NAME:	Vinit Madhyan
UID:	2021300070
SUBJECT	DAA
EXPERIMENT NO:	2b
AIM:	Understanding more concepts regarding quick sort algorithm

Algorithm:	**QUICK SORT ALGORITHM**
	partition (arr[], low, high)
	{ // pivot (Element to be placed at right position)
	pivot = arr[high];
	i = (low – 1) // Index of smaller element and indicates the
	// right position of pivot found so far
	for (j = low; j <= high- 1; j++){
	// If current element is smaller than the pivot
	if (arr[j] < pivot){
	i++; // increment index of smaller element
	swap arr[i] and arr[j]
	}
	}
	swap arr[i + 1] and arr[high])
	return (i + 1)
	}
	quickSort(arr[], low, high) {
	if (low < high) {
	/* pi is partitioning index, arr[pi] is now at right place */
	pi = partition(arr, low, high);
	quickSort(arr, low, pi – 1); // Before pi
	quickSort(arr, pi + 1, high); // After pi
	}

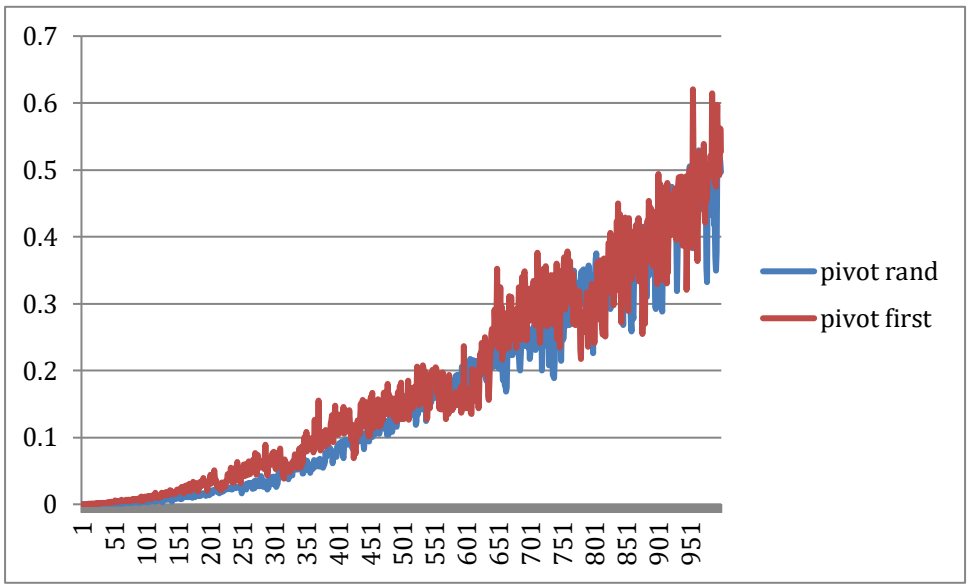
Code:

```
#include <iostream>
#include <fstream>
#include <cstdlib>
#include <ctime>

using namespace std;
int SWAP = 0;
int list[100000];

void read()
{
    ifstream fin("values.txt", ios::binary);
    for (long i = 0; i < 100000; i++)
    {
        fin.read((char *)&list[i], sizeof(int));
    }
    fin.close();
}

long partition(long left, long right)
{
    long rd = (rand() % (right - left + 1)) + left;
    int pivot_element = list[rd];
    int lb = left, ub = right;
    int temp;

    while (left < right)
    {
        while (list[left] <= pivot_element)
            left++;
        while (list[right] > pivot_element)
            right--;
        if (left < right)
        {
            temp = list[left];
            list[left] = list[right];
            list[right] = temp;
            SWAP++;
        }
    }
    list[lb] = list[right];
    list[right] = pivot_element;
    SWAP++;
    return right;
```

```
}

void quickSort(long left, long right)
{
    if (left < right)
    {
        long pivot = partition(left, right);
        quickSort(left, pivot - 1);
        quickSort(pivot + 1, right);
    }
}

int main()
{
    clock_t t1, t2, t3, t4;
    read();

    int num = 10;
    for (int i = 0; i < 1000; i++)
    {

        t3 = clock();
        quickSort(0, num - 1);
        t4 = clock();
        double quicktime = double(t4 - t3) / double(CLOCKS_PER_SEC);
        cout << endl;
        cout << " " << fixed << quicktime << "  " << SWAP;
        num += 100;
    }

    return 0;
}
```

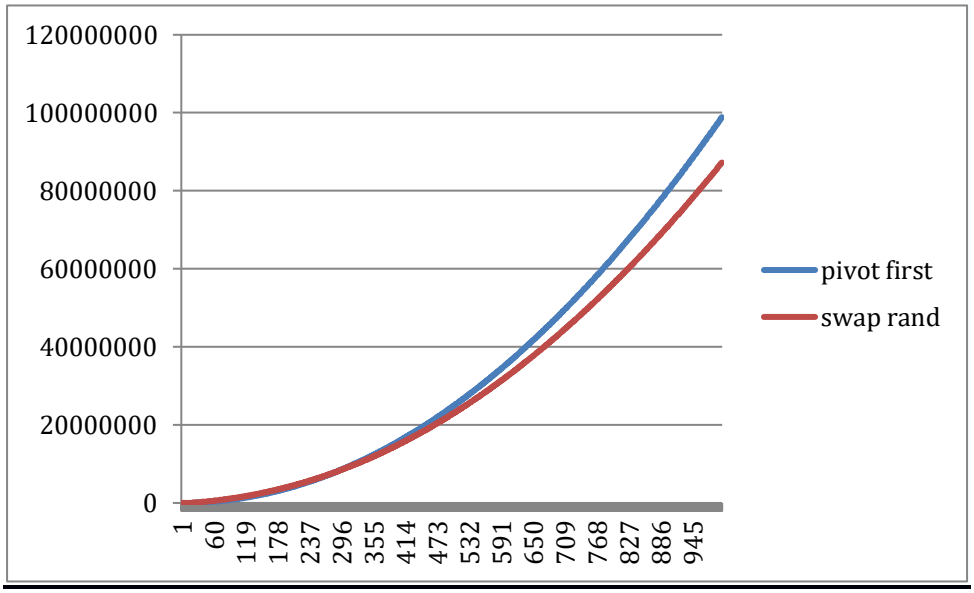
Graphs and Observation:

- **Now considering Different Pivot Positions:**



We can see here that even when different pivot points are considered, the time complexity of rapid sort is nearly the same.

- **Count of swaps considering different pivot positions:**



The number of swaps necessary for quick sort when the pivot is at a random position is fewer than the number of swaps required for quick sort when the pivot is in the end position.

Conclusion:	Thus , I have also understood Quick sort algorithm and their time complexities. And we've also shown observations for various pivots for quick sort algorithms