Sardar Patel Institute of Technology
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai-400058, India
(Autonomous College Affiliated to University of Mumbai)

Experiment No.	**2**
Name	**Varada Khadake**
UID No.	**2021300059**
Class & Division	**COMPS A BATCH D**
Date of submission	**27/02/2023**

Aim: To analyze run time of quick sort algorithm by taking random element as pivot.

Observation/Theory:

Best Case Complexity - In Quicksort, the best-case occurs when the pivot element is the middle element or near to the middle element. The best-case time complexity of quicksort is O(n*logn).
Average Case Complexity - It occurs when the array elements are in jumbled order that is not properly ascending and not properly descending. The average case time complexity of quicksort is O(n*logn).
Worst Case Complexity - In quick sort, worst case occurs when the pivot element is either greatest or smallest element. Suppose, if the pivot element is always the last element of the array, the worst case would occur when the given array is sorted already in ascending or descending order. The worst-case time complexity of quicksort is O(n2).

From the graph, it can be observed that the time complexity of quick sort by taking random pivot is less than that of both merge sort and quicksort by taking a fixed pivot.

Algorithm:

partition(arr[],lo,hi)
    pivot=arr[hi]
    i=lo
    for j:= lo to hi-1 do
        if arr[j]<=pivot them
        swap arr[i] with arr[j]
        i=i+1
    swap arr[i] with arr[ji]
    return i

partition_r(arr[],lo,hi)
r=Random Number from lo to hi
Swap arr[r] and arr[hi]
Return partition(arr,lo,hi)

quicksort(arr[],lo,hi)
if lo<hi
p=partition_r(arr,lo,hi)
uicksort(arr,lo,p-1)
quicksort(arr,p+1,hi)

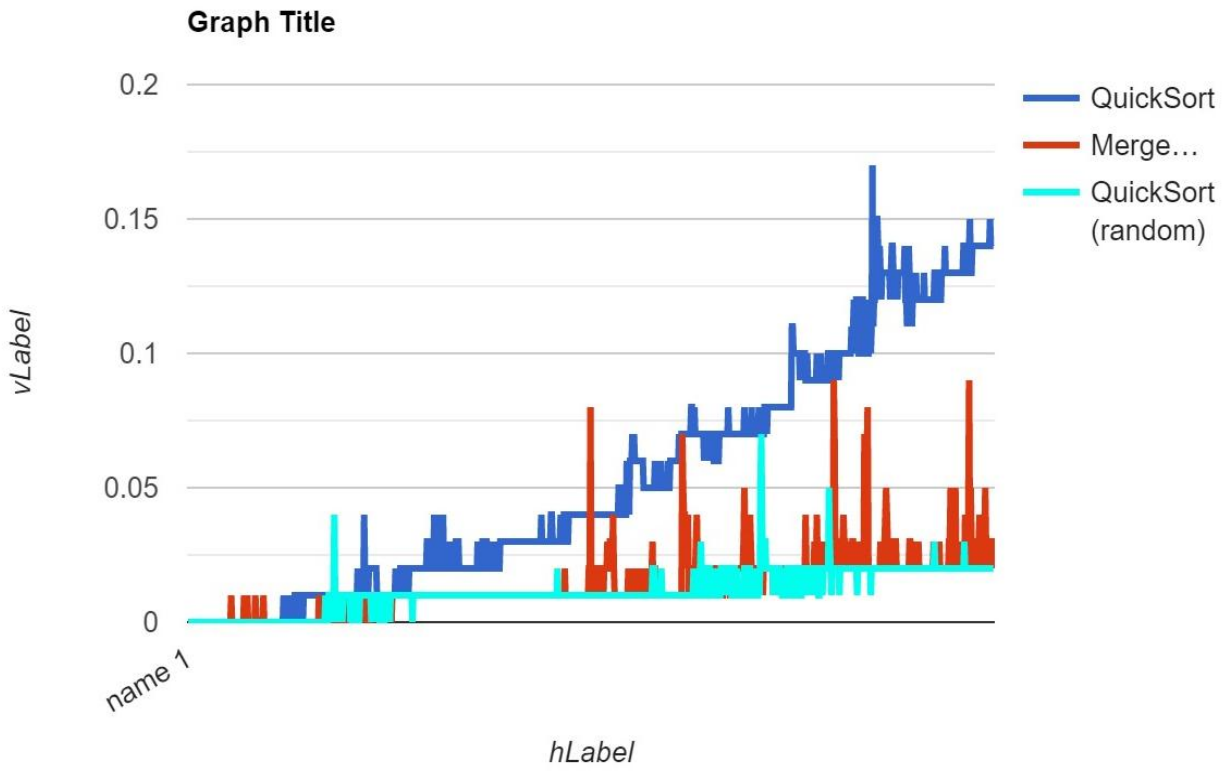
Code:

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

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

    while (1)
    {

        do
        {
            i++;
```

```
        } while (arr[i] < pivot);

        do
        {
            j--;
        } while (arr[j] > pivot);

        if (i >= j)
            return j;

        int temp = arr[i];
        arr[i] = arr[j];
        arr[j] = temp;
    }
}

int partition_r(int arr[], int low, int high)
{
    srand(time(0));
    int random = low + rand() % (high - low);

    int temp = arr[random];
    arr[random] = arr[low];
    arr[low] = temp;

    return partition(arr, low, high);
}

void quickSort(int arr[], int low, int high)
{
    if (low < high)
    {
        int pi = partition_r(arr, low, high);

        quickSort(arr, low, pi);
        quickSort(arr, pi + 1, high);
    }
}

void printArray(int arr[], int n)
{
    for (int i = 0; i < n; i++)
        printf("%d ", arr[i]);
    printf("\n");
}

void getData(int x, int arr[])
{
    for (int i = 0; i < x * 100; i++)
    {
        arr[i] = rand() % 100 + 1;
    }
}

int main()
{
    clock_t start, end;
    double time_req;
    int a[100000];
    int partition(int arr[], int low, int high), partition_r(int arr[], int low, int high);
    void quickSort(int arr[], int low, int high), printArray(int arr[], int n), getData(int x, int arr[]);
    for (int i = 0; i < 1000; i++)
    {

        getData(i + 1, a);
        start = clock();

        quickSort(a, 0, ((i + 1) * 100) - 1);
        end = clock();

        time_req = ((double)(end - start)) / CLOCKS_PER_SEC;
        printf("%.2lf ", time_req);
    }
```

```
    return 0;
}
```

Output:



Conclusion: I observed graphs of quick sort by taking random pivot and by keeping fixed pivot, it can be seen that the time by taking random pivot is the least.