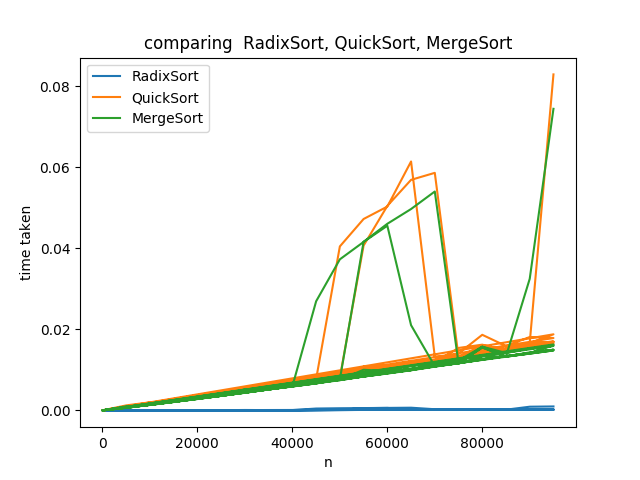
**LAB-1**

**Aim:** Implement and compare the following sorting algorithm Radix Sort Also compare these algorithm with the previous algorithm(quick sort, merge sort)

**Codes:**#include <bits/stdc++.h>  
using namespace std;  
  
void merge(int arr[], int l, int m, int r) {  
 int i, j, k;  
 int n1 = m - l + 1;  
 int n2 = r - m;  
  
 int L[n1], R[n2];  
  
 for (i = 0; i < n1; i++)  
 L[i] = arr[l + i];  
 for (j = 0; j < n2; j++)  
 R[j] = arr[m + 1+ j];  
  
 i = 0;  
 j = 0;  
 k = l;  
 while (i < n1 && j < n2) {  
 if (L[i] <= R[j]) {  
 arr[k] = L[i];  
 i++;  
 }  
 else{  
 arr[k] = R[j];  
 j++;  
 }  
 k++;  
 }  
  
 while (i < n1) {  
 arr[k] = L[i];  
 i++;  
 k++;  
 }  
  
 while (j < n2) {  
 arr[k] = R[j];  
 j++;  
 k++;  
 }  
}  
  
void mergeSort(int arr[], int l, int r) {  
 if (l < r) {  
 int m = l+(r-l)/2;  
 mergeSort(arr, l, m);  
 mergeSort(arr, m+1, r);  
 merge(arr, l, m, r);  
 }  
}  
  
  
int flagquick=0;  
  
void countSort(int arr[], int n, int pos){  
 int count[10];  
 int output[n];  
 for(int i = 0; i <= 9; i++){  
 count[i] = 0;  
 }  
 for(int i = 0 ; i < n ; i++){  
 count[(arr[i]/pos)%10]++;  
 }  
  
  
 for(int i = 1; i <= 9; i++){  
 count[i] = count[i]+ count[i-1];  
 }  
  
 for(int i = n-1; i >=0; i--){  
 output[count[ (arr[i]/pos)%10 ] - 1] = arr[i];  
 count[ (arr[i]/pos)%10 ]--;  
 }  
  
 for(int i = 0; i < n; i++)  
 arr[i] = output[i];  
}  
  
void radixSort(int arr[], int n){  
 int m = arr[0];  
 for(int i = 1; i < n; i++){  
 if(arr[i] > m){  
 m = arr[i];  
 }  
 }  
}  
  
  
int partition (int arr[], int l, int h) {  
 int pivot = arr[h];  
 int i = (l - 1);  
  
 for (int j = l; j <= h - 1; j++) {  
 if (arr[j] < pivot) {  
 i++;  
 swap(arr[i], arr[j]);  
 }  
 flagquick+=1;  
 }  
 swap(arr[i + 1], arr[h]);  
 return (i + 1);  
}  
  
void quickSort(int arr[], int l, int h) {  
 if (l < h) {  
 int pi = partition(arr, l, h);  
  
 quickSort(arr, l, pi - 1);  
 quickSort(arr, pi + 1, h);  
 }  
}  
  
void writeIt1(int n, double t) {  
 fstream fout;  
 fout.open("report1.csv", ios::out | ios::app);  
 // Insert the data to file  
 fout << n << ", " << t << "\n";  
}  
  
void writeIt2(int n, double t) {  
 fstream fout;  
 fout.open("report2.csv", ios::out | ios::app);  
 // Insert the data to file  
 fout << n << ", " << t << "\n";  
}  
  
void writeIt3(int n, double t) {  
 fstream fout;  
 fout.open("report3.csv", ios::out | ios::app);  
 // Insert the data to file  
 fout << n << ", " << t << "\n";  
}  
  
int main() {  
 for(int n=100;n<=100000;n+=5000){  
 int arr1[n];  
 int arr2[n];  
 int arr3[n];  
 for(int i=0;i<n;i++) {  
 int temp=rand();  
 arr2[i]=temp;  
 arr1[i]=temp;  
 arr3[i]=temp;  
 }  
 clock\_t start, end;  
 start = clock();  
 quickSort(arr1, 0, n-1);  
 end = clock();  
 double time\_taken = double(end - start) / double(CLOCKS\_PER\_SEC);  
 writeIt2(n, time\_taken);  
  
 start = clock();  
 mergeSort(arr2, 0, n-1);  
 end = clock();  
 time\_taken = double(end - start) / double(CLOCKS\_PER\_SEC);  
 writeIt3(n, time\_taken);  
  
 start = clock();  
 radixSort(arr3, n);  
 end = clock();  
 time\_taken = double(end - start) / double(CLOCKS\_PER\_SEC);  
 writeIt1(n, time\_taken);  
 }  
  
 return 0;  
}

**Output:**



**LAB-1**

**Aim:** Implement and compare the following sorting algorithm Radix Sort Also compare these algorithm with the previous algorithm(quick sort, merge sort)

