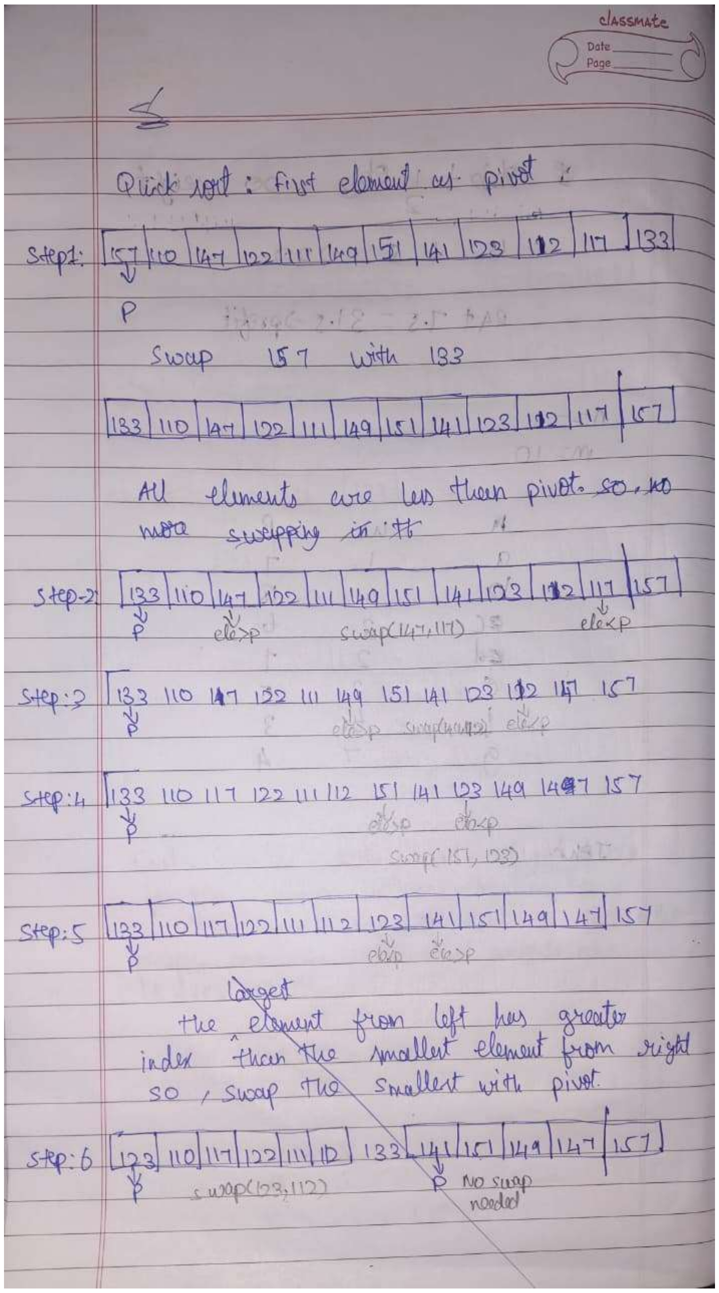
Design Analysis and Algorithm – Lab Work

Week 6

Question 1: Write a Program to perform Quick Sort using

a. first element

METHOD:



Quick sort : first element as pivot :

Step1: | 157 | 110 | 147 | 122 | 111 | 149 | 151 | 141 | 123 | 112 | 117 | 133 |

P

Swap 157 with 133

| 133 | 110 | 147 | 122 | 111 | 149 | 151 | 141 | 123 | 112 | 117 | 157 |

All elements are less then pivot. so, no more swapping ~~in~~ it

Step-2: | 133 | 110 | 147 | 122 | 111 | 149 | 151 | 141 | 123 | 112 | 117 | 157 |

P   ele>p   swap(147, 117)   ele<P

Step:3 133 110 117 122 111 149 151 141 123 112 147 157

P   ele>p   swap(149,112)   ele<p

Step:4 133 110 117 122 111 112 151 141 123 149 147 157

P   ele>p   ele<p

swap(151, 123)

