited

Department of Computer Science & Engineering

beg = 0, last = 8

mid = int((beg + last)/2) = int((0+8)/2) = int(8/2)
= int(4) = 4

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]
05	08	19	20	32	48	75	81	91
beg				mid				last

Step2 :- a[mid] ie. a[3] = 32
32 < 81 (T) then
beg = mid + 1 = 4 + 1 = 5

Step3 :- mid = int((beg + last)/2)
= int((5+8)/2)
= int(13/2)
= int(6)
= 6

a[5]	a[6]	a[7]	[8]
48	75	81	91
beg	mid		last

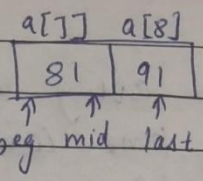
a[mid] ie. a[6] = 75
75 < 81 (T) then
beg = mid + 1 = 6 + 1 = 7

Step 4 :- mid = int((beg + last)/2)
= int((7+8)/2)
= int(15/2)
= int(7)
= 7

Tulsiramji Gaikwad-Patil College of Engineering and Technology
Wardha Road, Nagpur-441 108
NAAC Accredited

Department of Computer Science & Engineering

PAGE NO.
DATE :



a[mid] ie. a[7] = 81

81 == 81 . Search Successful.

Hence, element 81 is found at Index position 7.

Time complexity :-

Linear Search :-

(i) If the desired record is present in the first position of the search table, then only one comparision is made.

(ii) If desired record is the last one, then n comparisions have to be made.

(iii) If the record is present somewhere in the search table, on an average, the number of comparisions will be (n+1)/2.

(iv) The worst case efficiency of Linear search is O(n).

Binary Search :- In this, every time we are reducing the search area. So, no. of comparisions keep on decreasing. In worst case, the no. of comparisions is atmost log(N+1). So, it is an efficient algorithm when compared to linear search but array has to be sorted before doing Binary Search.

Tulsiramji Gaikwad-Patil College of Engineering and Technology
Wardha Road, Nagpur-441 108
NAAC Accredited

Department of Computer Science & Engineering

DATE :

Q. Sort the following array using selection sort. Also specify time complexity.
80, 27, 42, 14, 69, 22, 85. 5M (S-18)

Ans. Given 7 elements of array. we arrange these elements in Ascending order
As we have 7 elements, so we need (n-1) pass
ie Max. 6 pass to sort the elements.

pass 1 of 6:
* Find the smallest element from the search range and exchange with swap position.

	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]
Given Array	80	27	42	14	69	22	85

Swap position ↓ Smallest element ↓

	80	27	42	14	69	22	85

← Search Range →

After swap	14	27	42	80	69	22	85

pass 2 of 6:

From pass 1:-	14	27	42	80	69	22	85

Swap position ↓ Smallest element ↓

	14	27	42	80	69	22	85

← Search Range →

After Swap:-	14	22	42	80	69	27	85

Tulsiramji Gaikwad-Patil College of Engineering and Technology
Wardha Road, Nagpur-441 108
NAAC Accredited

Department of Computer Science & Engineering

DATE :

pass 3 of 6 :-

from pass 2 :- | 14 | 22 | 42 | 80 | 69 | 27 | 85 |

Swap Position, Smallest element

| 14 | 22 | 42 | 80 | 69 | 27 | 85 |

← Search Range →

After Swap :- | 14 | 22 | 27 | 80 | 69 | 42 | 85 |

pass 4 of 6 :-

from pass 3 :- | 14 | 22 | 27 | 80 | 69 | 42 | 85 |

Swap Position, Smallest element

| 14 | 22 | 27 | 80 | 69 | 42 | 85 |

← Search Range →

After Swap :- | 14 | 22 | 27 | 42 | 69 | 80 | 85 |

pass 5 of 6 :-

from pass 4 :- | 14 | 22 | 27 | 42 | 69 | 80 | 85 |

Swap position, Smallest element

| 14 | 22 | 27 | 42 | 69 | 80 | 85 |

← Search Range →

As, Swap position & Smallest element pointing to same index

So, No change | 14 | 22 | 27 | 42 | 69 | 80 | 85 |

pass 6 of 6 :-

from pass 5 :- | 14 | 22 | 27 | 42 | 69 | 80 | 85 |

Swap position, Smallest element

| 14 | 22 | 27 | 42 | 69 | 80 | 85 |

← Search Range →

Swap position & Smallest element point at same position

So, No change | 14 | 22 | 27 | 42 | 69 | 80 | 85 |

Sorted Array.

Tulsiramji Gaikwad-Patil College of Engineering and Technology
Wardha Road, Nagpur-441 108
NAAC Accredited

Department of Computer Science & Engineering

~~Q.~~ ~~Sort the following Array using Selection Sort.~~
~~Also~~

Time Complexity:

For n elements array,
In 1st pass we have n-1 comparisions
In 2nd pass we have n-2 comparisions
Ily, In Kth pass we have n-K comparisions
And last pass requires 1 comparision

∴ Total comparisions are,

$$f(n) = (n-1)+(n-2)+(n-3)+\dots+3+2+1$$
$$= \frac{n(n-1)}{2}$$
$$= O(n^2)$$

Tulsiramji Gaikwad-Patil College of Engineering and Technology
Wardha Road, Nagpur-441 108
NAAC Accredited

Department of Computer Science & Engineering

PAGE NO.
DATE :

8. Sort the following Array using selection sort. Also specify Time complexity.
7, 3, 44, 11, 2, 66, 5, 9.
6M (S-16)

Ans. Given 8 elements of array. we arrange these elements in Ascending order.
As we have 8 elements, so we need (n-1) i.e. Max 7 pass to sort the elements.

