**Experiment Number: 8**

**Title: Quick sort**

**Problem Statement:** Write a C++ program to arrange the given set of numbers in ascending order using Quick sort.

**Algorithm:**

* Pick the first element as pivot element.
* Now all the elements smaller than pivot are placed at its left while elements bigger are placed at right.
* Repeat the above two steps recursively for both halves.

**Code:**

#include <iostream>

using namespace std;

void quickSort(int[],int,int);

int partition(int[],int,int);

int main()

{

int a[50],n,i;

cout<<"How many elements to be sorted? :";

cin>>n;

cout<<"\nEnter the elements :";

for(i=0;i<n;i++) //To read the array elements

cin>>a[i];

quickSort(a,0,n-1);

cout<<"\nArray after sorting : ";

for(i=0;i<n;i++) //To print the sorted array

cout<<" "<<a[i]<<" ";

return 0;

}

void quickSort(int a[],int low,int up)

{

int j,i;

if(low<up)

{

j=partition(a,low,up);

// The following 3 blocks of code is to observe the partitioning

// and intermediate results

cout<<"\n";

for(i=low;i<=j-1;i++) //To print the array

cout<<" "<<a[i]<<" ";

cout<<"\n";

cout<<"\n";

for(i=j+1;i<=up;i++) //To print the array

cout<<" "<<a[i]<<" ";

cout<<"\n";

cout<<"\n";

for(i=0;i<=7;i++) //To print the array

cout<<" "<<a[i]<<" ";

cout<<"\n";

quickSort(a,low,j-1);

quickSort(a,j+1,up);

}

}

int partition(int a[],int low,int up)

{

int piv,i,j,temp;

piv=a[low]; //piv is the element whose final position is sought

i=low;

j=up+1;

cout<< "\n piv="<<piv<<"\n";

do

{

do

i++; // move up the array

while(a[i]< piv && i<=up);

do

j--; // move dpwn the array

while(piv<a[j]);

if(i<j)

{

temp=a[i];

a[i]=a[j];

a[j]=temp;

}

}while (i<j);

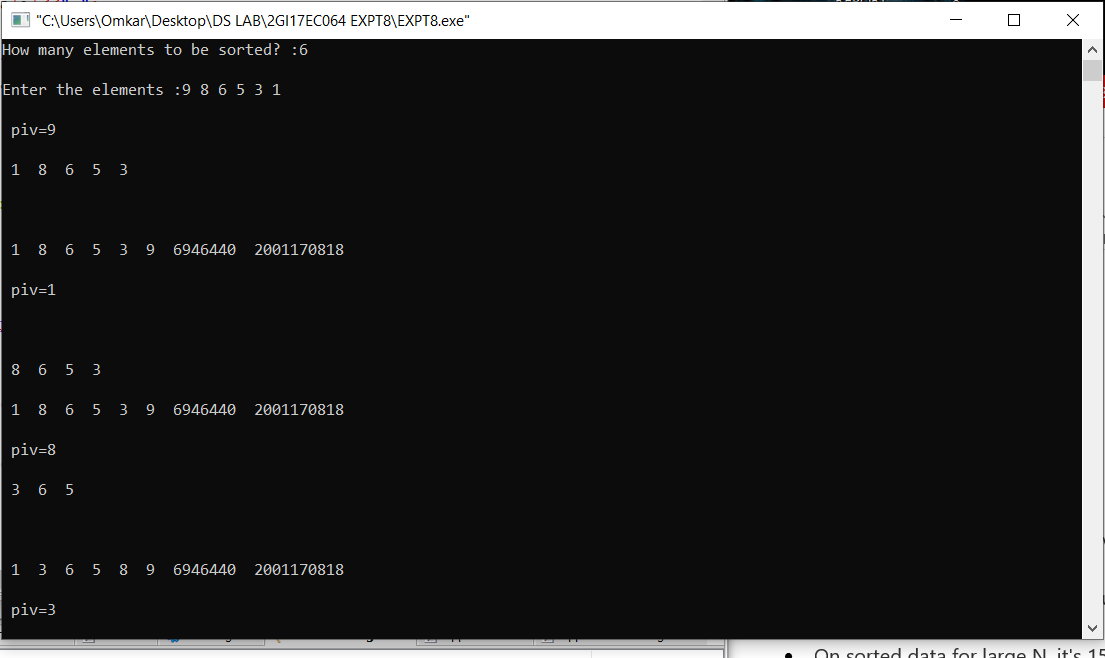
a[low]=a[j];

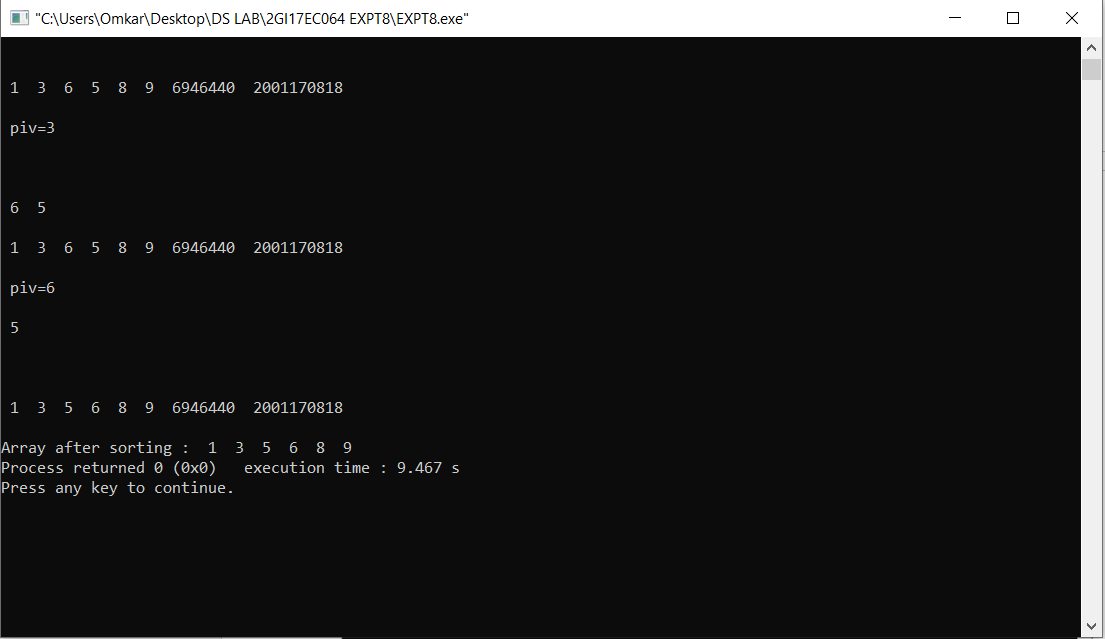
a[j]=piv;

return(j);

}

**Output:**





**ANALYSIS(LIMITATIONS):**

* Quick Sort is not adaptive.
* On sorted data, Quick Sort devolves to O(n^2) performance.
* It is not stable. It is less than ideal for highly repetitive data.
* It's not always fast or ideal, although for the AVERAGE case, it's acceptably efficient in time and memory.
* On sorted data for large N, it's 15–16 times SLOWER than Bubble Sort