



Experiment No.: 2

Title: Divide and Conquer Strategy



Batch: B-4

Roll No.: 16010422234

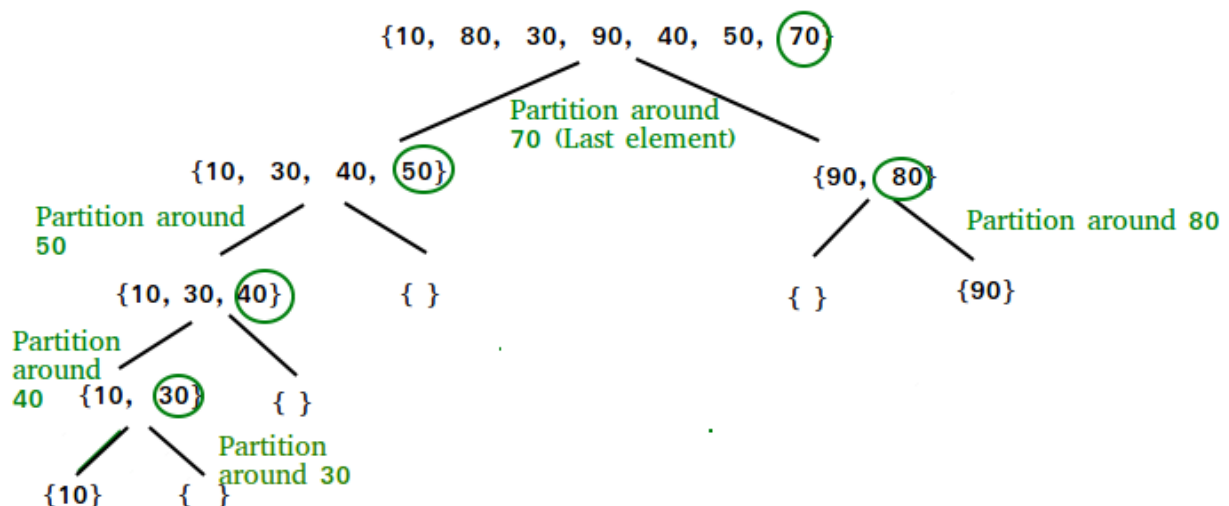
Name: Chandana Ramesh Galgali

Experiment No.: 02**Aim:** To implement and analyse time complexity of Quick-sort.**Explanation and Working of Quick sort:**

QuickSort is a sorting algorithm based on the Divide and Conquer algorithm that picks an element as a pivot and partitions the given array around the picked pivot by placing the pivot in its correct position in the sorted array.

The key process in quickSort is a partition(). The target of partitions is to place the pivot (any element can be chosen to be a pivot) at its correct position in the sorted array and put all smaller elements to the left of the pivot, and all greater elements to the right of the pivot.

Partition is done recursively on each side of the pivot after the pivot is placed in its correct position and this finally sorts the array.

Choice of Pivot:

There are many different choices for picking pivots.

- Always pick the first element as a pivot.
- Always pick the last element as a pivot.
- Pick a random element as a pivot.
- Pick the middle as the pivot.

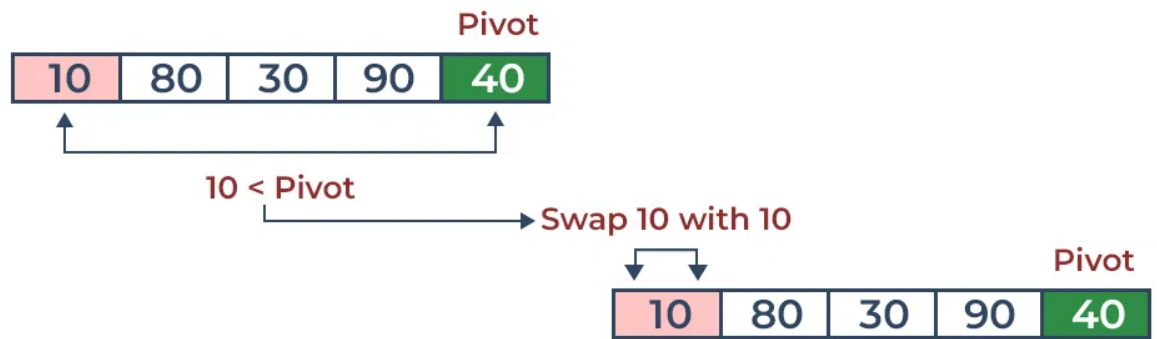
Partition Algorithm:

The logic is simple, we start from the leftmost element and keep track of the index of smaller (or equal) elements as i . While traversing, if we find a smaller element, we swap the current element with $arr[i]$. Otherwise, we ignore the current element.