Pass 1 of 7:-
* find the smallest element from the search range and exchange with swap position.

Given Array

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]
7	3	44	11	2	66	5	9

Swap Position — Smallest element

7	3	44	11	2	66	5	9

← Search Range →

After swap:-

2	3	44	11	7	66	5	9

Pass 2 of 7:-
From pass 1:-

2	3	44	11	7	66	5	9

Swap Position, Smallest Element

2	3	44	11	7	66	5	9

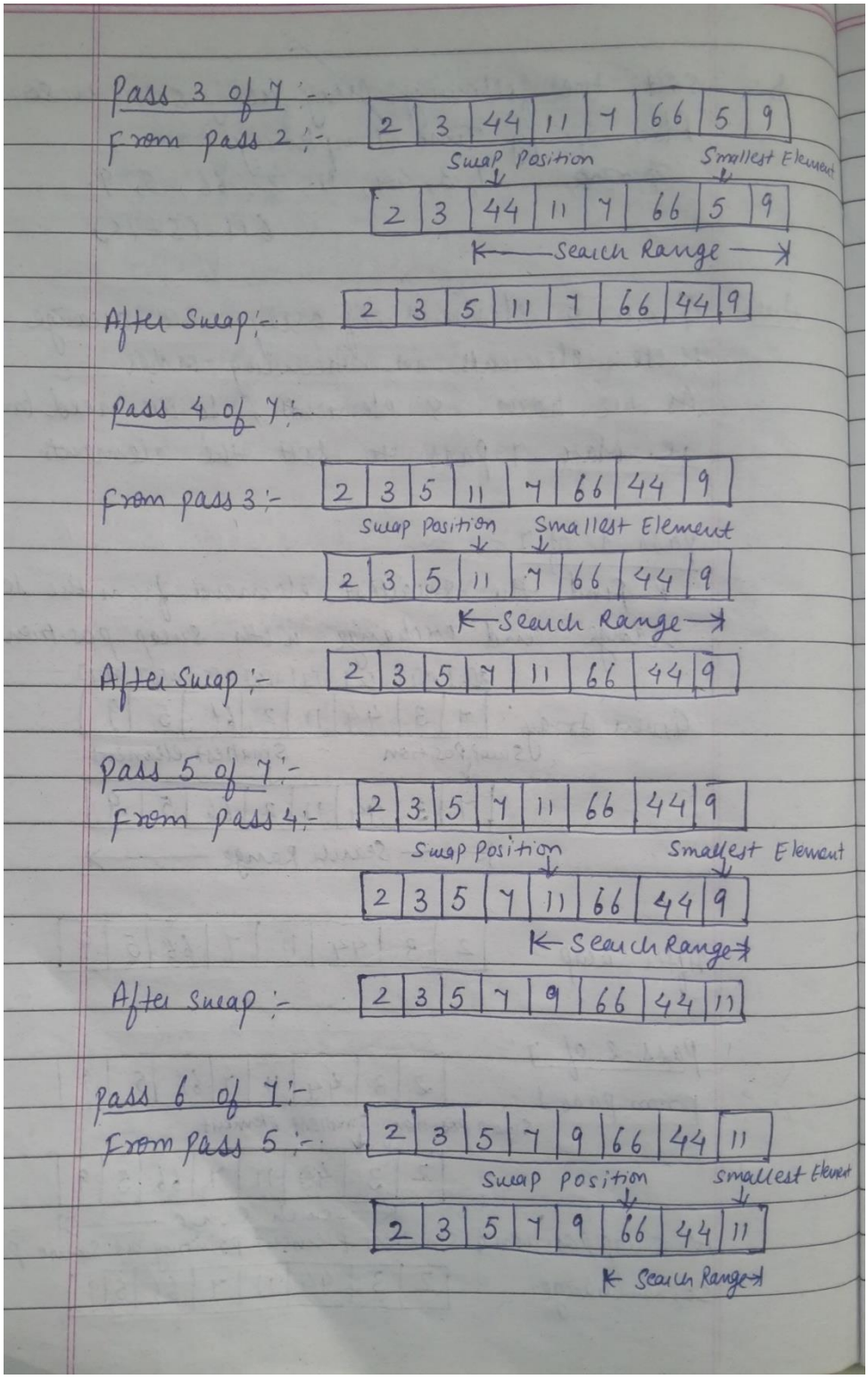
← search Range →

As swap position & Smallest Element pointing at same position
So, no change.