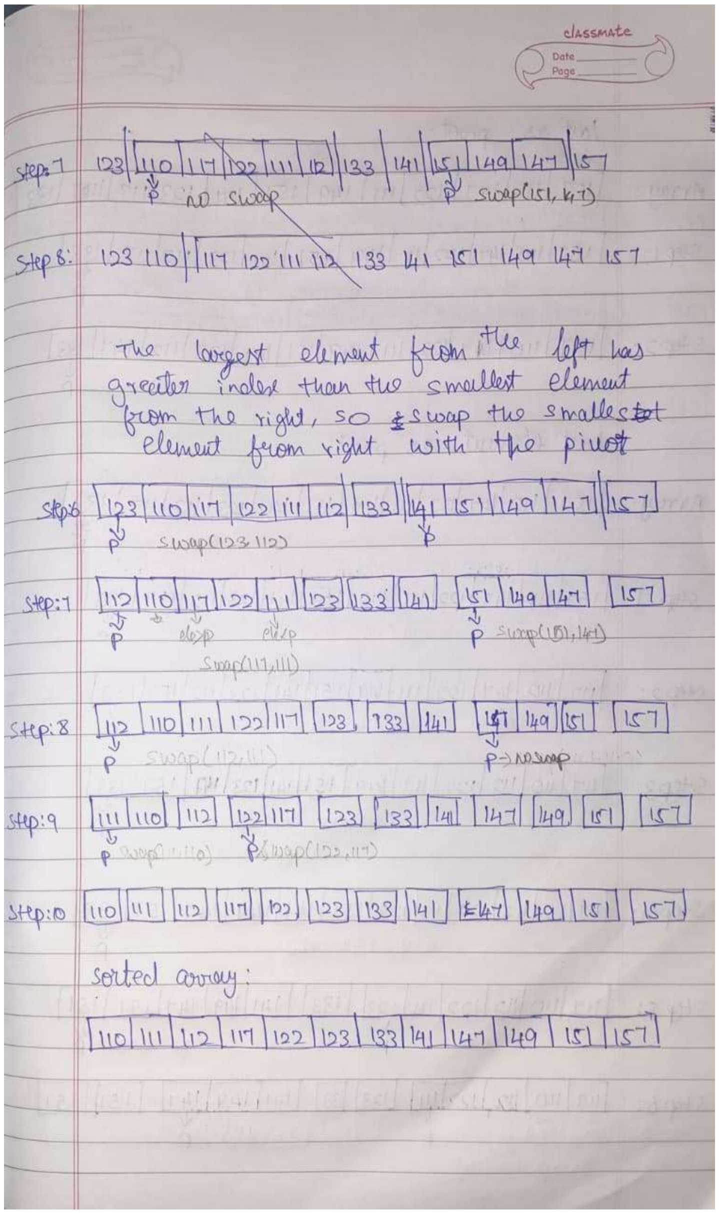
Step:5 | 133 | 110 | 117 | 122 | 111 | 112 | 123 | 141 | 151 | 149 | 147 | 157

P   ele<p   ele>p

the largest element from left has greater index than the smallest element from right so, swap the smallest with pivot.

Step:6 | 123 | 110 | 117 | 122 | 111 | 112 | 133 | 141 | 151 | 149 | 147 | 157 |

P   swap(123,112)   P No swap needed



classmate

Date

Page

~~step:7 123 | 110 | 117 | 122 | 111 | 112 | 133 | 141 | 151 | 149 | 147 | 157~~

~~p no swap p swap(151, 147)~~

~~Step 8: 123 110 | 117 122 111 112 133 141 151 149 147 157~~

The largest element from the left has greater index than the smallest element from the right, so swap the smallest element from right with the pivot

step:6 | 123 | 110 | 117 | 122 | 111 | 112 | 133 | 141 | 151 | 149 | 147 | 157 |

p swap(123, 112) p

step:7 | 112 | 110 | 117 | 122 | 111 | 123 | 133 | 141 | 151 | 149 | 147 | 157 |

p ele>p ele<p p swap(151, 141)

swap(117, 111)

step:8 | 112 | 110 | 111 | 122 | 117 | 123 | 133 | 141 | 147 | 149 | 151 | 157 |

p swap(112, 111) p -> no swap

step:9 | 111 | 110 | 112 | 122 | 117 | 123 | 133 | 141 | 147 | 149 | 151 | 157 |

p swap(111, 110) p swap(122, 117)

