QUICK SORT

Aim- Write a C program to implement Quick Sort on array of integers

Problem Statement – Given a array of integers implement quick sort to sort the array of integers in assending order

INPUT - The number of elements in the array = 11

- Array Elements – 34,56,-19,26,58,38,30,-24,22,35,76

OUTPUT – Display each step of quick sort along with i,j,pivot and show the sorted array

ALGORITHM –

i] Algorithm QuickSort (p,q)

// Sorts the array elements a[p],a[p+1] on global array a[0-n-1] are arranged in such a way

//that they are in ascending order

{

If (p<q) then

{ j := partition (a,p,q+1);

quicksort(p,j-1) ; quicksort(j+1,q)

; } }

ii] Algorithm Partition(a,m,q)

//Within a[m],a[m+1]…… a[p-1] the elements are rearranged in such a manner that if

// initially t= a[m] ,then after completion a[q] = t for some q between m and p-1 ;

// a[k] <=t for some m<=k<=q , and a[k] >= t for some q<k<p , q is returned

{ v := a[m] ; i

:= m ; j : = p ;

repeat { repeat { i

:= i+1 ; } until

(a[i] >=v) ; repeat { j

```
:= j+1 ;  until (a[j]
<=v) ;
if (i < j    ) {
temp := a[i] ; a[i]
= a[j] ;
a[j] = temp ; }
until  (i>=j)    ;
a[m] = a[j] ;
a[j] : = v ; return
j ; }
```

Space and Time Complexity :

I] Algorithm QuickSort

Time Complexity:

i) **Best Case:**

- **O(n log n)**
- The pivot divides the array into two equal halves at every step, minimizing recursion depth.

ii) **Worst Case:**

- **$O(n^2)$**
- The pivot is always the smallest or largest element, leading to one side of the partition having all the elements, and recursion proceeds linearly.

iii) **Average Case:**

- **O(n log n)**
- On average, the pivot divides the array into reasonably balanced partitions, resulting in logarithmic recursion depth.

Space Complexity:

i) **Best Case:**

- **O(log n)**
- For recursive calls, the stack depth is proportional to the logarithm of the size of the array, as the array is divided into balanced halves.

ii) **Worst Case:**

- **O(n)**
- In the worst case, the stack depth is equal to the size of the array due to unbalanced partitions. iii) **Average Case:**
- **O(log n)**
- On average, the stack depth remains logarithmic.

II] Algorithm Partition

Time Complexity:

i) **Best Case:**

- **O(n)**
- All elements are scanned and rearranged once relative to the pivot.

ii) **Worst Case:**

- **O(n)**
- Every element is still scanned and rearranged in a single pass regardless of their order.

iii) **Average Case:**

- **O(n)**
- The partitioning process always involves a single scan of all elements in the subarray.

Space Complexity:

i) **Best Case:**

- **O(1)**
- Partitioning requires constant auxiliary space for temporary variables.

ii) **Worst Case:**

- **O(1)**
- No additional space is required apart from the input array and temporary variables.

iii) **Average Case:**

- **O(1)**
- Space usage remains constant.

RECURSION EQUATION –

I] Quick Sort

$$T(n) = T(k) + T(n - k - 1) + O(n)$$

II] Partition

The Partition algorithm does not have a recurrence relation because it is not recursive. It is a singlepass operation that rearranges elements relative to a pivot. Its complexity is handled entirely within its single invocation, and no further subproblems are generated. Therefore, no recurrence equation exists for Partition.

PROGRAM –

```
#include <stdio.h>

#define MAX 15

#include <time.h>

int a[MAX];

int n;


void displayArray() {

   int i;

   for (i = 0; i < n; i++) {

      printf("|%d ", a[i]);

   }

   printf("|\n");

}


void partition(int low, int high, int *pp) {



   int i, j, k, pivot, temp;

   pivot = a[low];

   i = low;

   j = high + 1;

   printf(" i = %d , j = %d ,pivot = %d .  \n ", i, j, pivot);

   displayArray();

   while (1) {

      do {

         i++;

      } while (i <= high && a[i] <= pivot);

      do {

         j--;

      } while (a[j] > pivot);

      if (i >= j) {

         break;

      }

      printf("\n");

      temp = a[i];

      a[i] = a[j];

      a[j] = temp;

      printf(" i = %d , j = %d ,pivot = %d . ( i , j
interchanged )\n  ", i, j, pivot);

      printf("\n");

      displayArray();

   }

   temp = a[low];

   a[low] = a[j];

   a[j] = temp;

   *pp = j;

   printf("\n i = %d , j = %d ,pivot = %d . (pivot exchanged
) \n ", i, j, pivot);

   displayArray();

}


void quicksort(int low, int high) {



   int p;

   if (low < high) {

      partition(low, high, &p);

      quicksort(low, p - 1);

      quicksort(p + 1, high);

   }



}


int main() {



       printf
("*****************************************
*********");

   printf ("\n Roll number: 23B-CO-010\n");

   printf (" PR Number - 202311390\n");
```

```
   printf("**********************************
******************\n\n\n");


   int choice, i;
    clock_t start, end;
   double cpu_time_used;
   do {
      printf("Menu:\n");
      printf("1. Enter array elements\n");
      printf("2. Sort array using Quicksort\n");
      printf("3. Display sorted array\n");
      printf("4. Exit\n");
      printf("Enter your choice: ");
      scanf("%d", &choice);


      switch (choice) {
         case 1:
            printf("Enter the number of elements in the
array: ");
            scanf("%d", &n);
            printf("Enter the elements of the array: ");
            for (i = 0; i < n; i++) {
               scanf("%d", &a[i]);
            }
            break;
         case 2:


   start = clock();
            quicksort(0, n - 1);
            printf("Array sorted using Quicksort.\n");
             end = clock();
           cpu_time_used = ((double)(end - start)) /
CLOCKS_PER_SEC;
           printf("Time taken by quicksort: %f seconds\n",
cpu_time_used);
            break;
         case 3:
            printf("Sorted array: ");
            displayArray();
            break;
         case 4:
            printf("Exiting...\n");
            break;
         default:
            printf("Invalid choice. Please try again.\n");
      }
   } while (choice != 4);
   return 0;
}
```