2	3	44	11	7	66	5	9

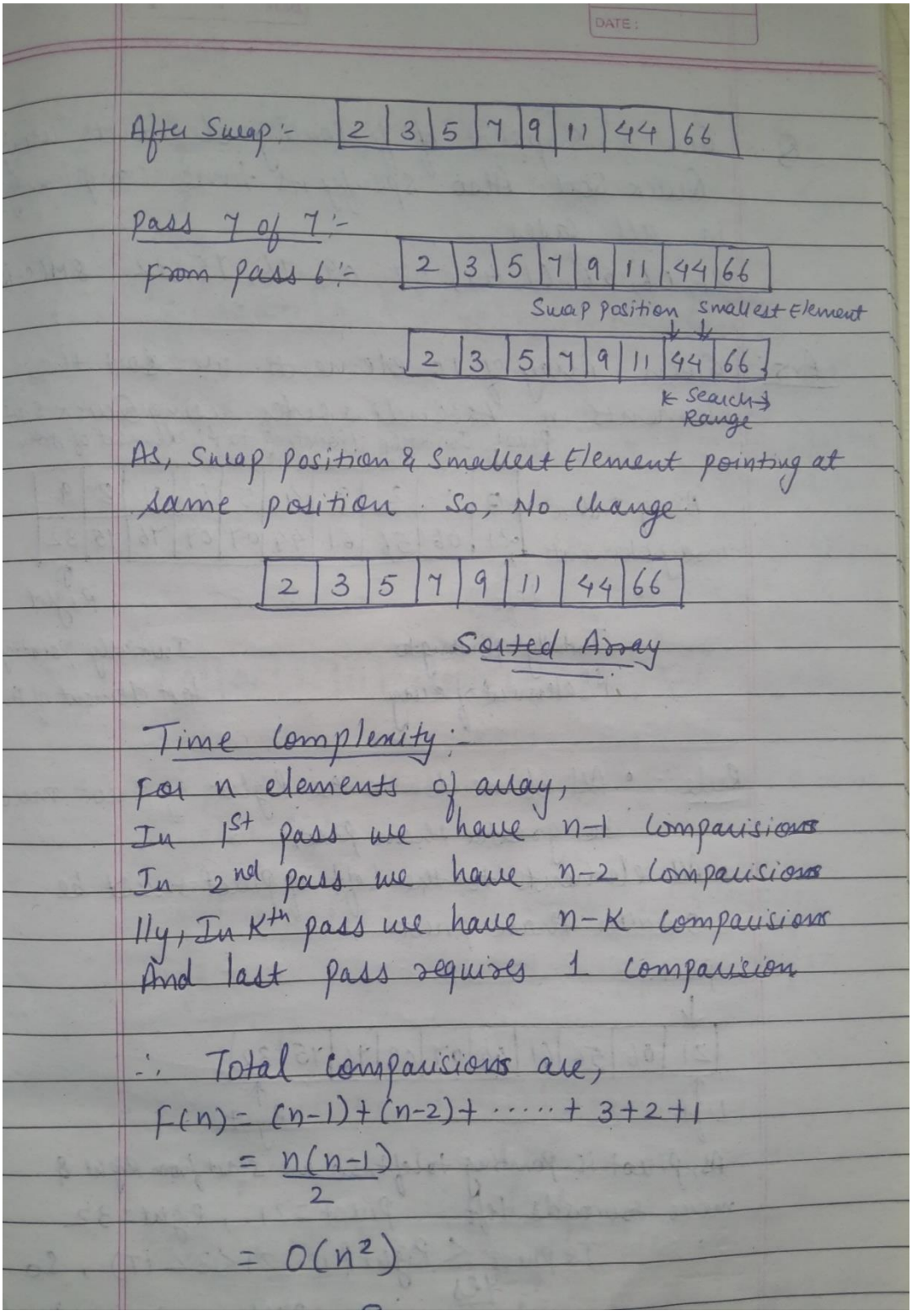
Tulsiramji Gaikwad-Patil College of Engineering and Technology
Wardha Road, Nagpur-441 108
NAAC Accredited

Department of Computer Science & Engineering



Tulsiramji Gaikwad-Patil College of Engineering and Technology
Wardha Road, Nagpur-441 108
NAAC Accredited

Department of Computer Science & Engineering



DATE :

After Swap :- | 2 | 3 | 5 | 7 | 9 | 11 | 44 | 66 |

Pass 7 of 7 :-

from pass 6 :- | 2 | 3 | 5 | 7 | 9 | 11 | 44 | 66 |

Swap position, Smallest Element

| 2 | 3 | 5 | 7 | 9 | 11 | 44 | 66 |

K, Search Range

As, Swap position & Smallest Element pointing at same position. So, No change.

| 2 | 3 | 5 | 7 | 9 | 11 | 44 | 66 |

Sorted Array

Time complexity :-

For n elements of array,

In 1st pass we have n-1 comparisions

In 2nd pass we have n-2 comparisions

Ily, In K^{th} pass we have n-K comparisions

And last pass requires 1 comparision

∴ Total comparisions are,

$f(n) = (n-1)+(n-2)+\cdots+3+2+1$

$= \frac{n(n-1)}{2}$

$= O(n^2)$

Tulsiramji Gaikwad-Patil College of Engineering and Technology
Wardha Road, Nagpur-441 108
NAAC Accredited

Department of Computer Science & Engineering

Q. Give a Snapshot of following elements using Quick Sort. Also specify its time complexity in all cases.
21, 06, 56, 61, 44, 07, 09, 76, 75, 32. SMCS-10

Ans:- Given, Array of 10 elements. we sort the elements in Ascending order using Quick Sort

Pivot Initially Pointing to 1st element of Array

Array Index	0	1	2	3	4	5	6	7	8	9
Array Elements	21	06	56	61	44	07	09	76	75	32

Left — Initially pointing to 1st element of array

Right — Initially pointing to last element of Array

Rule:- • All elements to the Right of Pivot must be greater than Pivot.
• All elements to the left of Pivot must be smaller than Pivot.