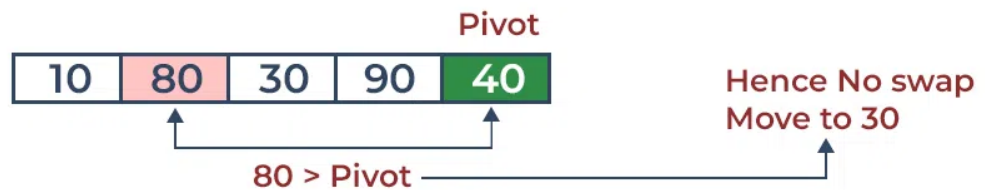
Let us understand the working of partition and the Quick Sort algorithm with the help of the following example:

Consider: $arr[] = \{10, 80, 30, 90, 40\}$.

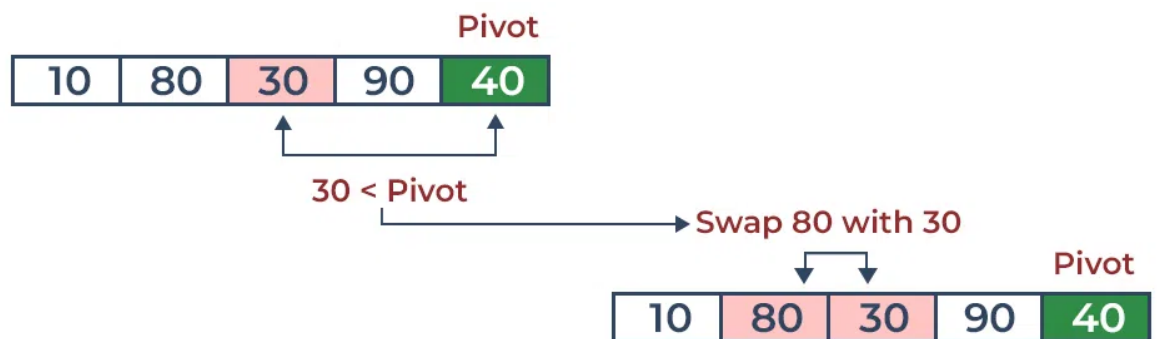
- Compare 10 with the pivot and as it is less than pivot arrange it accordingly.



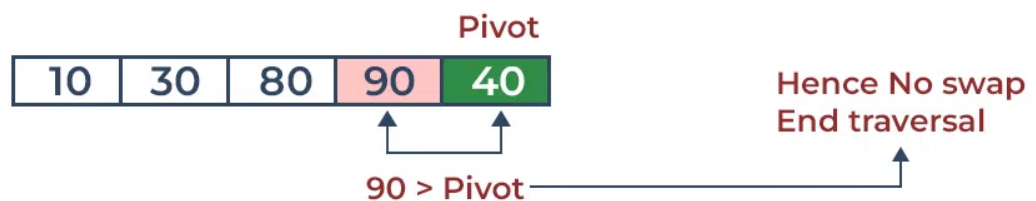
- Compare 80 with the pivot. It is greater than pivot.



- Compare 30 with pivot. It is less than pivot so arrange it accordingly.



- Compare 90 with the pivot. It is greater than the pivot.



- Arrange the pivot in its correct position.