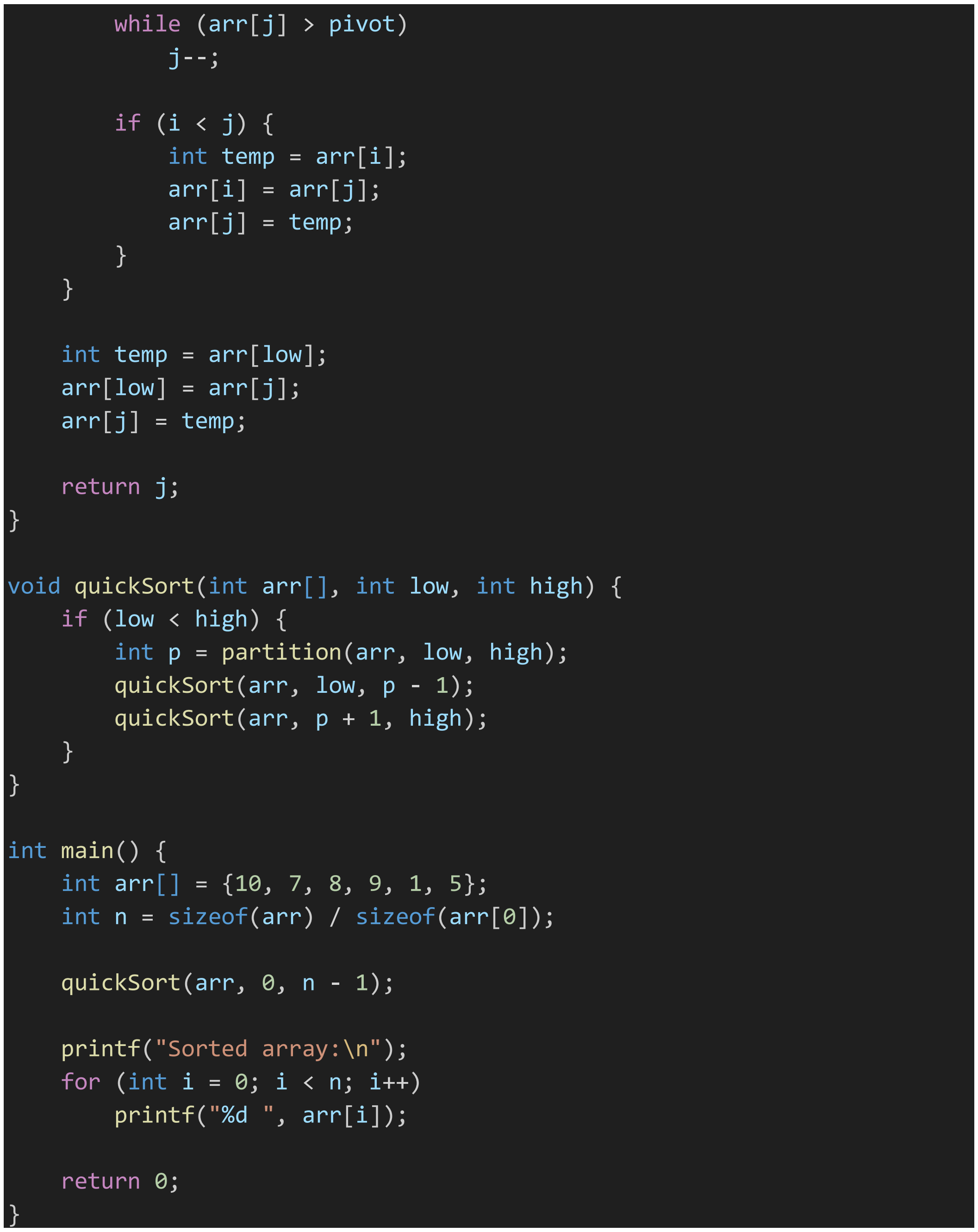
step:10 | 110 | 111 | 112 | 117 | 122 | 123 | 133 | 141 | 147 | 149 | 151 | 157 |

sorted array:

| 110 | 111 | 112 | 117 | 122 | 123 | 133 | 141 | 147 | 149 | 151 | 157 |

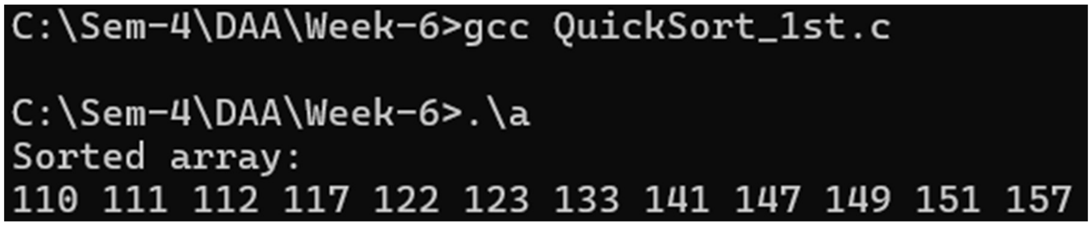
CODE:

```
#include <stdio.h>

int partition(int arr[], int low, int high) {
    int pivot = arr[low];
    int i = low + 1;
    int j = high;

    while (i <= j) {
        while (i <= high && arr[i] <= pivot)
            i++;
```



```
        while (arr[j] > pivot)
            j--;

        if (i < j) {
            int temp = arr[i];
            arr[i] = arr[j];
            arr[j] = temp;
        }
    }

    int temp = arr[low];
    arr[low] = arr[j];
    arr[j] = temp;

    return j;
}

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

int main() {
    int arr[] = {10, 7, 8, 9, 1, 5};
    int n = sizeof(arr) / sizeof(arr[0]);

    quickSort(arr, 0, n - 1);

    printf("Sorted array:\n");
    for (int i = 0; i < n; i++)
        printf("%d ", arr[i]);

    return 0;
}
```