Pivot

21	06	56	61	44	07	09	76	75	32

Left — Right

As, Pivot is Pointing to left, so we start from Right & move towards left. Pivot = 21, Right = 32

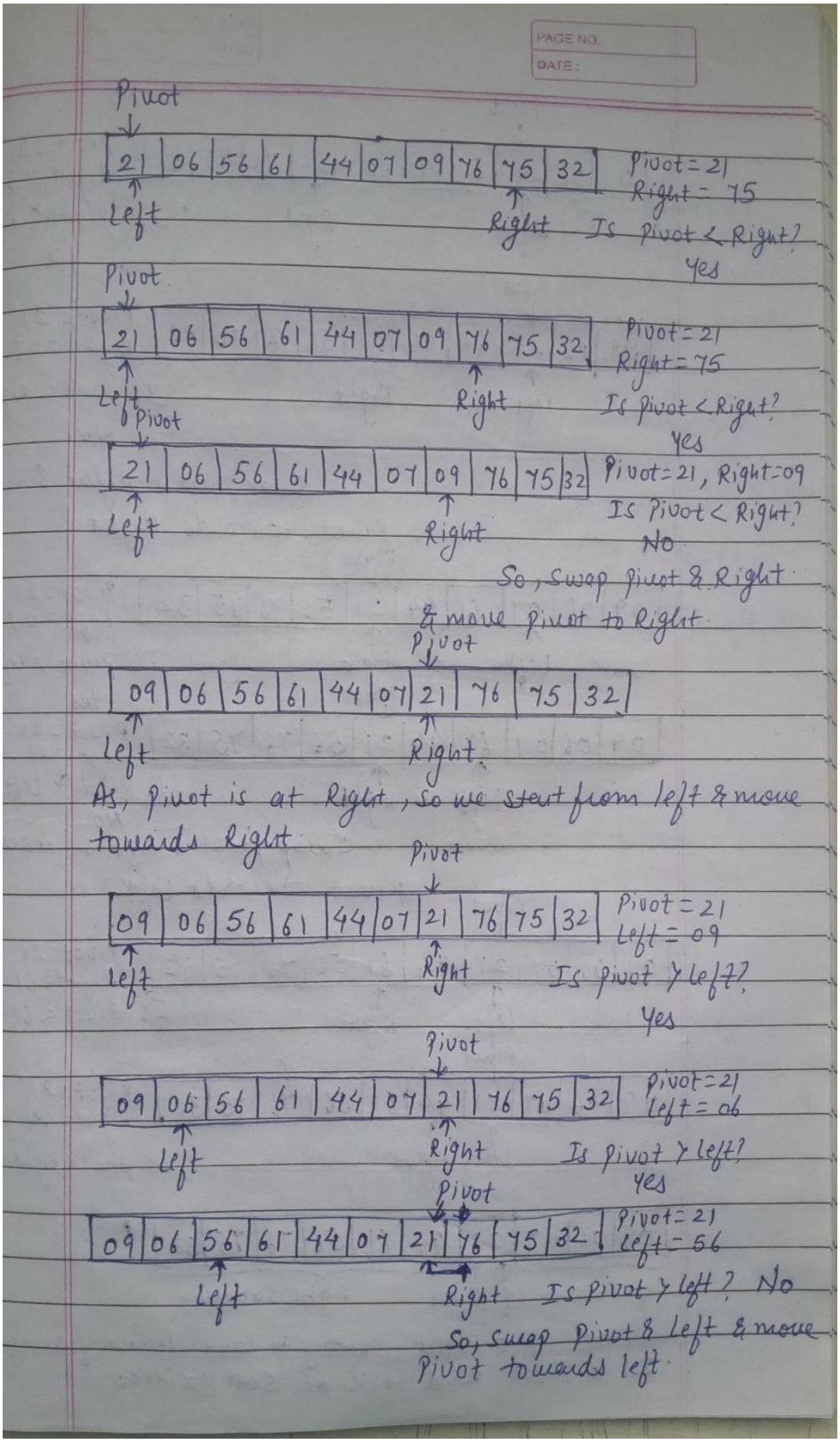
Is Pivot < Right? ie. 21 < 32 (T), So

Yes

move Right one position towards left.