INPUT –

```
**************************************************
 Roll number: 23B-CO-010
 PR Number - 202311390
**************************************************


Menu:
1. Enter array elements
2. Sort array using Quicksort
3. Display sorted array
4. Exit
Enter your choice: 1
Enter the number of elements in the array: 11
Enter the elements of the array: 34
56
-19
26
58
38
30
-24
22
35
76
Menu:
1. Enter array elements
2. Sort array using Quicksort
3. Display sorted array
4. Exit
Enter your choice: 2
```

OUTPUT –

```
i = 0 , j = 11 ,pivot = 34 .
|34 |56 |-19 |26 |58 |38 |30 |-24 |22 |35 |76 |

i = 1 , j = 8 ,pivot = 34 . ( i , j interchanged )

34 |22 |-19 |26 |58 |38 |30 |-24 |56 |35 |76 |

i = 4 , j = 7 ,pivot = 34 . ( i , j interchanged )

34 |22 |-19 |26 |-24 |38 |30 |58 |56 |35 |76 |

i = 5 , j = 6 ,pivot = 34 . ( i , j interchanged )

34 |22 |-19 |26 |-24 |30 |38 |58 |56 |35 |76 |

i = 6 , j = 5 ,pivot = 34 . (pivot exchanged )
|30 |22 |-19 |26 |-24 |34 |38 |58 |56 |35 |76 |
i = 0 , j = 5 ,pivot = 30 .
|30 |22 |-19 |26 |-24 |34 |38 |58 |56 |35 |76 |

i = 5 , j = 4 ,pivot = 30 . (pivot exchanged )
|-24 |22 |-19 |26 |30 |34 |38 |58 |56 |35 |76 |
i = 0 , j = 4 ,pivot = -24 .
|-24 |22 |-19 |26 |30 |34 |38 |58 |56 |35 |76 |

i = 1 , j = 0 ,pivot = -24 . (pivot exchanged )
|-24 |22 |-19 |26 |30 |34 |38 |58 |56 |35 |76 |
i = 1 , j = 4 ,pivot = 22 .
|-24 |22 |-19 |26 |30 |34 |38 |58 |56 |35 |76 |

i = 3 , j = 2 ,pivot = 22 . (pivot exchanged )
|-24 |-19 |22 |26 |30 |34 |38 |58 |56 |35 |76 |
i = 6 , j = 11 ,pivot = 38 .
|-24 |-19 |22 |26 |30 |34 |38 |58 |56 |35 |76 |

i = 7 , j = 9 ,pivot = 38 . ( i , j interchanged )

-24 |-19 |22 |26 |30 |34 |38 |35 |56 |58 |76 |

i = 8 , j = 7 ,pivot = 38 . (pivot exchanged )
|-24 |-19 |22 |26 |30 |34 |35 |38 |56 |58 |76 |
i = 8 , j = 11 ,pivot = 56 .
|-24 |-19 |22 |26 |30 |34 |35 |38 |56 |58 |76 |

i = 9 , j = 8 ,pivot = 56 . (pivot exchanged )
|-24 |-19 |22 |26 |30 |34 |35 |38 |56 |58 |76 |
i = 9 , j = 11 ,pivot = 58 .
|-24 |-19 |22 |26 |30 |34 |35 |38 |56 |58 |76 |

i = 10 , j = 9 ,pivot = 58 . (pivot exchanged )

i = 9 , j = 8 ,pivot = 56 . (pivot exchanged )
|-24 |-19 |22 |26 |30 |34 |35 |38 |56 |58 |76 |
i = 9 , j = 11 ,pivot = 58 .
|-24 |-19 |22 |26 |30 |34 |35 |38 |56 |58 |76 |

i = 10 , j = 9 ,pivot = 58 . (pivot exchanged )
|-24 |-19 |22 |26 |30 |34 |35 |38 |56 |58 |76 |



i = 9 , j = 8 ,pivot = 56 . (pivot exchanged )
|-24 |-19 |22 |26 |30 |34 |35 |38 |56 |58 |76 |
i = 9 , j = 11 ,pivot = 58 .
|-24 |-19 |22 |26 |30 |34 |35 |38 |56 |58 |76 |

i = 10 , j = 9 ,pivot = 58 . (pivot exchanged )
 |-24 |-19 |22 |26 |30 |34 |35 |38 |56 |58 |76 |
Array sorted using Quicksort.
```

TIME TAKEN –

```
Time taken by quicksort: 0.288000 seconds
```

CONCLUSION - The array of integers was succesfully sorted using quick sort algorithm without errors .