# WEEK-5

Name: B Lohith Krishnan
Roll Number: CH.SC.U4CSE24153

Quick Sort with First element as Pivot:

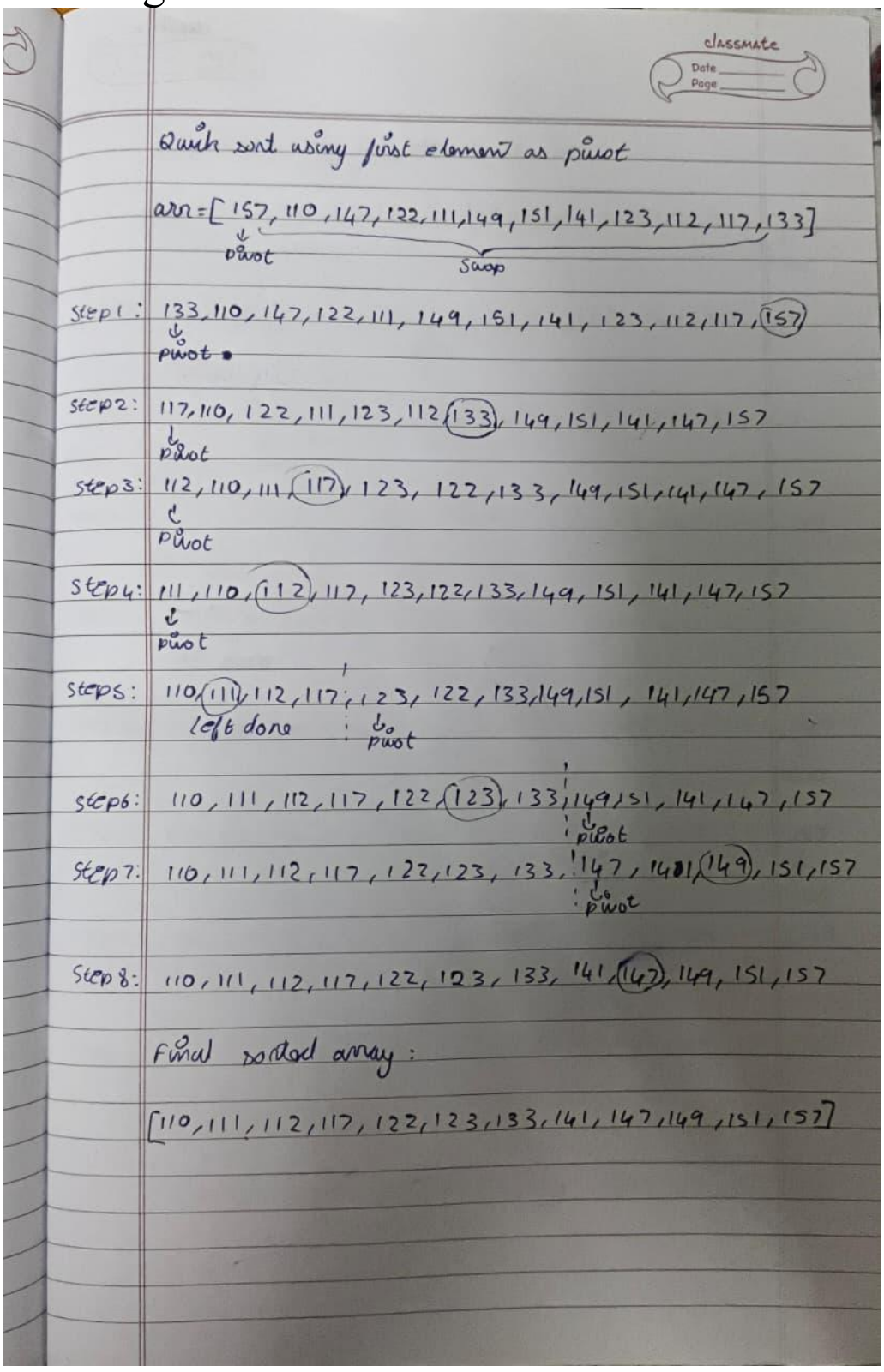
Code:

```c
#include <stdio.h>

int partitionFirst(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;
        }
    }

    arr[low] = arr[j];
    arr[j] = pivot;

    return j;
}

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

int main() {
    int arr[] = {157,110,147,122,111,149,151,141,123,112,117,133};
    int n = 12;

    quickSortFirst(arr, 0, n - 1);

    for (int i = 0; i < n; i++)
        printf("%d ", arr[i]);
    return 0;
}
```

