NAME: SUSHANT GAWADE

ROLL NO.: 118

BATCH: B3

ASSIGNMENT 2

CODE:

```
#include <iostream>
#include <bits/stdc++.h>
#include <chrono>
using namespace std::chrono;
using namespace std;

class quicksort
{
	public:
		int find_med_index(int n);
		void using_partition(int arr[],int n);
		void using_med_of_med(int arr[],int n);
		void quicksortbypart(int arr[],int low,int high);
		void quicksortbymedofmed(int arr[],int low,int high);
		int partition(int arr[],int low,int high);
		int partition_mom(int arr[],int low,int high);
		int med_of_med(int arr[],int n);
};

int quicksort::partition_mom(int arr[], int low, int high)
{
		int n = high-low+1;
```

```
 int pivot = med_of_med(arr,n);
 int i;
 for (i=low; i<high; i++)
    if (arr[i] == pivot)
       break;

                //swap pivot and last element
                int temp=arr[i];
                arr[i]=arr[high];
                arr[high]=temp;

int k=low-1;          //keeps track of elements that are less than pivot

 for(int j=low;j<high;j++)
 {
    if(arr[j]<pivot)
    {
       k++;
       //swap
       int temp=arr[j];
       arr[j]=arr[k];
       arr[k]=temp;
    }
 }
 //swap wth pivot
 k++;
 temp=arr[k];
 arr[k]=pivot;
 arr[high]=temp;
```

```
    return k;
}
int quicksort::med_of_med(int arr[],int n)
{
        int no_of_subarr=0;
        int r=find_med_index(n);
        int temp[n];
        for(int i=0;i<n;i++)
                temp[i]=arr[i];

        while(1)
        {
                ///Find no of subarrays_/
                if(n%5==0)
                        no_of_subarr=n/5;
                else
                        no_of_subarr=(n/5)+1;

                ///_For Splitting array into subarrays_/
                int cnt=0;
                int subarr[no_of_subarr][5]={0};

                        for(int i=0;i<no_of_subarr;i++)
                        {
                                for(int j=0;j<5;j++)
                                {
                                        if(cnt<n)
                                        {
                                                subarr[i][j]=temp[cnt];
```

```
                                cnt++;
                        }
                        else
                                break;
                }
        }

///__sorting the sublists__/
for(int i=0;i<no_of_subarr;i++)
{
        if(i!=0 && i==no_of_subarr-1 && n%5!=0)
                sort(subarr[no_of_subarr-1],subarr[no_of_subarr-1] + (n%5));
        else
                sort(subarr[i],subarr[i] + 5);
}

//__Medians of median list_
int M[no_of_subarr];
for(int i=0;i<no_of_subarr;i++)
{
        int len=sizeof(subarr[i])/sizeof(int);
        if(i!=0 && i==no_of_subarr-1 && n%5!=0)
                len=n%5;

        int med=find_med_index(len);
        M[i]=subarr[i][med-1];
}

//Breaking condition
```

```
if((sizeof(M)/sizeof(int))==1)
{
	return M[0];
}

sort(M,M+no_of_subarr);		//sorting median array

//_finding new pivot
int med=find_med_index(no_of_subarr);
int pivot=M[med-1];

int left[n],right[n],li=0,ri=0;
for(int i=0;i<n;i++)
{
	if(temp[i]<pivot)
	{
		left[li]=temp[i];li++;
	}
	else if(temp[i]>pivot)
	{
		right[ri]=temp[i];ri++;
	}
}

int k=li+1;

if(r<k)
{
	for(int i=0;i<li;i++)
```

```
                    temp[i]=left[i];
                n=li;
            }
            else
            {
                for(int i=0;i<ri;i++)
                    temp[i]=right[i];
                n=ri;
            }
        }
}

int quicksort::partition(int arr[],int low,int high)
{
    int i=low-1;
    int pivot=arr[high];

    for(int j=low;j<high;j++)
    {
      if(arr[j]<pivot)
      {
        i++;
        //swap
        int temp=arr[j];
        arr[j]=arr[i];
        arr[i]=temp;
      }
    }
    //swap wth pivot
```

```
        i++;
        int temp=arr[i];
        arr[i]=arr[high];
        arr[high]=temp;
        return i;

    }

void quicksort::quicksortbypart(int arr[],int low,int high)
{
        if(low<high)
        {
                        int pivind=partition(arr,low,high);
                quicksortbypart(arr,low,pivind-1);
                quicksortbypart(arr,pivind+1,high);
                    }
    }

void quicksort::quicksortbymedofmed(int arr[],int low,int high)
{
        if(low<high)
        {
                        int pivind=partition_mom(arr,low,high);
                quicksortbypart(arr,low,pivind-1);
                quicksortbypart(arr,pivind+1,high);
        }
    }

void quicksort::using_partition(int arr[],int n)
```

```
{
   cout<<"Size of arr is "<<n<<endl;
   quicksortbypart(arr,0,n-1);
   for(int i=0;i<n;i++)
      cout<<arr[i]<<" ";
   cout<<endl;
}

void quicksort::using_med_of_med(int arr[],int n)
{
   cout<<"Size of arr is "<<n<<endl;
   quicksortbymedofmed(arr,0,n-1);
   for(int i=0;i<n;i++)
      cout<<arr[i]<<" ";
   cout<<endl;
}
int quicksort::find_med_index(int n)
{
        //find median index
        int med=0;
        if(n%2==0)
                med=(n/2);
        else
                med=(n+1)/2;
        return med;
}

int main()
{
```

```
// Write C++ code here
cout<<"\n------------------------Time Comparison of quick sort by partition and medians of median----------------------------------\n\n";
quicksort obj;
int n;

                cout<<"Enter the array limit:";
                cin>>n;
                int arr[n] ;
                //cout<<sizeof(arr);
                for(int i=0;i<n;i++)
                        arr[i] = rand()%n;

                cout<<"Original array is \n";
                for( int i=0;i<n;i++ )
                        cout<<arr[i]<<" ";
                cout<<endl;

        int arr1[n]={0};

        for(int i=0;i<n;i++)
                arr1[i]=arr[i];

cout<<"Size of arr is "<<n<<endl;
cout<<"1.Quick sort using partition\n";

auto start = high_resolution_clock::now();
obj.using_partition(arr,n);
auto stop = high_resolution_clock::now();// ending time
        auto duration = duration_cast<microseconds>(stop - start);
```

```
        cout<<"Time taken by Quick sort using partition " <<duration.count()<< " ms";


   cout<<"\n\n2.Quick sort using median of medians\n";
   start = high_resolution_clock::now();
   obj.using_med_of_med(arr1,n);
   stop = high_resolution_clock::now();// ending time
        duration = duration_cast<microseconds>(stop - start);
        cout<<"Time taken by Quick sort using median of median " <<duration.count()<< "
ms";


   return 0;
}
```

OUTPUT: