ROLL NO: 20MIT3308

NAME: JAYNIL PATEL

BRANCH: DATA SCIENCE

COURSE: COMPLEXITY THEORY AND ALGORITHMS (3CS1109)

TOPIC: PRACTICAL 3

AIM

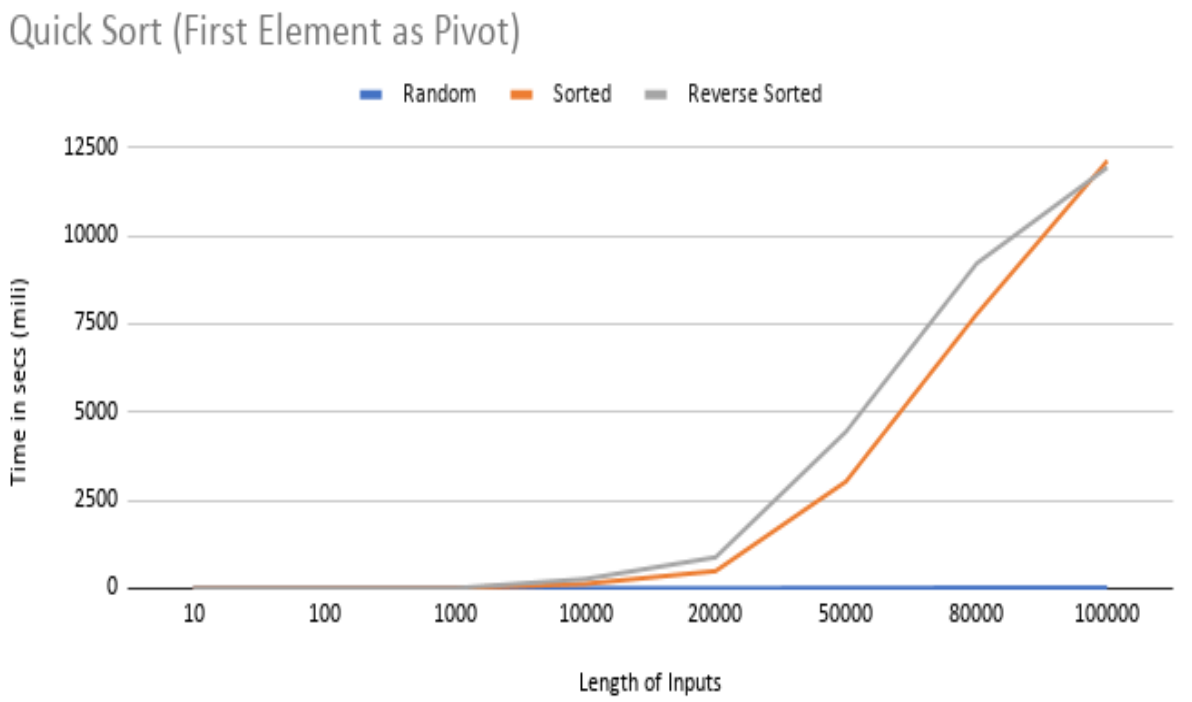
Implement quick sort algorithm with the following ways to select pivot element and give your observations for the same.

1. Always pick first element as pivot.
2. Always pick last element as pivot (implemented below)
3. Pick a random element as pivot.
4. Pick median as pivot

QUICK SORT

(First Element As Pivot)

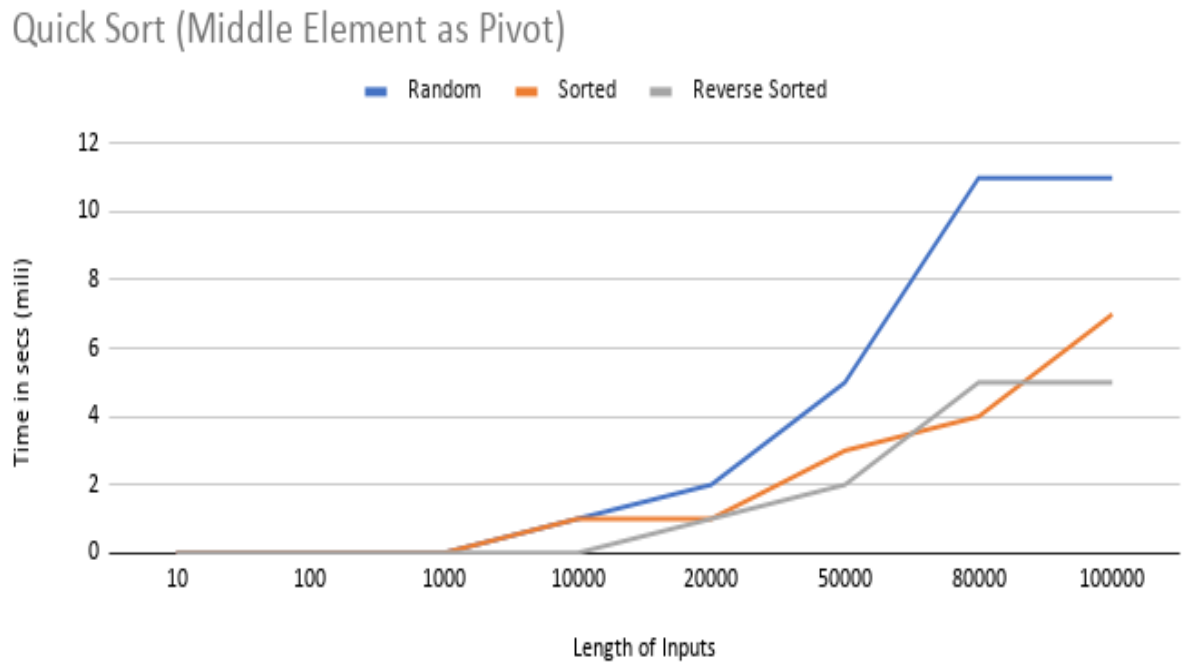
Length	Random	Sorted	Reverse Sorted
10	0	0	0
100	0	0	0
1000	0	2	3
10000	2	128	264
20000	3	491	882
50000	10	3036	4451
80000	17	7784	9224
100000	20	12136	11945



QUICK SORT

(Middle Element As Pivot)

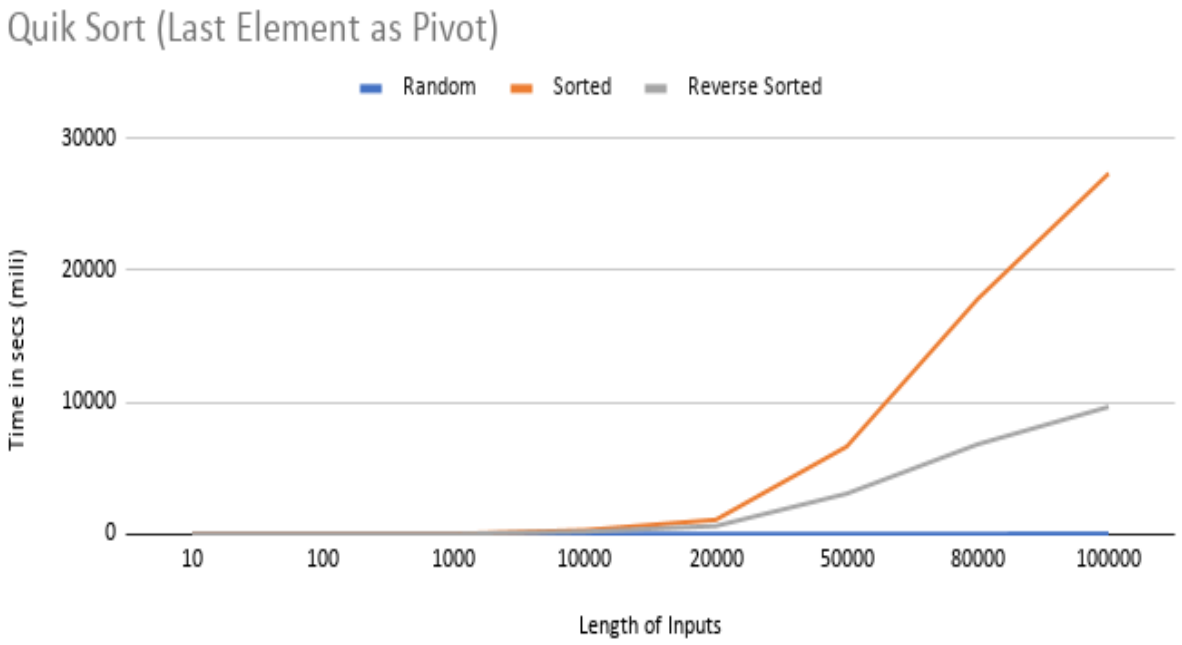
Length	Random	Sorted	Reverse Sorted
10	0	0	0
100	0	0	0
1000	0	0	0
10000	1	1	0
20000	2	1	1
50000	5	3	2
80000	11	4	5
100000	11	7	5



QUICK SORT

(Last Element As Pivot)