Output:

```
C:\DAA\lab\week 5>gcc quickFirst.c

C:\DAA\lab\week 5>a
110 111 112 117 122 123 133 141 147 149 151 157
C:\DAA\lab\week 5>
```

Working:



Quick Sort with Last element as Pivot:

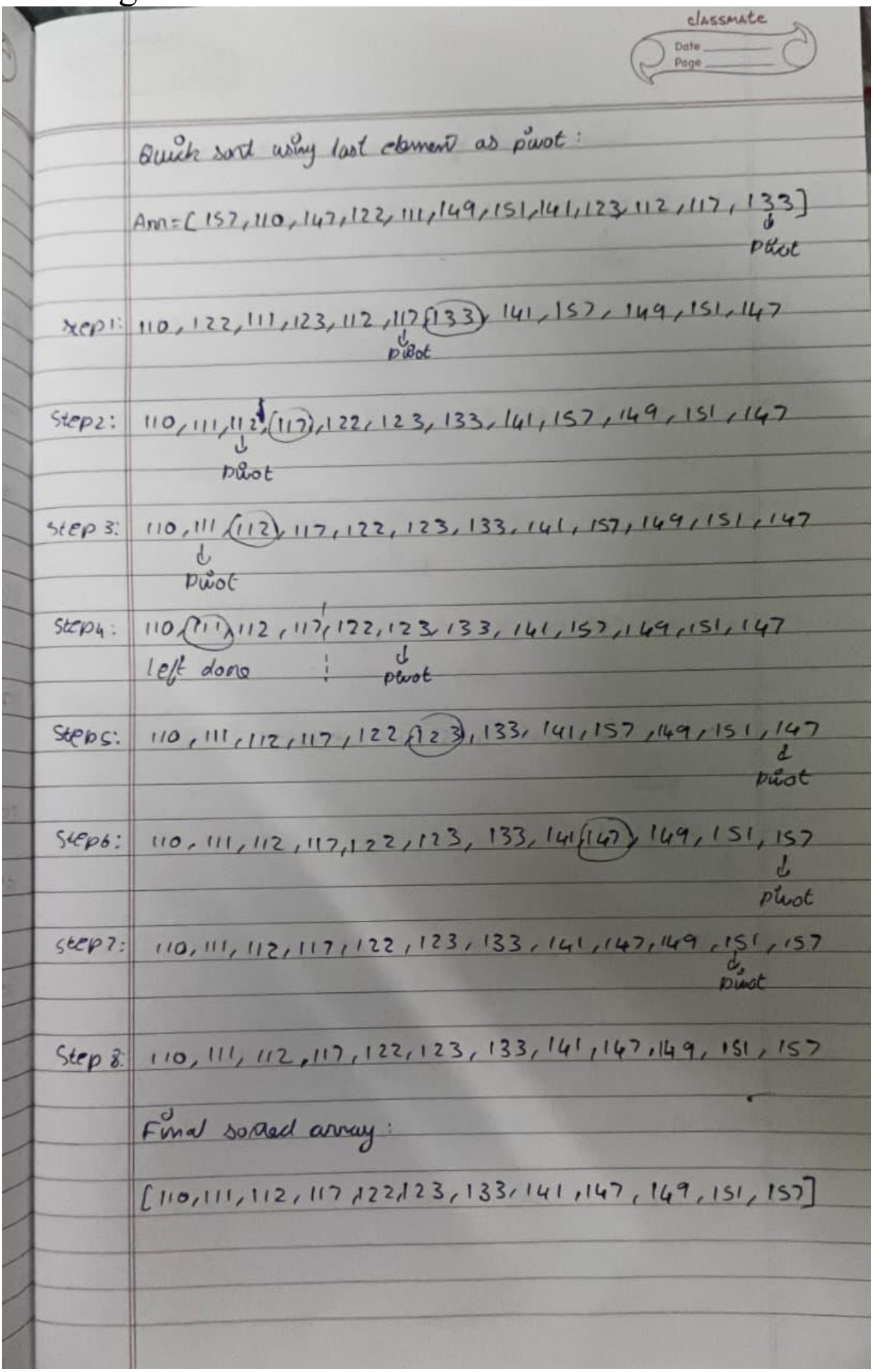
Code:

```c
#include <stdio.h>

int partitionLast(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];
            arr[j] = temp;
        }
    }

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

    return i + 1;
}

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

int main() {
    int arr[] = {157,110,147,122,111,149,151,141,123,112,117,133};
    int n = 12;

    quickSortLast(arr, 0, n - 1);

    for (int i = 0; i < n; i++)
        printf("%d ", arr[i]);
    return 0;
}
```