**Codes:**#include <bits/stdc++.h>  
using namespace std;  
  
void merge(int arr[], int l, int m, int r) {  
 int i, j, k;  
 int n1 = m - l + 1;  
 int n2 = r - m;  
  
 int L[n1], R[n2];  
  
 for (i = 0; i < n1; i++)  
 L[i] = arr[l + i];  
 for (j = 0; j < n2; j++)  
 R[j] = arr[m + 1+ j];  
  
 i = 0;  
 j = 0;  
 k = l;  
 while (i < n1 && j < n2) {  
 if (L[i] <= R[j]) {  
 arr[k] = L[i];  
 i++;  
 }  
 else{  
 arr[k] = R[j];  
 j++;  
 }  
 k++;  
 }  
  
 while (i < n1) {  
 arr[k] = L[i];  
 i++;  
 k++;  
 }  
  
 while (j < n2) {  
 arr[k] = R[j];  
 j++;  
 k++;  
 }  
}  
  
void mergeSort(int arr[], int l, int r) {  
 if (l < r) {  
 int m = l+(r-l)/2;  
 mergeSort(arr, l, m);  
 mergeSort(arr, m+1, r);  
 merge(arr, l, m, r);  
 }  
}  
  
  
int flagquick=0;  
  
void countSort(int arr[], int n, int pos){  
 int count[10];  
 int output[n];  
 for(int i = 0; i <= 9; i++){  
 count[i] = 0;  
 }  
 for(int i = 0 ; i < n ; i++){  
 count[(arr[i]/pos)%10]++;  
 }  
  
  
 for(int i = 1; i <= 9; i++){  
 count[i] = count[i]+ count[i-1];  
 }  
  
 for(int i = n-1; i >=0; i--){  
 output[count[ (arr[i]/pos)%10 ] - 1] = arr[i];  
 count[ (arr[i]/pos)%10 ]--;  
 }  
  
 for(int i = 0; i < n; i++)  
 arr[i] = output[i];  
}  
  
void radixSort(int arr[], int n){  
 int m = arr[0];  
 for(int i = 1; i < n; i++){  
 if(arr[i] > m){  
 m = arr[i];  
 }  
 }  
}  
  
  
int partition (int arr[], int l, int h) {  
 int pivot = arr[h];  
 int i = (l - 1);  
  
 for (int j = l; j <= h - 1; j++) {  
 if (arr[j] < pivot) {  
 i++;  
 swap(arr[i], arr[j]);  
 }  
 flagquick+=1;  
 }  
 swap(arr[i + 1], arr[h]);  
 return (i + 1);  
}  
  
void quickSort(int arr[], int l, int h) {  
 if (l < h) {  
 int pi = partition(arr, l, h);  
  
 quickSort(arr, l, pi - 1);  
 quickSort(arr, pi + 1, h);  
 }  
}  
  
void writeIt1(int n, double t) {  
 fstream fout;  
 fout.open("report1.csv", ios::out | ios::app);  
 // Insert the data to file  
 fout << n << ", " << t << "\n";  
}  
  
void writeIt2(int n, double t) {  
 fstream fout;  
 fout.open("report2.csv", ios::out | ios::app);  
 // Insert the data to file  
 fout << n << ", " << t << "\n";  
}  
  
void writeIt3(int n, double t) {  
 fstream fout;  
 fout.open("report3.csv", ios::out | ios::app);  
 // Insert the data to file  
 fout << n << ", " << t << "\n";  
}  
  
int main() {  
 for(int n=100;n<=100000;n+=5000){  
 int arr1[n];  
 int arr2[n];  
 int arr3[n];  
 for(int i=0;i<n;i++) {  
 int temp=rand();  
 arr2[i]=temp;  
 arr1[i]=temp;  
 arr3[i]=temp;  
 }  
 clock\_t start, end;  
 start = clock();  
 quickSort(arr1, 0, n-1);  
 end = clock();  
 double time\_taken = double(end - start) / double(CLOCKS\_PER\_SEC);  
 writeIt2(n, time\_taken);  
  
 start = clock();  
 mergeSort(arr2, 0, n-1);  
 end = clock();  
 time\_taken = double(end - start) / double(CLOCKS\_PER\_SEC);  
 writeIt3(n, time\_taken);  
  
 start = clock();  
 radixSort(arr3, n);  
 end = clock();  
 time\_taken = double(end - start) / double(CLOCKS\_PER\_SEC);  
 writeIt1(n, time\_taken);  
 }  
  
 return 0;  
}

**Output:**