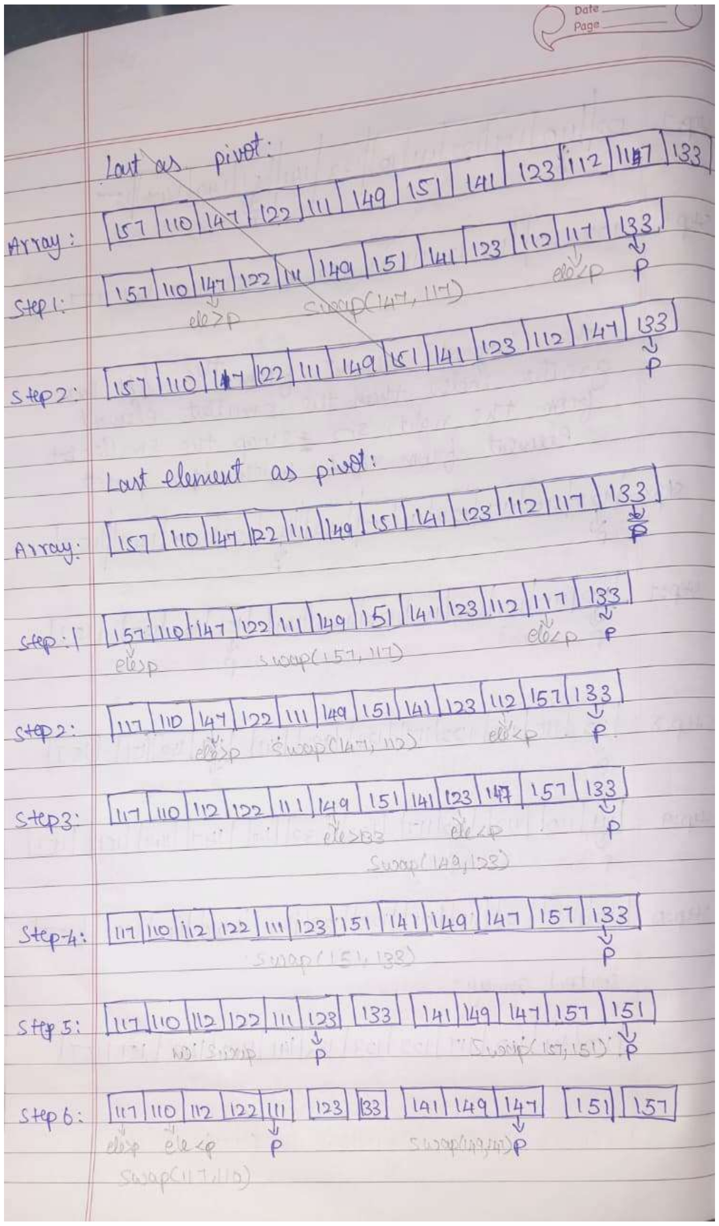
OUTPUT:



```
C:\Sem-4\DAA\Week-6>gcc QuickSort_1st.c

C:\Sem-4\DAA\Week-6>.\a
Sorted array:
110 111 112 117 122 123 133 141 147 149 151 157
```

b. middle element

METHOD:



Date

Page

~~Last as pivot:~~

~~Array: | 157 | 110 | 147 | 122 | 111 | 149 | 151 | 141 | 123 | 112 | 117 | 133 |~~

~~Step 1: | 157 | 110 | 147 | 122 | 111 | 149 | 151 | 141 | 123 | 112 | 117 | 133 |~~
~~ele>p swap(147, 117) ele<p p~~

~~Step 2: | 157 | 110 | 147 | 122 | 111 | 149 | 151 | 141 | 123 | 112 | 147 | 133 |~~
~~p~~

Last element as pivot:

Array: | 157 | 110 | 147 | 122 | 111 | 149 | 151 | 141 | 123 | 112 | 117 | 133 |
p

Step: 1 | 157 | 110 | 147 | 122 | 111 | 149 | 151 | 141 | 123 | 112 | 117 | 133 |
ele>p swap(157, 117) ele<p p

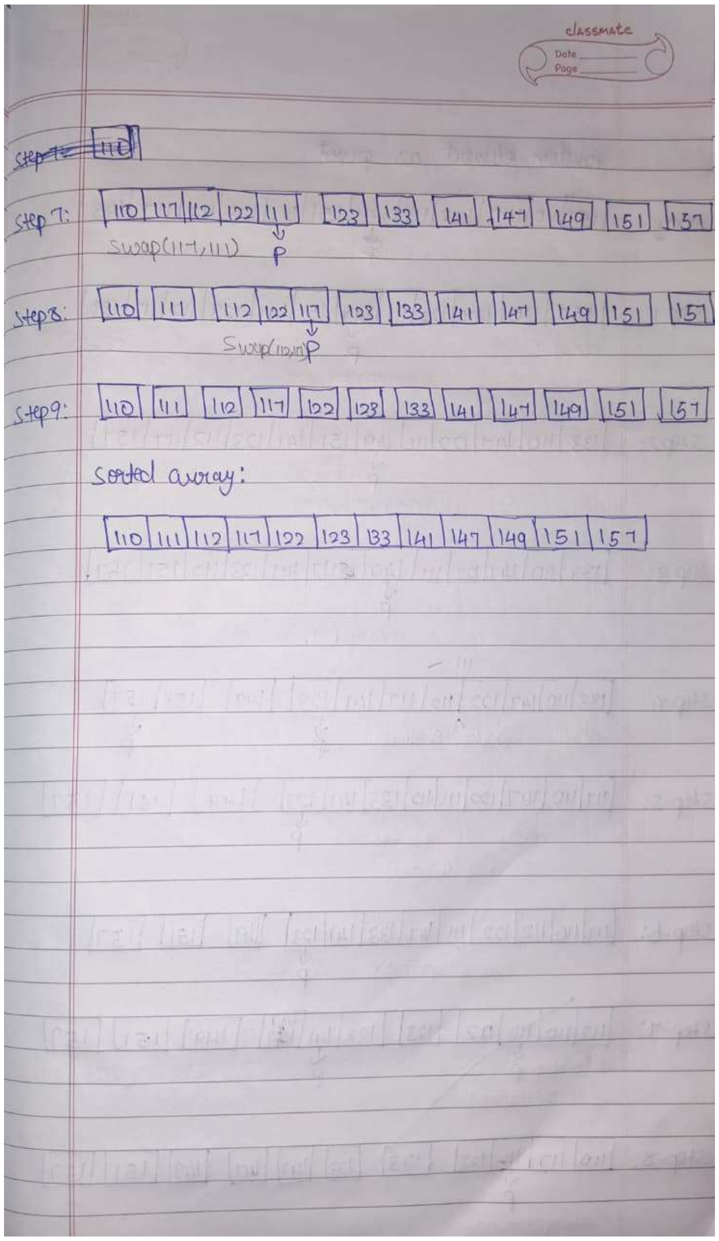
Step 2: | 117 | 110 | 147 | 122 | 111 | 149 | 151 | 141 | 123 | 112 | 157 | 133 |
ele>p swap(147, 112) ele<p p

