Practical No:- 03

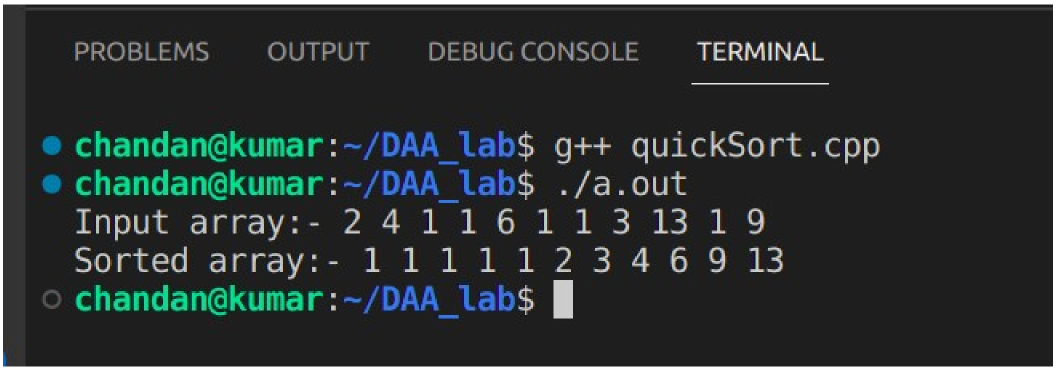
Analyze the time complexity of Quick-sort algorithm.

Code->

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

int partition(int arr[], int start, int end)
{
int pivot = arr[start];
int count = 0;
for (int i = start + 1; i <= end; i++)
{
if (arr[i] <= pivot)
{
count++;
}
}
// place pivot at right position
int pivotIndex = start + count;
swap(arr[start], arr[pivotIndex]);

// handle leftPart amd right part
int i = start, j = end;
while (i < pivotIndex && j > pivotIndex)
{
while (arr[i] < pivot)
{
i++;
}
while (arr[j] > pivot)
{
j--;
}
if (i < pivotIndex && j > pivotIndex)
{
swap(arr[i], arr[j]);
i++;
j--;
}
}
return pivotIndex;
}
void quickSort(int arr[], int start, int end)
{
// Base case
if (start >= end)
return;

// Partition
int p = partition(arr, start, end);
```

```
// Left part sorting
quickSort(arr, start, p - 1);
// Right part sorting
quickSort(arr, p + 1, end);
}
int main()
{
int arr[] = {2, 4, 1, 1, 6, 1, 1, 3, 13, 1, 9};
int size = sizeof(arr) / sizeof(arr[0]);
cout << "Input array:- ";
for (int i = 0; i < size; i++)
{
cout << arr[i] << " ";
}
cout << endl;
quickSort(arr, 0, size - 1);
cout << "Sorted array:- ";
for (int i = 0; i < size; i++)
{
cout << arr[i] << " ";
}
cout << endl;

return 0;
}
```

Output:-



Analysis of Quick Sort algorithm:-

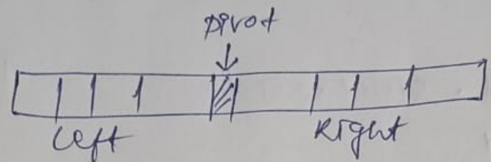
-: Analysis of Quick-sort algorithm:-

Algorithm :-

```
Quicksort (Array A , start, end)  ——  T(n)
{ if start < end
  then q ← Partition (A, start, end)  ——  bn.
    Quick sort (A, start, q-1)  ——  T(n/2)
    Quicksort (A, q+1, end)  ——  T(n/2)
}
Partition (A, start, end)
{ pivot = A[start]
  count = 0
  do { count ++
     } while (A[i] <= pivot)

  pivotIndex = start + count
  swap (A[start], A[pivotIndex])
  int i = start , j = end
  do { i++ } while (A[i] < pivot)
  do ( j++) while (A[j] > pivot)
  swap (A[i], A[j])
      i++;
      j--;
  return pivotIndex
}
```

Case-1

Average case :- The elements are divided approximately half in no by pivot.



$$T(n) = \underbrace{T\left(\tfrac{n}{2}\right)}_{\text{First recursive call}} + \underbrace{T(n/2)}_{\text{second recursive call}} + bn$$

bn ↳ partitioning & some comparisions.

Partitoning $\propto n$
$= \underbrace{b}_{\text{constant}}n$

$$T(n) = 2\,T(n/2) + bn$$

$$T(n/2) = 2T(n/4) + b(n/2)$$

$$T(n) = 2\left[2T(n/4) + b(n/2)\right] + bn$$

$$= 4T\left(\tfrac{n}{4}\right) + \cancel{2}\frac{bn}{\cancel{2}} + bn$$

$$T(n) = 4T\left(\tfrac{n}{4}\right) + 2bn$$

$$\vdots$$

$$T(n) = 2^k\,T\left(\tfrac{n}{2^k}\right) + k\,bn$$

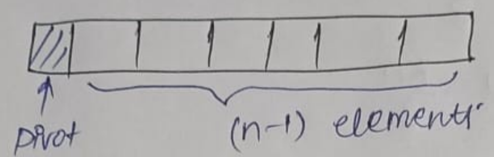
$n = 2^k$ $\quad (k = \log_2 n)$
let $n = 4$
$2^2 = 4$
$2^k = 4$ $\quad \underline{k = 2}$

$[T(1) = a$ (constant)$)$

$$T(n) = 2^2\,T(1) + 2bn$$
$$= 2^k * a + kbn$$
$$= 2^k * a + kbn \qquad \because k = \log_2 n$$
$$= a * n + \log_2 n * b * n.$$
$$= \boxed{O(n\log n)}$$

case-2

worst case :- The pivot divides the array in such a way that pivot is at one side and in the another side rest of the elements are there.



$$T(n) = \underset{\downarrow \atop a\ (\text{constant})}{T(1)} + T(n-1) + bn \hookrightarrow \text{for partitioning}$$

$$T(n) = T(n-1) + bn$$

$$T(n-1) = T(n-2) + b(n-1)$$

$$T(n) = [T(n-2) + b(n-1)] + bn$$

$$T(n-2) = T(n-3) + b(n-2)$$

$$T(n) = [T(n-3) + b(n-2)] + b(n-1) + bn$$

upto k times

$$T(n) = T(n-k) + b(n-(k-1)) + b(n-(k-2)) + \cdots + b(n-1) + bn$$

assume $n-k=0$

$n=k$.

$$T(n) = T(n-n) + b(n-(n-1)) + b(n-(n-2)) + \cdots + b(n-1) + bn.$$

$$T(n) = T(0) + b(1) + b(2) + \cdots + b(n-1) + bn$$

$T(0) = 1$ (say).

$$T(n) = 1 + b[1 + 2 + 3 + \cdots + n-1 + n]$$

$$= 1 + b\left[\frac{n(n+1)}{2}\right]$$

$$= 1 + b\left[\frac{n^2 + n}{2}\right]$$

$$= 1 + \frac{bn^2}{2} + \frac{bn}{2}$$

By ignoring the lower order terms and constants

$$T(n) = O(n^2)$$

In worst case Quick-sort algorithm will of $\boxed{O(n^2)}$ complexity.