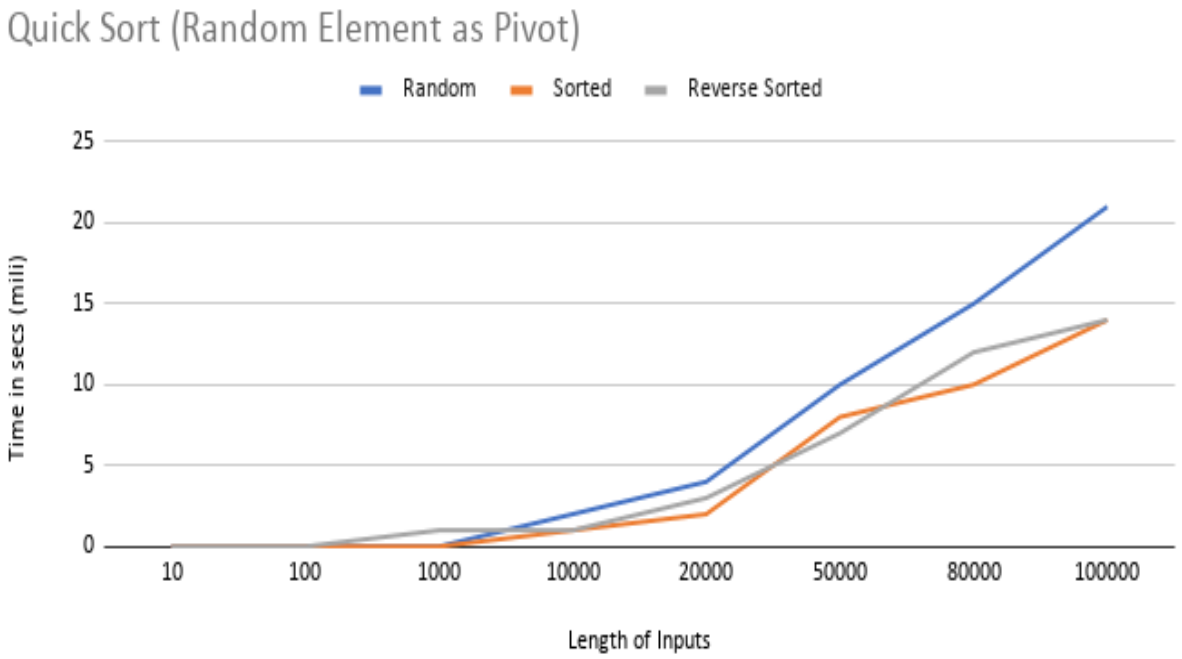
Length	Random	Sorted	Reverse Sorted
10	0	0	0
100	0	1	0
1000	1	3	2
10000	2	284	179
20000	3	1075	603
50000	9	6638	3061
80000	12	17841	6811
100000	15	27374	9653



QUICK SORT

(Random Element As Pivot)

Length	Random	Sorted	Reverse Sorted
10	0	0	0
100	0	0	0
1000	0	0	1
10000	2	1	1
20000	4	2	3
50000	10	8	7
80000	15	10	12
100000	21	14	14