Step 3: | 117 | 110 | 112 | 122 | 111 | 149 | 151 | 141 | 123 | 147 | 157 | 133 |
ele>133 ele<p p
swap(149, 123)

Step-4: | 117 | 110 | 112 | 122 | 111 | 123 | 151 | 141 | 149 | 147 | 157 | 133 |
swap(151, 133) p

Step 5: | 117 | 110 | 112 | 122 | 111 | 123 | | 133 | | 141 | 149 | 147 | 157 | 151 |
no swap p swap(157, 151) p

Step 6: | 117 | 110 | 112 | 122 | 111 | | 123 | | 133 | | 141 | 149 | 147 | | 151 | | 157 |
ele>p ele<p p swap(149, 147) p
swap(117, 110)


~~step 7:~~ ~~110~~

step 7: | 110 | 117 | 112 | 122 | 111 | 123 | 133 | 141 | 147 | 149 | 151 | 157 |

swap(117,111) P

step 8: | 110 | 111 | 112 | 122 | 117 | 123 | 133 | 141 | 147 | 149 | 151 | 157 |

Swap(122,117) P

step 9: | 110 | 111 | 112 | 117 | 122 | 123 | 133 | 141 | 147 | 149 | 151 | 157 |

sorted array:

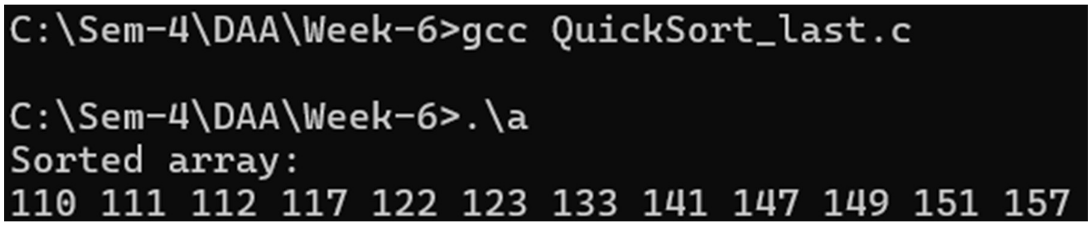
| 110 | 111 | 112 | 117 | 122 | 123 | 133 | 141 | 147 | 149 | 151 | 157 |

CODE:

```
#include <stdio.h>

int partition(int arr[], int low, int high) {
    int pivot = arr[high];
    int i = low - 1;

    for (int j = low; j < high; j++) {
        if (arr[j] <= pivot) {
            i++;
            int temp = arr[i];
            arr[i] = arr[j];
```

```c
            arr[j] = temp;
        }
    }

    int temp = arr[i + 1];
    arr[i + 1] = arr[high];
    arr[high] = temp;

    return i + 1;
}

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

int main() {
    int arr[] = {10, 7, 8, 9, 1, 5};
    int n = sizeof(arr) / sizeof(arr[0]);

    quickSort(arr, 0, n - 1);

    printf("Sorted array:\n");
    for (int i = 0; i < n; i++)
        printf("%d ", arr[i]);

    return 0;
}
```

