3. Problem Statement: Quick Sort

**Problem Analysis:**

Quick sort is a highly efficient sorting algorithm and is based on partitioning of array of data into smaller arrays. A large array is partitioned into two arrays one of which holds values smaller than the specified value, say pivot, based on which the partition is made and another array holds values greater than the pivot value.

Quick sort partitions an array and then calls itself recursively twice to sort the two resulting subarrays. This algorithm is quite efficient for large-sized data sets as its average and worst-case complexity are of Ο(n2), where n is the number of items.

Following are the steps involved in quick sort algorithm:

*1.* After selecting an element as pivot, which is the last index of the array in our case, we divide the array for the first time.

*2.* In quick sort, we call this partitioning. It is not simple breaking down of array into 2 subarrays, but in case of partitioning, the array elements are so positioned that all the elements smaller than the pivot will be on the left side of the pivot and all the elements greater than the pivot will be on the right side of it.

*3.* And the pivot element will be at its final sorted position.

*4.* The elements to the left and right, may not be sorted.

*5.* Then we pick subarrays, elements on the left of pivot and elements on the right of pivot, and we perform partitioning on them by choosing a pivot in the subarrays.

**Algorithm:**

Algorithm Partition (arr[], p, r)

{

*// pivot (Element to be placed at right position*

pivot = arr[r];

i = (p - 1) *// Index of smaller element*

for (j = p; j <= r- 1; j++)

{

*// If current element is smaller than or equal to pivot*

if (arr[j] <= x)

{

i++; *// increment index of smaller element*

tmp = arr[j];

arr[j] = arr[i];

arr[i] = tmp;

}

}

tmp = arr[i+1];

arr[i+1] = arr[r];

arr[r] = tmp;

return i+1;

}

Algorithm quickSort(arr[], p, r)

{

if (p < r)

{

*/\* q is partitioning index, arr[p] is now at right place \*/*

q = partition(arr, p, r);

quickSort(arr, p, q - 1); // Before q

quickSort(arr, q + 1, r); // After q

}

}

**Source Code:**

#include<iostream>

using namespace std;

int PARTITION(int arr[], int p, int r)

{

int x = arr[r];

int i = p-1;

for(int j = p; j <= r-1; j++)

{

if(arr[j]<=x)

{

i++;

int tmp = arr[j];

arr[j] = arr[i];

arr[i] = tmp;

}

}

int tmp = arr[i+1];

arr[i+1] = arr[r];

arr[r] = tmp;

return i+1;

}

int quick(int arr[], int p, int r)

{

if(p<r)

{

int q = PARTITION(arr, p, r);

quick(arr, p, q - 1);

quick(arr, q + 1, r);

}

}

int main()

{

int n;

cout<<"Enter array size:"<<endl; cin>>n;

int ar[n+1]; cout<<"Enter array elements:"<<endl;

for(int i=1;i<=n;i++)

cin>>ar[i];

quick(ar,1,n); cout<<"Sorted array elements:"<<endl;

for(int i=1;i<=n;i++)

cout<<ar[i]<<' ';

}

**Sample Input:**

Enter array size:

7

Enter array elements:

99 55 11 88 22 77 33

**Sample Output:**

Sorted array elements:

11 22 33 55 77 88 99