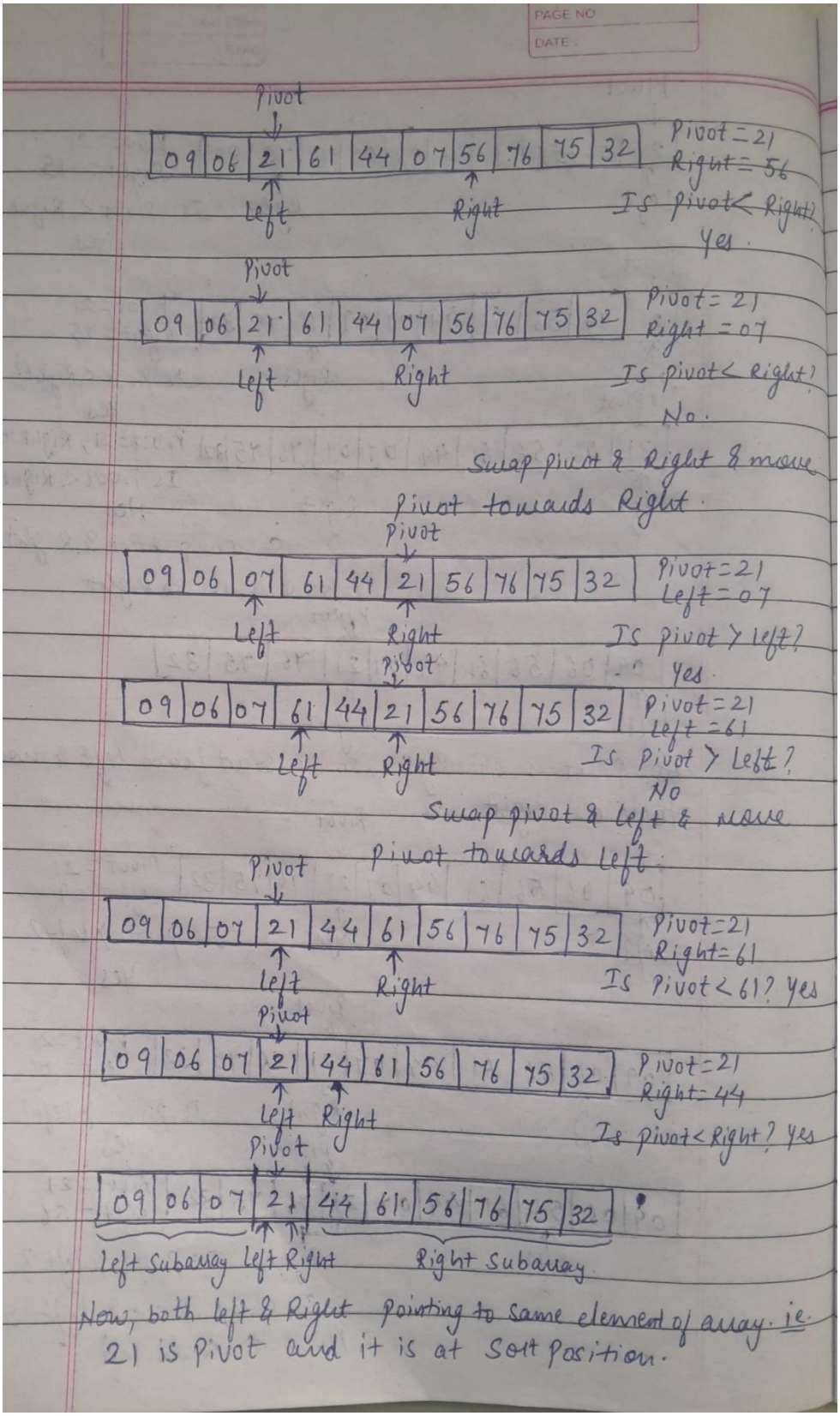
Tulsiramji Gaikwad-Patil College of Engineering and Technology
Wardha Road, Nagpur-441 108
NAAC Accredited

Department of Computer Science & Engineering



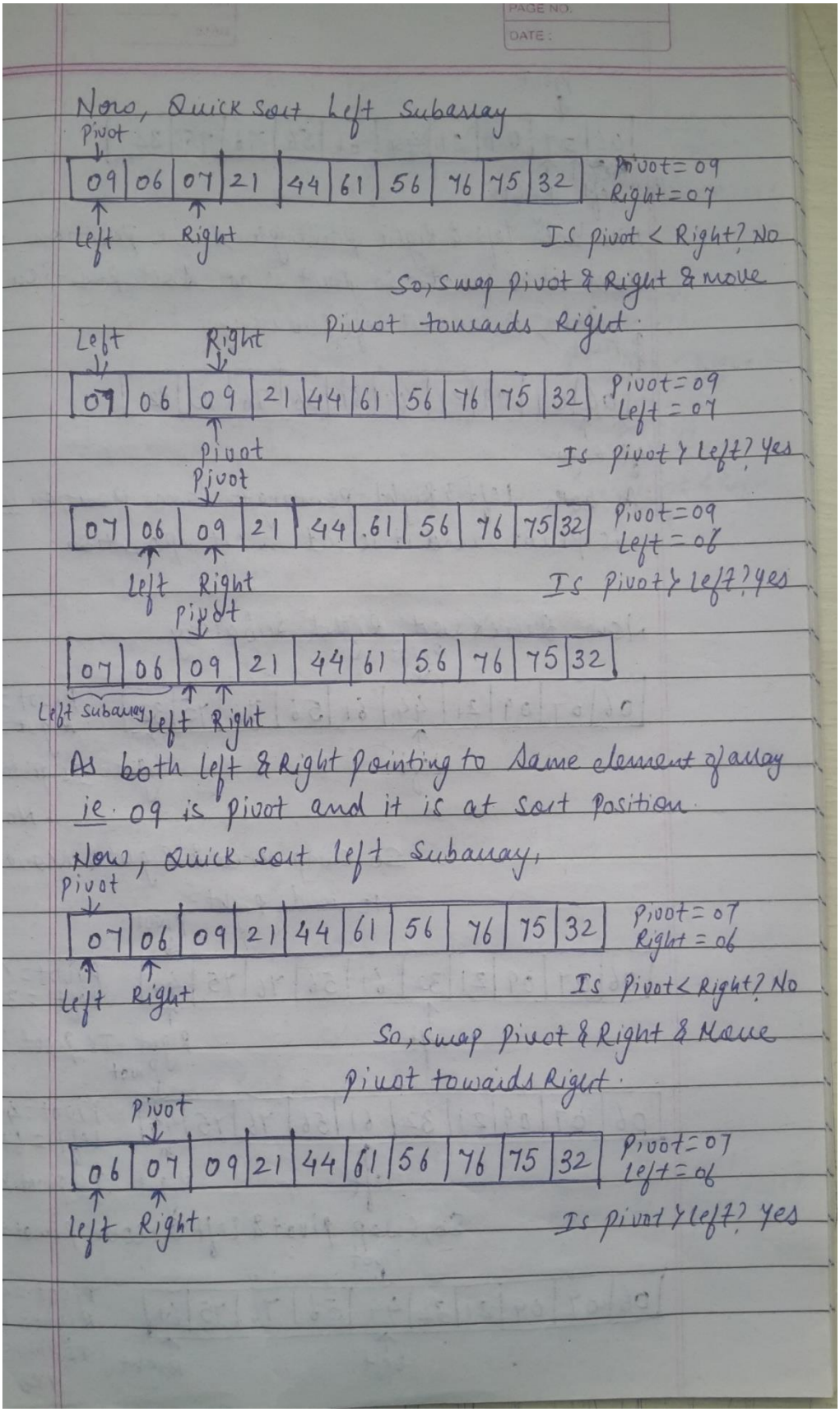
Tulsiramji Gaikwad-Patil College of Engineering and Technology
Wardha Road, Nagpur-441 108
NAAC Accredited