Output:

```
C:\DAA\lab\week 5>gcc quickLast.c

C:\DAA\lab\week 5>a
110 111 112 117 122 123 133 141 147 149 151 157
C:\DAA\lab\week 5>
```

Working:



Quick sort using last element as pivot:

Arr = [157, 110, 147, 122, 111, 149, 151, 141, 123, 112, 117, 133]
pivot

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

Step2: 110, 111, 112, 117, 122, 123, 133, 141, 157, 149, 151, 147
pivot

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

Step4: 110, 111, 112, 117, 122, 123, 133, 141, 157, 149, 151, 147
left done ; pivot

Step5: 110, 111, 112, 117, 122, 123, 133, 141, 157, 149, 151, 147
pivot

Step6: 110, 111, 112, 117, 122, 123, 133, 141, 147, 149, 151, 157
pivot

Step7: 110, 111, 112, 117, 122, 123, 133, 141, 147, 149, 151, 157
pivot

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

Final sorted array:

[110, 111, 112, 117, 122, 123, 133, 141, 147, 149, 151, 157]

# Quick Sort with Random element as Pivot:

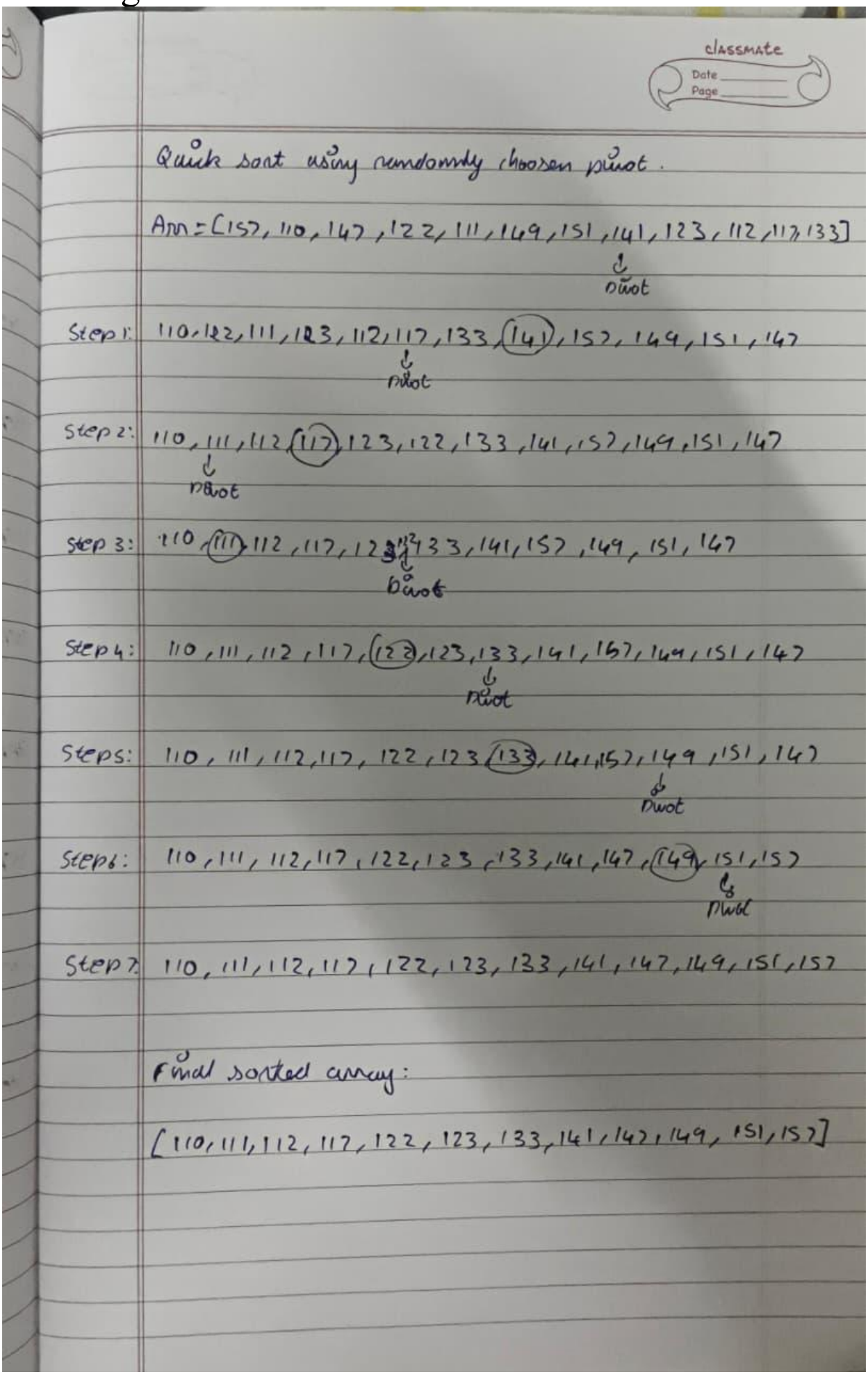
## Code:

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

int partitionLast(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];
            arr[j] = temp;
        }
    }

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

    return i + 1;
}

int partitionRandom(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 partitionLast(arr, low, high);
}

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

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

    int arr[] = {157,110,147,122,111,149,151,141,123,112,117,133};
    int n = 12;

    quickSortRandom(arr, 0, n - 1);

    for (int i = 0; i < n; i++)
        printf("%d ", arr[i]);
    return 0;
}
```

## Output:

```
C:\DAA\lab\week 5>gcc quickRand.c

C:\DAA\lab\week 5>a
110 111 112 117 122 123 133 141 147 149 151 157
C:\DAA\lab\week 5>
```

Working:


classmate
Date
Page
Quick sort using randomly choosen pivot.
Arr = [157, 110, 147, 122, 111, 149, 151, 141, 123, 112, 117, 133]
pivot
Step 1: 110, 122, 111, 123, 112, 117, 133, (141), 157, 149, 151, 147
pivot
Step 2: 110, 111, 112, (117), 123, 122, 133, 141, 157, 149, 151, 147
pivot
Step 3: 110, (111), 112, 117, 123, 122, 133, 141, 157, 149, 151, 147
pivot
Step 4: 110, 111, 112, 117, (123), 122, 133, 141, 157, 149, 151, 147
pivot
Step 5: 110, 111, 112, 117, 122, 123, (133), 141, 157, 149, 151, 147
pivot
Step 6: 110, 111, 112, 117, 122, 123, 133, 141, 147, (149), 151, 157
pivot
Step 7: 110, 111, 112, 117, 122, 123, 133, 141, 147, 149, 151, 157
Final sorted array:
[110, 111, 112, 117, 122, 123, 133, 141, 147, 149, 151, 157]

# Analysis:

Random element is the best choice as pivot in Quick Sort.

Using the first or last element as pivot can easily lead to the worst-case time complexity of $O(n^2)$ when the array is already sorted or nearly sorted, because the partitions become highly unbalanced.

On the other hand, choosing a random pivot makes it very unlikely to consistently produce bad partitions, so the algorithm tends to divide the array more evenly. As a result, Quick Sort with a random pivot achieves an expected time complexity of $O(n\log n)$ in most practical cases, making it the most efficient and reliable pivot selection method.