CODE (C++)

```
// C++ program by Jaynil Patel 3308
#include <bits/stdc++.h>
#include<cstdlib>
#include<ctime>
#include<algorithm>
#include <fstream>
#include <iostream>
#include<time.h>
using namespace std;


void swap(int *xp, int *yp)
{
    int temp = *xp;
    *xp = *yp;
    *yp = temp;
}
// partition the array using last element as pivot

int partition (int arr[], int low, int high,int pivot_loc)
{
    int pivot;
    if(pivot_loc == 0)
    {
        //Fisrt pivot
         pivot = arr[low];
         int i = (low + 1);

    for(int j =low + 1; j <= high ; j++ )  {


          if ( arr[ j ] < pivot) {
                 swap (arr[ i ],arr [ j ]);
            i += 1;
        }
   }
   swap ( arr[ low ] ,arr[ i-1 ] ) ;  //put the pivot element in its proper place.
   return i-1;

    }
    else if (pivot_loc == 1)
    {
        //last pivot
         pivot = arr[high];
```

```
      int i = (low - 1);

  for (int j = low; j <= high- 1; j++)
  {

      if (arr[j] <= pivot)
      {
          i++;    // increment index of smaller element
          swap(&arr[i], &arr[j]);
      }
  }
      swap(&arr[i + 1], &arr[high]);
      return (i + 1);
  }
   else if (pivot_loc == 2)
  {
     int i = low, j = high;

    int tmp;

    int pivot = arr[(low + high) / 2];



    while (i <= j) {

          while (arr[i] < pivot)

                i++;

          while (arr[j] > pivot)

                j--;

          if (i <= j) {

                tmp = arr[i];

                arr[i] = arr[j];

                arr[j] = tmp;

                i++;

                j--;

          }

    };



    return i;

  }
  else if (pivot_loc = 3)
```

```
    {
        //Random pivot

        int random = low + rand( )%(high-low +1 ) ;
        swap ( arr[random] , arr[low]) ;
         pivot = arr[low];
         int i = (low + 1);

    for(int j =low + 1; j <= high ; j++ )  {


          if ( arr[ j ] < pivot) {
                 swap (arr[ i ],arr [ j ]);
            i += 1;
        }
   }
   swap ( arr[ low ] ,arr[ i-1 ] ) ;  //put the pivot element in its proper place.
   return i-1;


    }


}

//quicksort algorithm
void quickSort(int arr[], int low, int high,int pivot_loc)
{
    if (low < high)
    {
        //partition the array
        int pivot = partition(arr, low, high,pivot_loc);

        //sort the sub arrays independently
        quickSort(arr, low, pivot - 1,pivot_loc);
        quickSort(arr, pivot + 1, high,pivot_loc);
    }
}

void printArray(int arr[], int n)
{
    int i;
    for (i = 0; i < n; i++)
        cout << arr[i] << " ";
    cout << endl;
}

void reverseArray(int arr[], int n){
   for (int low = 0, high = n - 1; low < high; low++, high--){
      swap(arr[low], arr[high]);
   }
}

int main()
{
```

```
    ofstream quick_low,quick_last,quick_mid,quick_ran;

    quick_low.open("quick_low1.csv");
    quick_mid.open("quick_mid1.csv");
    quick_ran.open("quick_ran1.csv");
    quick_last.open("quick_last1.csv");


    srand((unsigned) time(0));
    int randomNumber_len, randomNumber;
    time_t start_1, end_1,start_2, end_2,start_3, end_3,start_4, end_4;


    int len_arr[] = {10,100,1000,10000,20000,50000,80000,100000};
    //int len_arr[] = {140000};
    int debug_len = 8;


    quick_last<< "Length , Random , Sorted , Reverse Sorted\n";
    quick_low<< "Length , Random , Sorted , Reverse Sorted\n";
    quick_mid<< "Length , Random , Sorted , Reverse Sorted\n";
    quick_ran<< "Length , Random , Sorted , Reverse Sorted\n";


     for (int i = 0; i < debug_len; i++)
     {


            randomNumber_len = len_arr[i];
            int arr[randomNumber_len],arr_q_low[randomNumber_len],arr_q_mid[randomNumber_le
n],arr_q_last[randomNumber_len],arr_q_ran[randomNumber_len];

            cout << "\nLength of  random numbers = " << randomNumber_len << "\n";
            cout<< "Seq Type = Random";


            quick_last << randomNumber_len << ",";
            quick_low << randomNumber_len << ",";
            quick_mid << randomNumber_len << ",";
            quick_ran << randomNumber_len << ",";




            for (int index = 0; index < randomNumber_len; index++)
               {
                   randomNumber = (rand() % (randomNumber_len*10)) + 1;

                   arr_q_low[index]  = randomNumber;
                   arr_q_last[index]  = randomNumber;
                   arr_q_mid[index]  = randomNumber;
                   arr_q_ran[index]  = randomNumber;
               }
```

```
cout << "(Quick_low = ";
clock_t t =clock();
quickSort(arr_q_low, 0, randomNumber_len - 1,0);
t=clock()-t;
double time_taken_4= ((double)t);
cout << fixed << time_taken_4 << setprecision(5) << "sec)\t";
quick_low << fixed << time_taken_4 << setprecision(5) << ",";