Department of Computer Science & Engineering



Tulsiramji Gaikwad-Patil College of Engineering and Technology
Wardha Road, Nagpur-441 108
NAAC Accredited

Department of Computer Science & Engineering



Now, Quick sort Left Subarray

09	06	07	21	44	61	56	76	75	32

Pivot, Left, Right

Pivot=09 Right=07

Is pivot < Right? No

So, swap pivot & Right & move pivot towards Right.

09	06	09	21	44	61	56	76	75	32

Left, Right, pivot

Pivot=09 Left=07

Is pivot > left? yes

07	06	09	21	44	61	56	76	75	32

Left, Right, pivot

Pivot=09 Left=06

Is pivot > left? yes

07	06	09	21	44	61	56	76	75	32

Left subarray: Left, Right

As both left & Right pointing to same element of array ie. 09 is pivot and it is at sort position.

Now, Quick sort left subarray.

07	06	09	21	44	61	56	76	75	32

Pivot, Left, Right

Pivot=07 Right=06

Is Pivot < Right? No

So, Swap Pivot & Right & Move pivot towards Right.

06	07	09	21	44	61	56	76	75	32

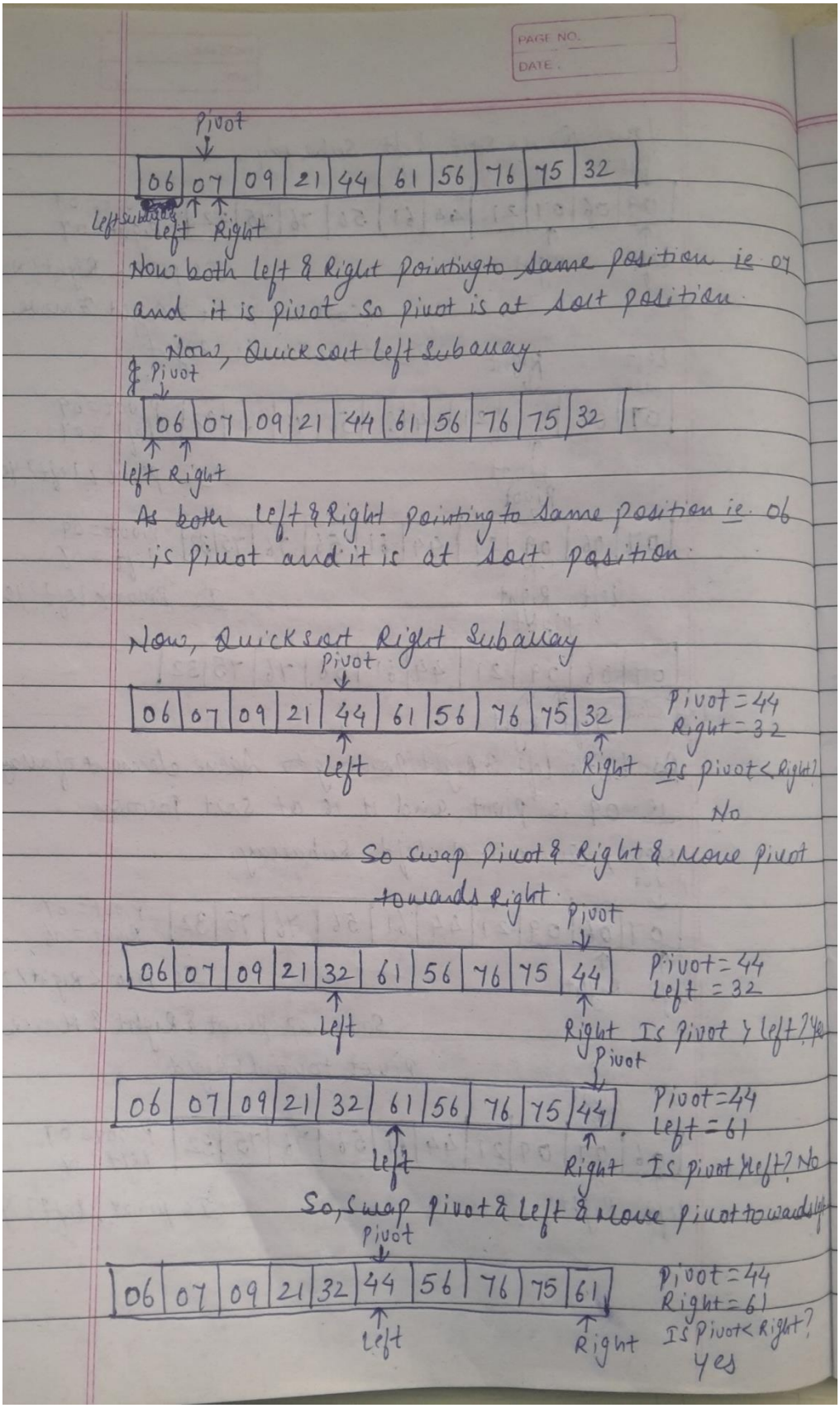
Pivot, left, Right

Pivot=07 Left=06

Is pivot > left? yes

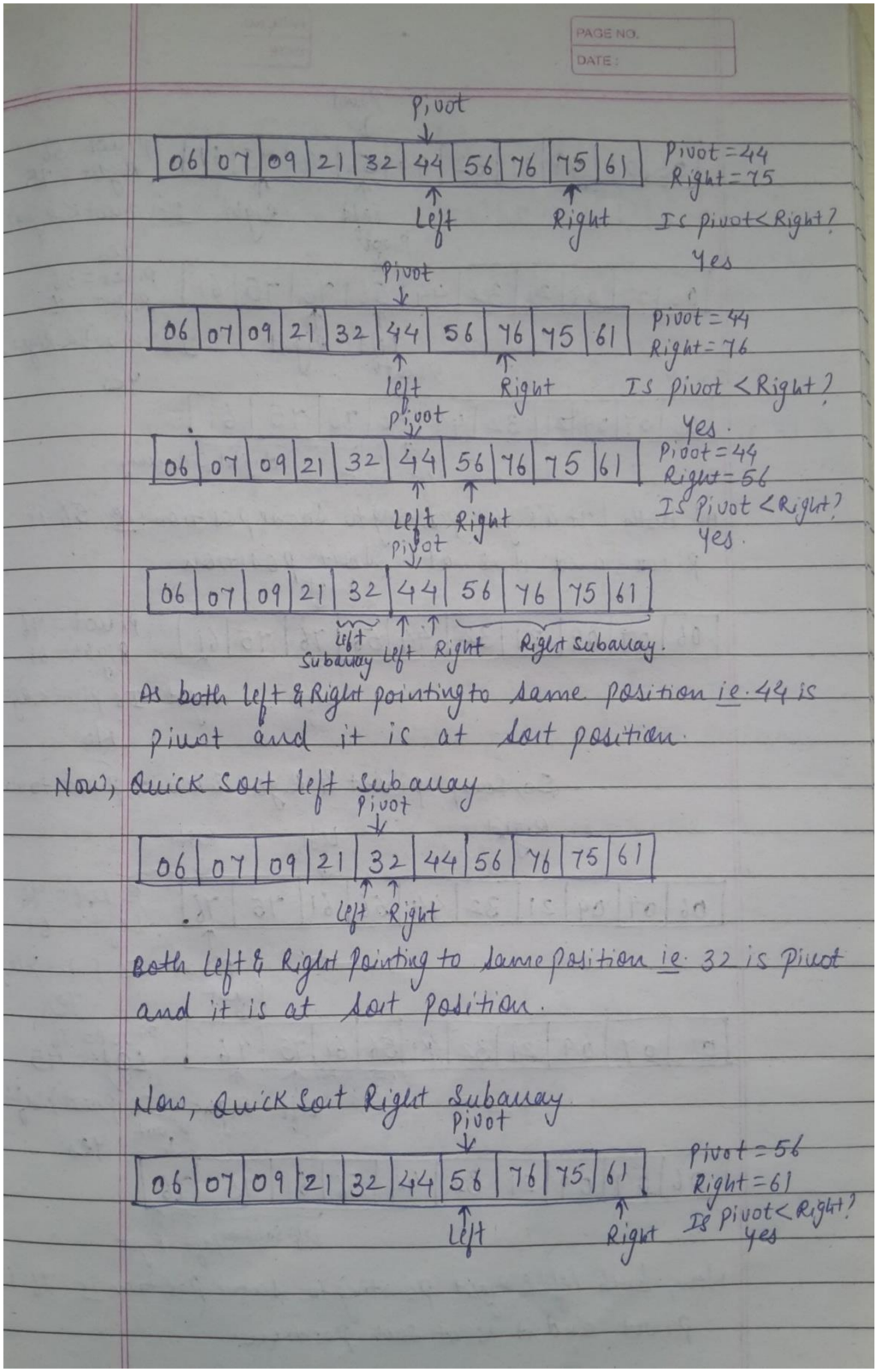
Tulsiramji Gaikwad-Patil College of Engineering and Technology
Wardha Road, Nagpur-441 108
NAAC Accredited