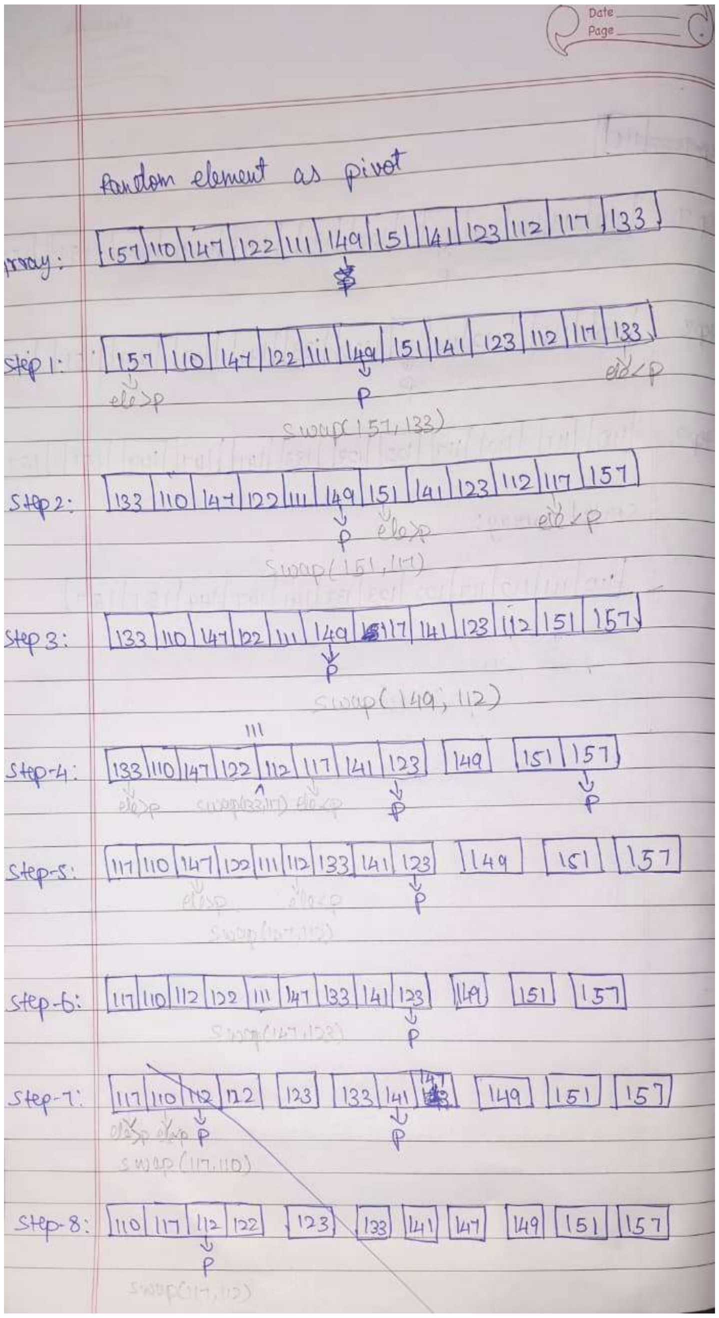
OUTPUT:

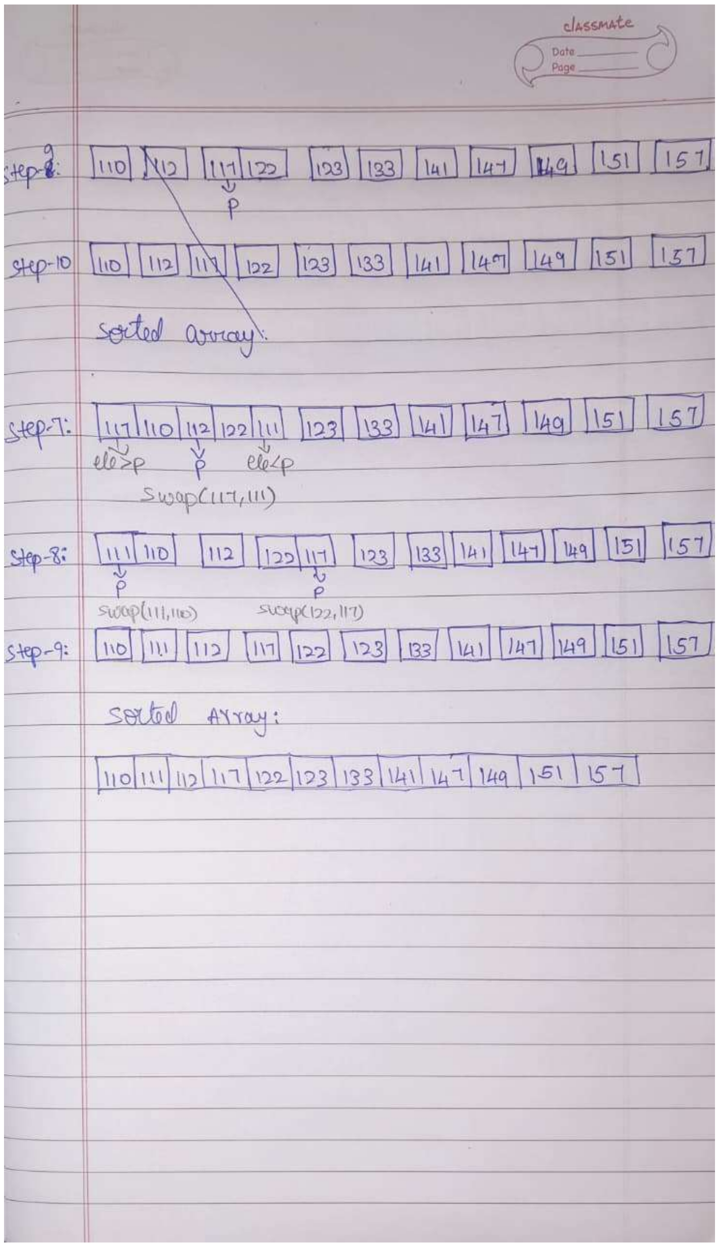


```
C:\Sem-4\DAA\Week-6>gcc QuickSort_last.c

C:\Sem-4\DAA\Week-6>.\a
Sorted array:
110 111 112 117 122 123 133 141 147 149 151 157
```

