**Assignment – 6**

**Q1. Implementation of Randomized Quick Sort and display the working of it.**

#include <stdio.h>  
#include <stdlib.h>  
#include <time.h>  
  
void randomized\_quick\_sort(int\*, int, int);  
void quick\_sort(int\*, int, int);  
int randomized\_partition(int\*, int, int);  
int partition(int\*, int, int);  
int random\_index(int, int);  
int\* create\_array(int);  
void display\_array(int\*, int, int);  
  
int quick\_sort\_call = 1, partition\_call = 1;  
  
int main() {  
 srand(time(NULL));  
 int size;  
 printf("Enter the size of array: ");  
 scanf("%d", &size);  
 int\* array = create\_array(size);  
 randomized\_quick\_sort(array, 0, size - 1);  
 printf("Final sorted array: ");  
 display\_array(array, 0, size - 1);  
 free(array);  
 return 0;  
}  
  
void randomized\_quick\_sort(int\* array, int low, int high) {  
 printf("Randomized Quick Sort call %d with low=%d and high=%d\n", quick\_sort\_call++, low, high);  
 display\_array(array, low, high);  
  
 if (low < high) {  
 int mid = randomized\_partition(array, low, high);  
 randomized\_quick\_sort(array, low, mid - 1);  
 randomized\_quick\_sort(array, mid + 1, high);  
 }  
}

void quick\_sort(int\* array, int low, int high) {  
 printf("Quick Sort call %d with low=%d and high=%d\n", quick\_sort\_call++, low, high);  
 display\_array(array, low, high);  
  
 if (low < high) {  
 int mid = partition(array, low, high);  
 quick\_sort(array, low, mid - 1);  
 quick\_sort(array, mid + 1, high);  
 }  
}  
  
int randomized\_partition(int\* array, int low, int high) {  
 int i = random\_index(low, high), temp;  
 printf("Random index chosen is %d with element %d\n", i, array[i]);  
 temp = array[i];  
 array[i] = array[high];  
 array[high] = temp;  
 return partition(array, low, high);  
}  
  
int partition(int\* array, int low, int high) {  
 printf("Partition call %d with low=%d, high=%d\n", partition\_call++, low, high);  
 printf("Before partition ");  
 display\_array(array, low, high);  
 int pivot = array[high], i = low, j, temp;  
  
 for (j = low; j <= high; j++) {  
 if (array[j] <= pivot) {  
 temp = array[j];  
 array[j] = array[i];  
 array[i++] = temp;  
 }  
 }  
 printf("After partition ");  
 display\_array(array, low, high);  
 return i - 1;  
}  
  
int random\_index(int low, int high) { return low + rand() % (high - low + 1); }  
  
int\* create\_array(int n) {  
 int \*array = (int\*)malloc(sizeof(int) \* n), i;  
  
 if (!array) {  
 printf("Memory was not allocated during creation of array");  
 exit(0);  
 }  
 printf("Enter the array elements: ");  
 for (i = 0; i < n; i++) {  
 scanf("%d", &array[i]);  
 }  
 return array;  
}  
  
void display\_array(int\* array, int low, int high) {  
 int i;  
 printf("The array elements are: ");  
  
 for (i = low; i <= high; i++) {  
 printf("%d ", array[i]);  
 }  
 printf("\n\n");  
}

**Output**:

Enter the size of array: 10  
Enter the array elements: 4 1 3 2 16 9 10 14 8 7

Randomized Quick Sort call 1 with low=0 and high=9  
The array elements are: 4 1 3 2 16 9 10 14 8 7  
  
Random index chosen is 6 with element 10  
Partition call 1 with low=0, high=9  
Before partition The array elements are: 4 1 3 2 16 9 7 14 8 10  
After partition The array elements are: 4 1 3 2 9 7 8 10 16 14  
  
Randomized Quick Sort call 2 with low=0 and high=6  
The array elements are: 4 1 3 2 9 7 8  
  
Random index chosen is 6 with element 8  
Partition call 2 with low=0, high=6  
Before partition The array elements are: 4 1 3 2 9 7 8  
After partition The array elements are: 4 1 3 2 7 8 9  
  
Randomized Quick Sort call 3 with low=0 and high=4  
The array elements are: 4 1 3 2 7  
  
Random index chosen is 3 with element 2  
Partition call 3 with low=0, high=4  
Before partition The array elements are: 4 1 3 7 2  
After partition The array elements are: 1 2 3 7 4  
  
Randomized Quick Sort call 4 with low=0 and high=0  
The array elements are: 1  
  
Randomized Quick Sort call 5 with low=2 and high=4  
The array elements are: 3 7 4  
  
Random index chosen is 3 with element 7  
Partition call 4 with low=2, high=4  
Before partition The array elements are: 3 4 7  
After partition The array elements are: 3 4 7  
  
Randomized Quick Sort call 6 with low=2 and high=3  
The array elements are: 3 4  
  
Random index chosen is 2 with element 3  
Partition call 5 with low=2, high=3  
Before partition The array elements are: 4 3  
After partition The array elements are: 3 4  
  
Randomized Quick Sort call 7 with low=2 and high=1  
The array elements are:  
Randomized Quick Sort call 8 with low=3 and high=3  
The array elements are: 4  
Randomized Quick Sort call 9 with low=5 and high=4  
The array elements are:  
Randomized Quick Sort call 10 with low=6 and high=6  
The array elements are: 9  
Randomized Quick Sort call 11 with low=8 and high=9  
The array elements are: 16 14  
  
Random index chosen is 8 with element 16  
Partition call 6 with low=8, high=9  
Before partition The array elements are: 14 16  
After partition The array elements are: 14 16  
  
Randomized Quick Sort call 12 with low=8 and high=8  
The array elements are: 14  
  
Randomized Quick Sort call 13 with low=10 and high=9  
The array elements are:  
  
Final sorted array: The array elements are: 1 2 3 4 7 8 9 10 14 16