DESIGN AND ANALYSIS OF ALGORITHMS

LAB ASSIGNMAENT

Quick Sort Implementation and Time Complexity Analysis

Submitted BY:

Gaurav Yadav

11911038

CSE

Quick Sort Implementation and Time Complexity Analysis

❖Source Code:

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 100
void random_shuffle(int arr[])
{
   srand(time(NULL));
   int i, j, temp;
   for (i = MAX - 1; i > 0; i--)
   {
      j = rand()%(i + 1);
      temp = arr[i];
      arr[i] = arr[j];
      arr[j] = temp;
   }
}

void swap(int *a, int *b)
{
   int temp;
   temp = *a;
   *a = *b;
   *b = temp;
}
int partion(int arr[], int p, int r)
{
   int pivotIndex = p + rand()%(r - p + 1);
   int pivot;
   int i = p - 1;
   int j;
   pivot = arr[pivotIndex];
   swap(&arr[pivotIndex], &arr[r]);
   for (j = p; j < r; j++)
   {
      if (arr[j] < pivot)
      {
         i++;
         swap(&arr[i], &arr[j]);
      }

   }
   swap(&arr[i+1], &arr[r]);
   return i + 1;
}
```

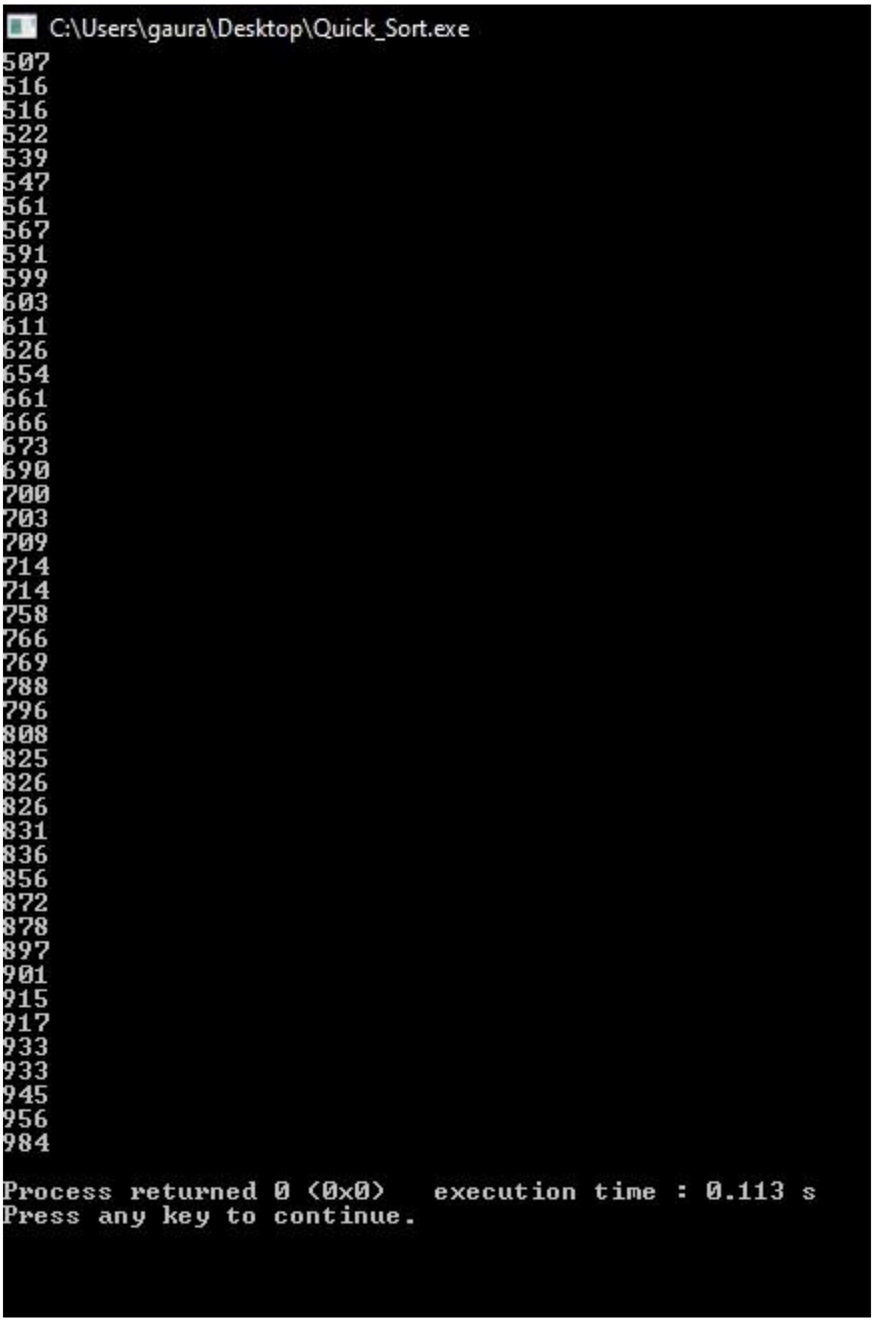
Quick Sort Implementation and Time Complexity Analysis