c. random element.

METHOD:


Random element as pivot
array: 157 110 147 122 111 149 151 141 123 112 117 133
step 1: 157 110 147 122 111 149 151 141 123 112 117 133
ele>p
p
ele<p
swap(157, 133)
step 2: 133 110 147 122 111 149 151 141 123 112 117 157
p
ele>p
ele<p
swap(151, 117)
step 3: 133 110 147 122 111 149 117 141 123 112 151 157
p
swap(149, 112)
111
step-4: 133 110 147 122 112 117 141 123 149 151 157
ele>p
swap(133,117)
ele<p
p
p
step-5: 117 110 147 122 111 112 133 141 123 149 151 157
ele>p
ele<p
p
swap(147,112)
step-6: 117 110 112 122 111 147 133 141 123 149 151 157
swap(147,123)
p
step-7: 117 110 112 122 123 133 141 147 149 151 157
ele>p
ele<p
p
p
swap(117,110)
step-8: 110 117 112 122 123 133 141 147 149 151 157
p
swap(117,110)



classmate
Date
Page

step-~~8~~9: | 110 | ~~112~~ | 117 | 122 | 123 | 133 | 141 | 147 | 149 | 151 | 157 |
p

step-10 | 110 | 112 | 117 | 122 | 123 | 133 | 141 | 147 | 149 | 151 | 157 |

~~sorted array:~~

step-7: | 117 | 110 | 112 | 122 | 111 | 123 | 133 | 141 | 147 | 149 | 151 | 157 |
ele>p p ele<p
Swap(117,111)

Step-8: | 111 | 110 | 112 | 122 | 117 | 123 | 133 | 141 | 147 | 149 | 151 | 157 |
p p
swap(111,110) swap(122,117)

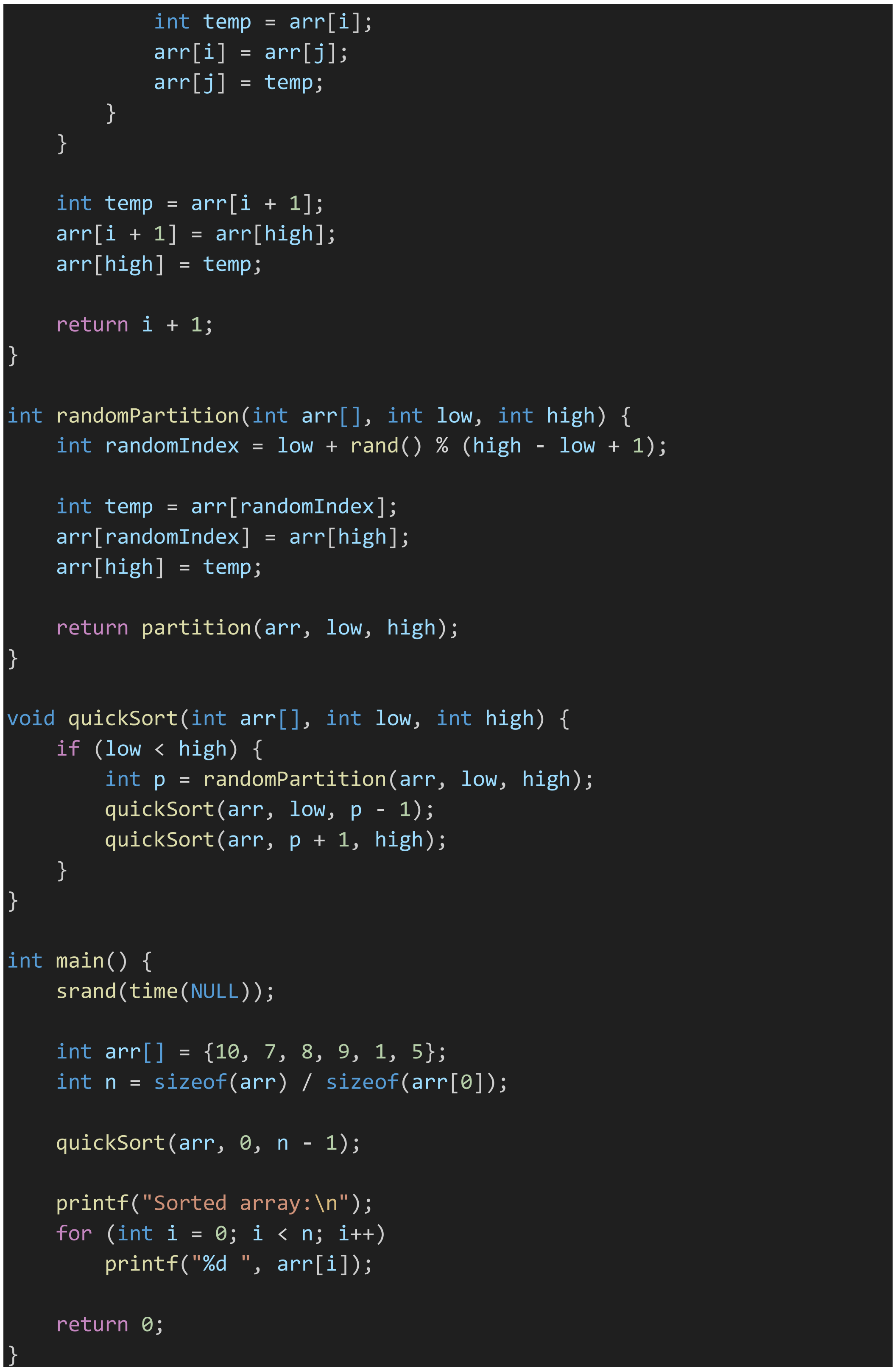
step-9: | 110 | 111 | 112 | 117 | 122 | 123 | 133 | 141 | 147 | 149 | 151 | 157 |