Department of Computer Science & Engineering



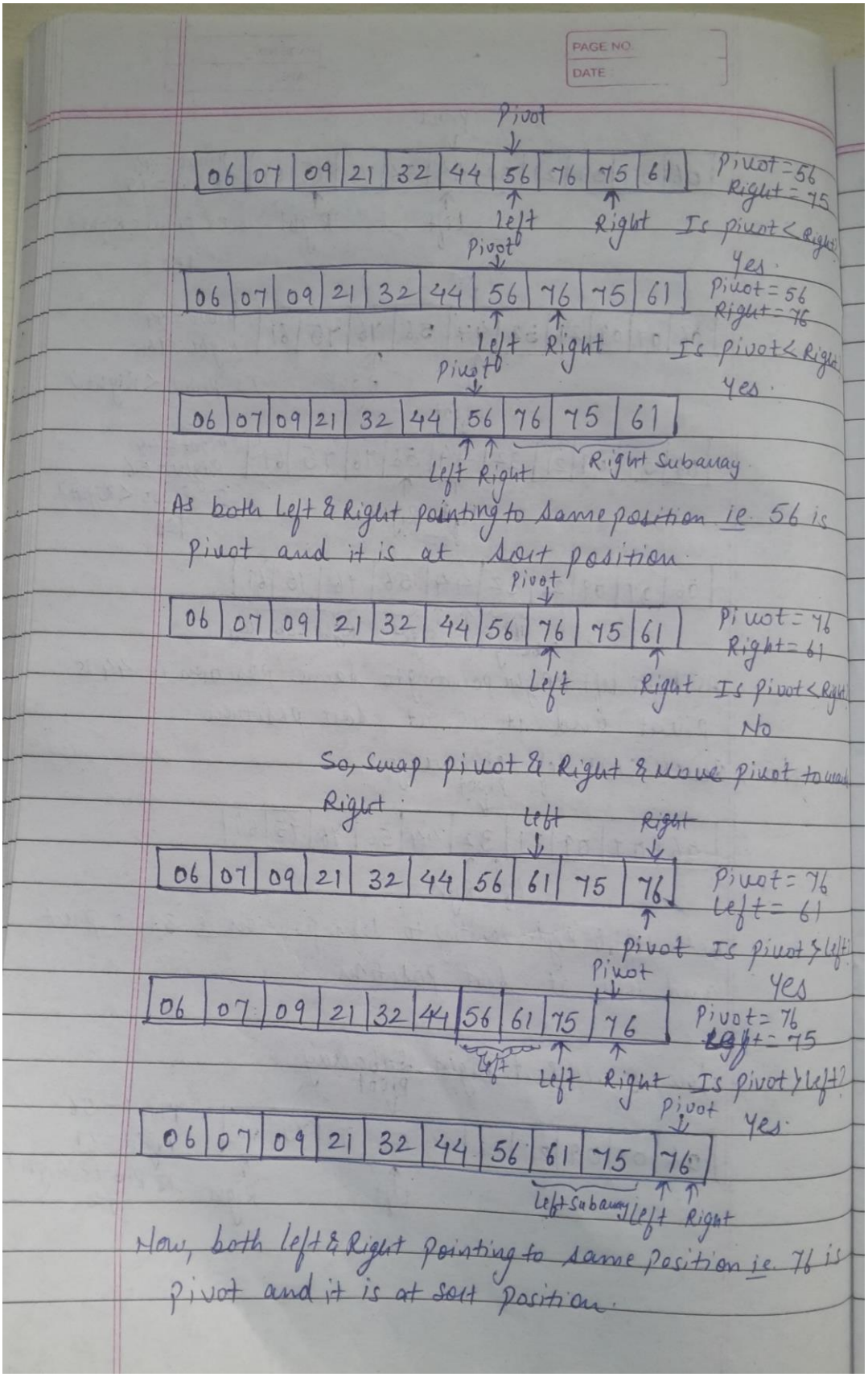
Tulsiramji Gaikwad-Patil College of Engineering and Technology
Wardha Road, Nagpur-441 108
NAAC Accredited

Department of Computer Science & Engineering



Tulsiramji Gaikwad-Patil College of Engineering and Technology
Wardha Road, Nagpur-441 108
NAAC Accredited

Department of Computer Science & Engineering



Tulsiramji Gaikwad-Patil College of Engineering and Technology
Wardha Road, Nagpur-441 108
NAAC Accredited

Department of Computer Science & Engineering

PAGE NO.
DATE :

Now, Quick sort left subarray.

06	07	09	21	32	44	56	61	75	76

Pivot

Pivot = 61
Right = 75

Left Right

If pivot < Right?
Yes.

06	07	09	21	32	44	56	61	75	76

Pivot

Left Right

Right subarray.

Now both left & Right pointing to same position ie 61 is pivot and it is at sort position

Now, Quicksort Right subarray.

06	07	09	21	32	44	56	61	75	76

Pivot

Left Right

As both Left & Right pointing to same position ie. 75 and is pivot and it is at sort position.

finally, Sorted Array is,

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]
06	07	09	21	32	44	56	61	75	76

Time complexity
for an Array of n elements

$$F(n) = O(n \log_2 n)$$