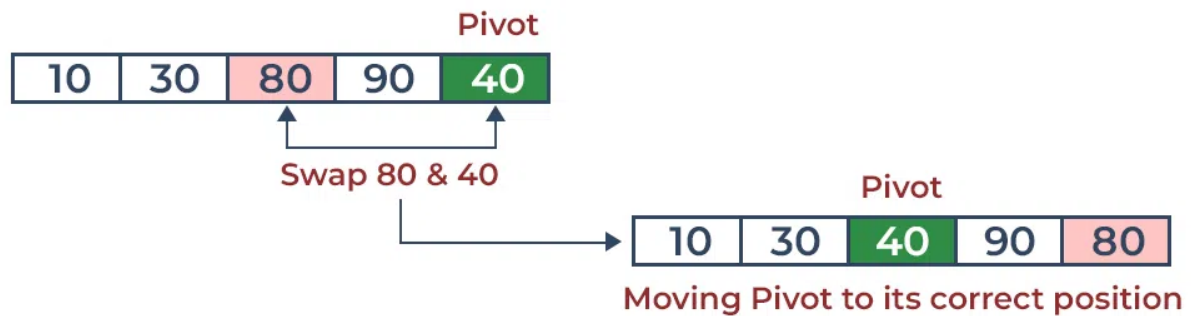
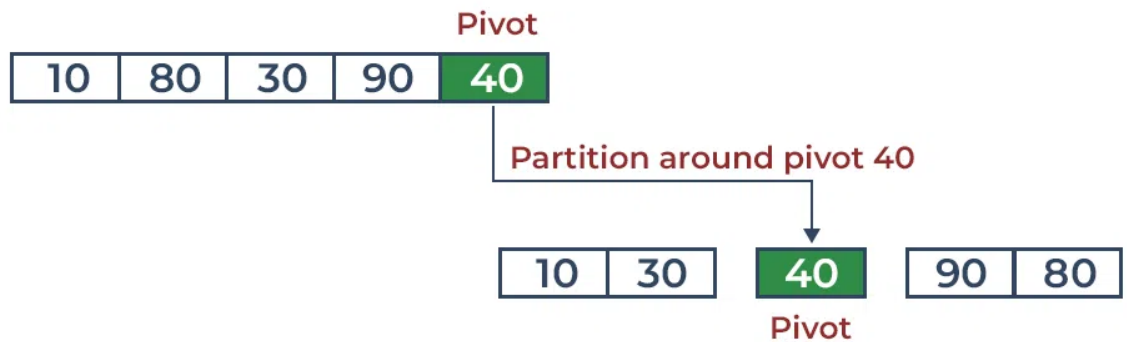


Illustration of Quicksort:

As the partition process is done recursively, it keeps on putting the pivot in its actual position in the sorted array. Repeatedly putting pivots in their actual position makes the array sorted. Follow the below images to understand how the recursive implementation of the partition algorithm helps to sort the array.

- Initial partition on the main array:



- Partitioning of the subarrays:



Algorithm of Quick sort:

```

QUICKSORT (array A, start, end)
{
  if (start < end)
  {
    p = partition(A, start, end)
    QUICKSORT (A, start, p - 1)
    QUICKSORT (A, p + 1, end)
  }
}

```

Partition Algorithm:

The partition algorithm rearranges the sub-arrays in a place.

PARTITION (array A, start, end)

```
{
  pivot = A[end]
  i = start-1
  for j = start to end -1
  {
    do if (A[j] < pivot)
    {
      then i = i + 1
      swap A[i] with A[j]
    }
  }
  swap A[i+1] with A[end]
  return i+1
}
```

Derivation of Analysis Quick sort:**Worst Case Analysis**

In quick sort, the worst case occurs when the pivot element is either the greatest or smallest element. Suppose, if the pivot element is always the last element of the array, the worst case would occur when the given array is sorted already in ascending or descending order. The worst-case time complexity of quicksort is $O(n^2)$.

Best Case Analysis

In Quicksort, the best-case occurs when the pivot element is the middle element or near to the middle element. The best-case time complexity of quicksort is $O(n * \log n)$.

Average Case Analysis

It occurs when the array elements are in jumbled order that is not properly ascending and not properly descending. The average case time complexity of quicksort is $O(n * \log n)$.