Sorted Array:

| 110 | 111 | 112 | 117 | 122 | 123 | 133 | 141 | 147 | 149 | 151 | 157 |

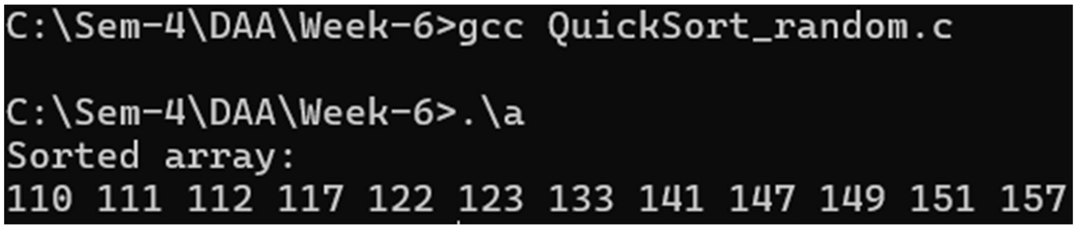
CODE:

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int partition(int arr[], int low, int high) {
    int pivot = arr[high];
    int i = low - 1;

    for (int j = low; j < high; j++) {
        if (arr[j] <= pivot) {
            i++;
```



```c
            int temp = arr[i];
            arr[i] = arr[j];
            arr[j] = temp;
        }
    }

    int temp = arr[i + 1];
    arr[i + 1] = arr[high];
    arr[high] = temp;

    return i + 1;
}

int randomPartition(int arr[], int low, int high) {
    int randomIndex = low + rand() % (high - low + 1);

    int temp = arr[randomIndex];
    arr[randomIndex] = arr[high];
    arr[high] = temp;

    return partition(arr, low, high);
}

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

int main() {
    srand(time(NULL));

    int arr[] = {10, 7, 8, 9, 1, 5};
    int n = sizeof(arr) / sizeof(arr[0]);

    quickSort(arr, 0, n - 1);

    printf("Sorted array:\n");
    for (int i = 0; i < n; i++)
        printf("%d ", arr[i]);

    return 0;
}
```

OUTPUT:



```
C:\Sem-4\DAA\Week-6>gcc QuickSort_random.c

C:\Sem-4\DAA\Week-6>.\a
Sorted array:
110 111 112 117 122 123 133 141 147 149 151 157
```

Time Complexity:

Quick Sort has a best-case and average-case time complexity of **O(n log n)**, which occurs when the pivot element divides the array into two nearly equal subarrays at each recursive step. However, in the worst case, when the pivot selection results in highly unbalanced partitions (such as when the smallest or largest element is consistently chosen as the pivot), the time complexity degrades to **O(n^2)**. Although different pivot selection strategies can reduce the likelihood of the worst case, they do not change the theoretical time complexity bounds of the algorithm.

Space Complexity:

Quick Sort is an in-place sorting algorithm and therefore requires **O(1)** auxiliary space for data storage. However, due to its recursive nature, additional space is required for the recursion stack. In the best and average cases, the depth of recursion is **O(log n)**, resulting in a space complexity of **O(log n)**, whereas in the worst case of highly unbalanced partitions, the recursion depth can grow to **O(n)**, leading to a space complexity of **O(n)**.