

EXPERIMENT – 5

Student Name: Gaganjot Singh

UID: 22BCS14843

Branch: BE-CSE

Section/Group: 22BCS_JT_802-B

Semester: 5th

Date of Performance: 2024

Subject Name: Design & Analysis of Algorithms

Subject Code: 22CSH-311

1. Aim: To understand and implement Quick Sort.

2. Objective: Sorting a given set of elements using the Quick sort method and determine the time required to sort the elements. Repeating experiment for different values of n, the number of elements in the list to be sorted. The elements can be read from user or can be generated using the random number generator.

3. Algorithm

- Get the number of elements (n)
- Allocate memory for the array (int *arr = new int[n];)
- Ask the user for input or random numbers
- Input numbers or generate random numbers
- Print the original array (printArr(arr, n))
- Start the timer (clock_t start = clock();)
- Call Quick Sort (quickSort(arr, 0, n - 1))
- Stop the timer (clock_t end = clock();)
- Print the sorted array (printArr(arr, n))
- Calculate and print the time taken
- Deallocate memory (delete[] arr;)
- End

4. Implementation/Code:

```
#include <iostream>
#include <cstdlib>
#include <ctime>
using namespace std;

void swap(int *a, int *b)
{
    int t = *a;
    *a = *b;
    *b = t;
}
```

```
int partition(int arr[], int low, int high)
{int pivot = arr[high]; // pivot element
int i = (low - 1);
for (int j = low; j <= high - 1; j++)
{ if (arr[j] < pivot)
    { i++;
      swap(&arr[i], &arr[j]);
    }
}
swap(&arr[i + 1], &arr[high]);
return (i + 1);}
```

```
void quickSort(int arr[], int low, int high)
{if (low < high)
    {int pivot = partition(arr, low, high);
    quickSort(arr, low, pivot - 1);
    quickSort(arr, pivot + 1, high);}}
```

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

```
void RandomNumbers(int arr[], int size)
{
    srand(time(0));
    for (int i = 0; i < size; i++)
        {arr[i] = rand() % 100;
        }}
}
```

```
int main()
{ int n; // number of elements
  cout << "Enter the number of elements: ";
  cin >> n;

  int *arr = new int[n];
  cout << "Do you want to (1) input numbers or (2) generate random numbers? ";
  int choice;
  cin >> choice;

  if (choice == 1)
  { cout << "Enter " << n << " numbers: ";
    for (int i = 0; i < n; i++)
        {cin >> arr[i];
        }}

  else if (choice == 2)
```

```
{RandomNumbers(arr, n);}

cout << "Original array: ";
printArr(arr, n);

clock_t start = clock();

quickSort(arr, 0, n - 1);

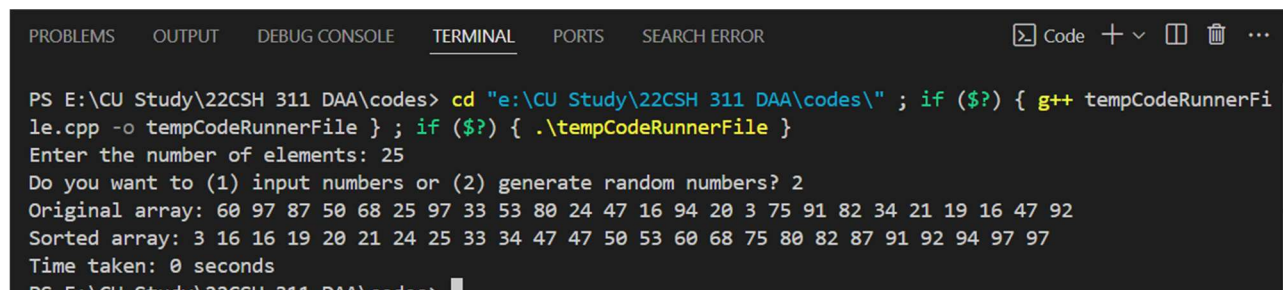
clock_t end = clock();

cout << "Sorted array: ";
printArr(arr, n);

double time_taken = double(end - start) / CLOCKS_PER_SEC;
cout << "Time taken: " << time_taken << " seconds" << endl;

return 0;}
```

5. Output



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS SEARCH ERROR
PS E:\CU Study\22CSH 311 DAA\codes> cd "e:\CU Study\22CSH 311 DAA\codes\" ; if ($?) { g++ tempCodeRunnerFi
le.cpp -o tempCodeRunnerFile } ; if ($?) { .\tempCodeRunnerFile }
Enter the number of elements: 25
Do you want to (1) input numbers or (2) generate random numbers? 2
Original array: 60 97 87 50 68 25 97 33 53 80 24 47 16 94 20 3 75 91 82 34 21 19 16 47 92
Sorted array: 3 16 16 19 20 21 24 25 33 34 47 47 50 53 60 68 75 80 82 87 91 92 94 97 97
Time taken: 0 seconds
PS E:\CU Study\22CSH 311 DAA\codes>
```

6. Time Complexity :

The overall time complexity of the code is dominated by the Quick Sort algorithm, which is $O(n \log n)$ on average and $O(n^2)$ in the worst case.

The time complexity of the RandomNumbers function is $O(n)$, where n is the number of elements in the array, but this is not a significant contributor to the overall time complexity.

7. Learning Outcomes:

- Understanding of the Quick Sort algorithm and its implementation
- Knowledge of the partitioning scheme and pivot selection in Quick Sort
- Familiarity with the concept of recursion in algorithm design
- Using Ctime's capabilities for the algorithmic analysis
- Dynamic memory allocation using new and delete operators