Program(s) of Quick sort:

```
#include<stdio.h>
void swap(int *x, int *y){
  int temp = *x;
  *x = *y;
  *y = temp;
}
int partition(int arr[], int start, int end){
  int pivot_v = arr[end];
  int i = start;
  for(int j = start; j < end; j++){
    if(arr[j] <= pivot_v){
      swap(&arr[i], &arr[j]);
    }
  }
}
```


```

        i++;
    }
}
swap(&arr[i], &arr[end]);
return i;
}
void quicksort_recursion(int arr[], int start, int end){
    if (start < end){
        int pivot_i = partition(arr, start, end);
        quicksort_recursion(arr, start, pivot_i-1);
        quicksort_recursion(arr, pivot_i+1, end);
    }
}
void quicksort(int arr[], int n){
    quicksort_recursion(arr, 0, n-1);
}
void print_arr(int arr[], int n){
    int i;
    for(i = 0; i < n; i++){
        printf("%d ", arr[i]);
    }
}
int main(){
    int n, i;
    printf("Enter the size of the array: ");
    scanf("%d", &n);
    int arr[n];
    printf("Enter the elements of the array: ");
    for(i = 0; i < n; i++){
        scanf("%d", &arr[i]);
    }
    quicksort(arr, n);
    printf("The array after sorting: ");
    print_arr(arr, n);
}

```



Output(o) of Quick sort:

 "C:\Users\chand\Downloads\IV SEM\AA\EXP-2\bin\Debug\EXP-2.exe"

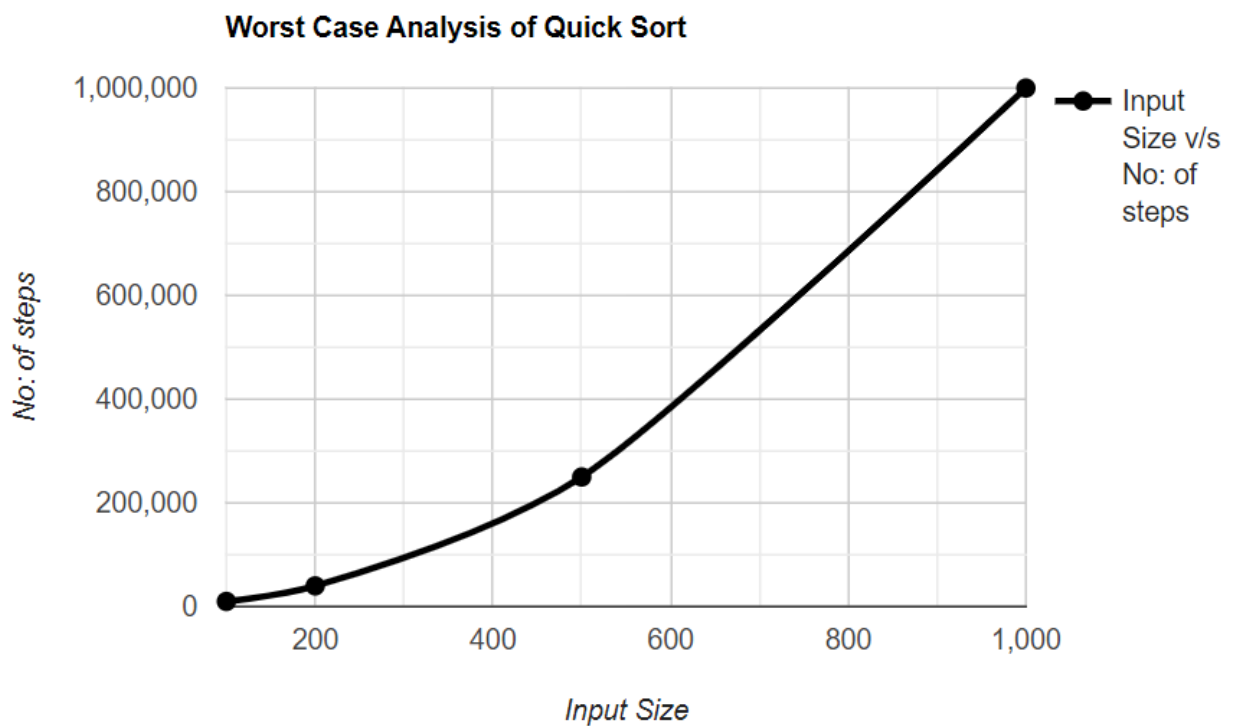
```

Enter the size of the array: 7
Enter the elements of the array: 10 80 30 90 40 50 70
The array after sorting: 10 30 40 50 70 80 90
Process returned 0 (0x0)   execution time : 13.430 s
Press any key to continue.

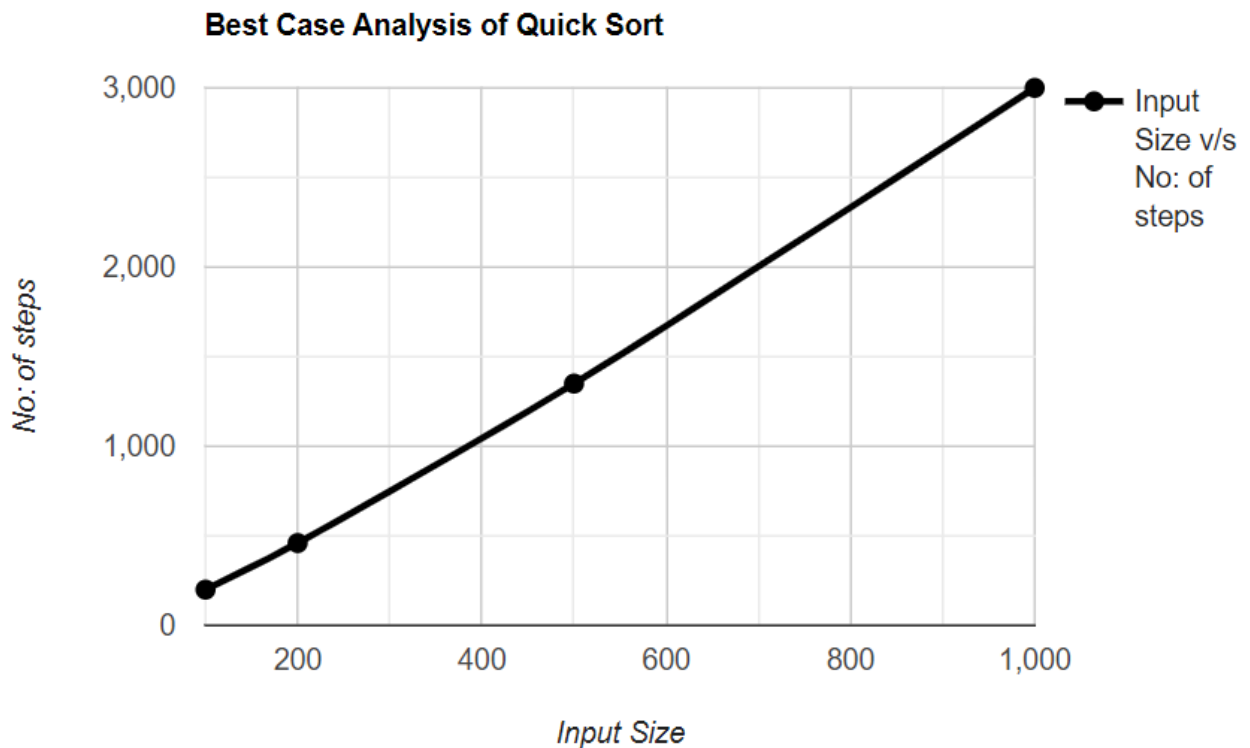
```