cout << "(Quick_mid = ";
 t =clock();
quickSort(arr_q_mid, 0, randomNumber_len - 1,2);
t=clock()-t;
 time_taken_4= ((double)t);
cout << fixed << time_taken_4 << setprecision(5) << "sec)\t";
quick_mid << fixed << time_taken_4 << setprecision(5) << ",";

cout << "(Quick_last = ";
 t =clock();
quickSort(arr_q_last, 0, randomNumber_len - 1,1);
t=clock()-t;
 time_taken_4= ((double)t);
cout << fixed << time_taken_4 << setprecision(5) << "sec)\t";
quick_last << fixed << time_taken_4 << setprecision(5) << ",";

cout << "(Quick_ran = ";
 t =clock();
quickSort(arr_q_ran, 0, randomNumber_len - 1,3);
t=clock()-t;
 time_taken_4= ((double)t);
cout << fixed << time_taken_4 << setprecision(5) << "sec)\t";
quick_ran << fixed << time_taken_4 << setprecision(5) << ",";




cout << "Seq Type = Sorted\n  ";



cout << "(Quick_low = ";

t=clock();
quickSort(arr_q_low, 0, randomNumber_len-1,0);
t=clock()-t;
 time_taken_4= ((double)t);
cout << fixed << time_taken_4 << setprecision(5) << "sec)\t";
quick_low << fixed << time_taken_4 << setprecision(5) << ",";

cout << "(Quick_mid = ";

t=clock();
```

```
quickSort(arr_q_mid, 0, randomNumber_len-1,2);
t=clock()-t;
 time_taken_4= ((double)t);
cout << fixed << time_taken_4 << setprecision(5) << "sec)\t";
quick_mid << fixed << time_taken_4 << setprecision(5) << ",";

cout << "(Quick_last = ";

t=clock();
quickSort(arr_q_last, 0, randomNumber_len-1,1);
t=clock()-t;
 time_taken_4= ((double)t);
cout << fixed << time_taken_4 << setprecision(5) << "sec)\t";
quick_last << fixed << time_taken_4 << setprecision(5) << ",";

cout << "(Quick_ran = ";

t=clock();
quickSort(arr_q_ran, 0, randomNumber_len-1,3);
t=clock()-t;
 time_taken_4= ((double)t);
cout << fixed << time_taken_4 << setprecision(5) << "sec)\t";
quick_ran << fixed << time_taken_4 << setprecision(5) << ",";









reverseArray(arr_q_low,randomNumber_len);
copy_n(arr_q_low,randomNumber_len,arr_q_mid);
copy_n(arr_q_low,randomNumber_len,arr_q_last);
copy_n(arr_q_low,randomNumber_len,arr_q_ran);



cout<< "Seq Type = Reverse Sorted\n  ";







cout << "(Quick_low = ";

t=clock();
quickSort(arr_q_low, 0, randomNumber_len-1,0);
t=clock()-t;
 time_taken_4= ((double)t);
cout << fixed << time_taken_4 << setprecision(5) << "sec)\t";
quick_low << fixed << time_taken_4 << setprecision(5) << "\n";

cout << "(Quick_mid = ";

t=clock();
```

```
                quickSort(arr_q_mid, 0, randomNumber_len-1,2);
                t=clock()-t;
                 time_taken_4= ((double)t);
                cout << fixed << time_taken_4 << setprecision(5) << "sec)\t";
                quick_mid << fixed << time_taken_4 << setprecision(5) << "\n";

                cout << "(Quick_last = ";

                t=clock();
                quickSort(arr_q_last, 0, randomNumber_len-1,1);
                t=clock()-t;
                 time_taken_4= ((double)t);
                cout << fixed << time_taken_4 << setprecision(5) << "sec)\t";
                quick_last << fixed << time_taken_4 << setprecision(5) << "\n";

                cout << "(Quick_ran = ";

                t=clock();
                quickSort(arr_q_ran, 0, randomNumber_len-1,3);
                t=clock()-t;
                 time_taken_4= ((double)t);
                cout << fixed << time_taken_4 << setprecision(5) << "sec)\t";
                quick_ran << fixed << time_taken_4 << setprecision(5) << "\n";




        }



       quick_low.close();
       quick_last.close();
       quick_mid.close();
       quick_ran.close();

    return 0;
}
```

OBSERVATIONS

QuickSort is a Divide and Conquer algorithm. It picks an element as pivot and partitions the given array around the picked pivot. There are many different versions of quickSort that pick pivot in different ways.

1. Always pick first element as pivot.
2. Always pick last element as pivot
3. Pick a random element as pivot.
4. Pick Median/Middle as pivot.

1.Quick Sort using First Element as Pivot

When we use first element as pivot and given inputs are in random order then it takes very less time compare to sorted order and Reverse order.

2.Quick Sort using Middle Element as Pivot

When we use Middle element as pivot and given inputs are in random order then it takes more time and the other two takes almost similar time to run.

3.Quick Sort using Last Element as Pivot

When we use Last element as pivot then also like first element as pivot it takes very less time fo random inputs comapre to sorted and reverse sorted inputs.

4.Quick Sort using Random Element as Pivot

For the random element as pivot the overall time taken by algo is very less compare two other 3 implementation and in this random inputs takes less time comapre to other 2 inputs.