```
void quick_sort(int arr[], int p, int q)
{
   int j;
   if (p < q)
   {
      j = partion(arr, p, q);
      quick_sort(arr, p, j-1);
      quick_sort(arr, j+1, q);
   }
}
int main()
{
   int i;
   int arr[MAX];
   for (i = 0; i < MAX; i++)
      arr[i] =(rand()%901)+100;
   random_shuffle(arr);
   quick_sort(arr, 0, MAX-1);
   for (i = 0; i < MAX; i++)
       printf("%d \n", arr[i]);
   return 0;
}
```

❖ Output:

```
C:\Users\gaura\Desktop\Quick_Sort.exe
103
109
127
130
130
133
140
141
141
143
150
166
167
173
190
202
223
227
231
232
237
253
259
268
270
279
288
314
315
326
329
345
348
380
388
392
392
398
399
399
402
419
421
422
422
423
432
443
455
471
473
479
485
504
507
516
516
522
539
547
561
567
```

Quick Sort Implementation and Time Complexity Analysis



❖Time Complexity Analysis:

Quick Sort Time complexity Analysis

Gaurav yadav
11911038
CSE

→ Time taken by quick sort can be written as

$$T(n) = \underbrace{T(k) + T(n-k-1)}_{\text{for recursive calls}} + \underbrace{\theta(n)}_{\text{for partition process.}}$$

- worst case: worst case occurs when the partition process always picks greatest or smallest element as pivot. So,

$$T(n) = T(0) + T(n-0-1) + \theta(n)$$
$$= T(n-1) + \theta(n)$$

$$T(n-1) = T(n-2) + \theta(n-1)$$

Similarly

$$T(n) = T(n-r) + r\,\theta(n)$$

So, for $T(1) = 1$,

$$n-r = 1 \Rightarrow n-1 = r$$

$$\Rightarrow T(n) = 1 + (n-1)\,\theta(n)$$
$$\Rightarrow T(n) \simeq \theta(n^2)$$

Quick Sort Implementation and Time Complexity Analysis

- Best case: The best case occurs when the partition process always picks the middle element as pivot. So,

$$T(n) = T(n/2) + T(n/2) + \theta(n)$$
$$= 2T(n/2) + \theta(n)$$

$$\Rightarrow \quad T(n-1) = 2T\left(\frac{n-1}{2}\right) + \theta(n-1)$$

$$\Rightarrow \quad T(n) = 4T\left(\frac{n-1}{2}\right) + 2\theta(n-1) + \theta(n)$$
$$\simeq 4T\left(\frac{n-1}{2}\right) + 3\theta(n)$$

Similarly, $T(n) = 2^{r} \cdot 2T\left(\frac{n-r}{2}\right) + (2^{r+1}-1)\,\theta(n)$

on solving for $T(1) = 1$,

$$T(n) = \theta(n \log n)$$

- Average case: The average-case running time of quicksort is much closer to the best case, then the worst case. So for the Average case,

$$T(n) = O(n \log n)$$