Results:**Time Complexity of Quick sort:****Worst Case Analysis:**

Sr. No.	Input size	No: of steps from Algorithm analysis	No: of steps from Theoretical analysis
1.	100	10000	10000
2.	200	40000	40000
3.	500	250000	250000
4.	1000	1000000	1000000

GRAPH:**Best Case Analysis:**

Sr. No.	Input size	No: of steps from Algorithm analysis	No: of steps from Theoretical analysis
1.	100	200	200
2.	200	460	460
3.	500	1350	1350
4.	1000	3000	3000

GRAPH:**Conclusion (Based on the observations):**

Quick-sort is a powerful sorting algorithm with favorable average-case time complexity, especially for large datasets. Careful consideration of pivot selection and potential mitigation of worst-case scenarios through randomized techniques contribute to its successful implementation.

Outcome: Analyze time and space complexity of basic algorithms

References:

1. Richard E. Neapolitan, " Foundation of Algorithms ", 5th Edition 2016, Jones & Bartlett Students Edition
2. Harsh Bhasin , " Algorithms : Design & Analysis", 1st Edition 2013, Oxford Higher education, India
3. T.H. Cormen ,C.E. Leiserson,R.L. Rivest, and C. Stein, " Introduction to algorithms", 3rd Edition 2009, Prentice Hall India Publication
4. Jon Kleinberg, Eva Tardos, " Algorithm Design", 10th Edition 2013, Pearson India Education Services Pvt. Ltd.