DESIGN AND ANALYSIS OF ALGORITHMS

LAB WORKBOOK WEEK – 6

NAME: Konda Bhavani

ROLL NUMBER: CH.SC.U4CSE24120

CLASS: CSE-B

Quick Sort (First element as pivot)

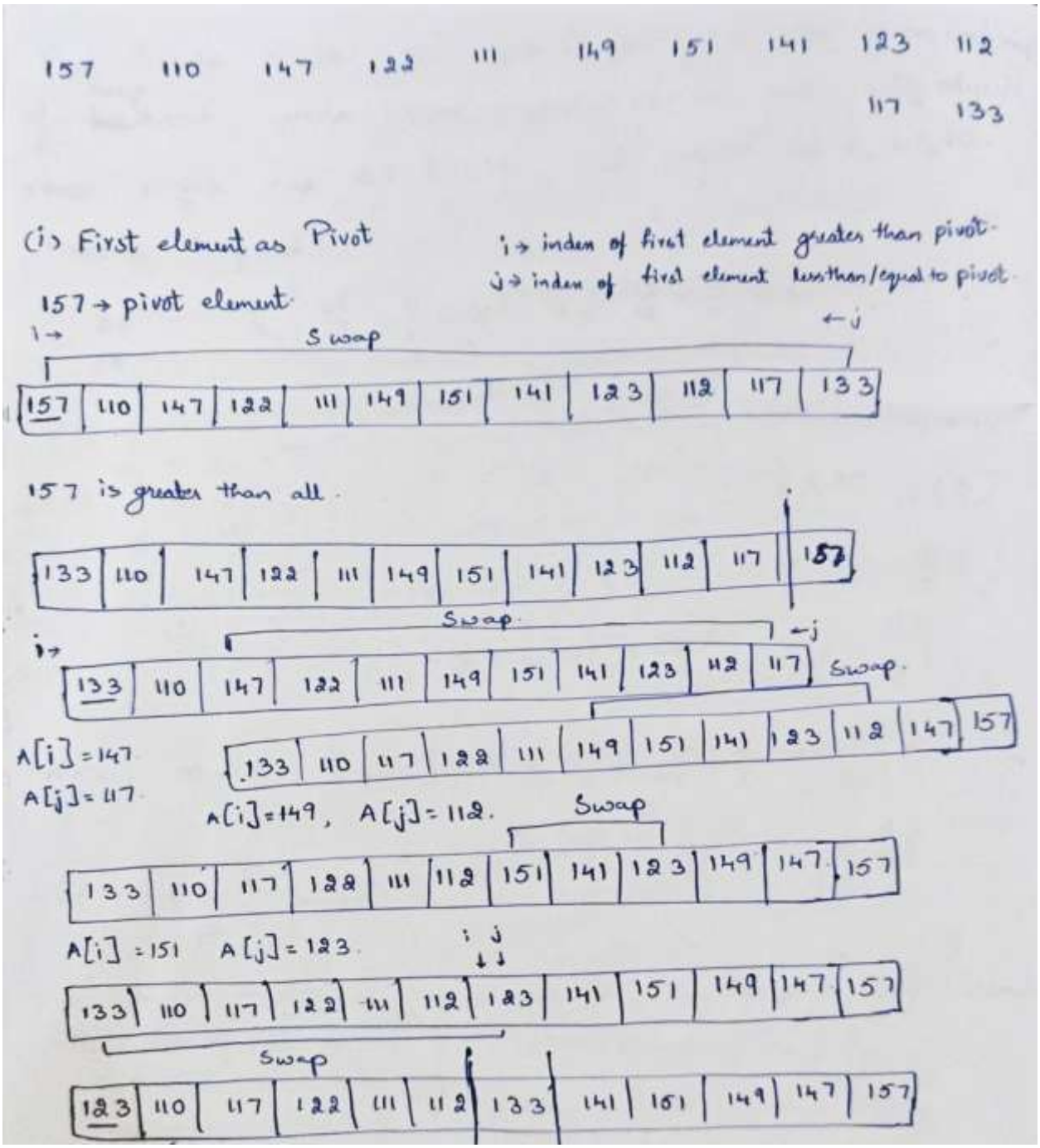
Code:

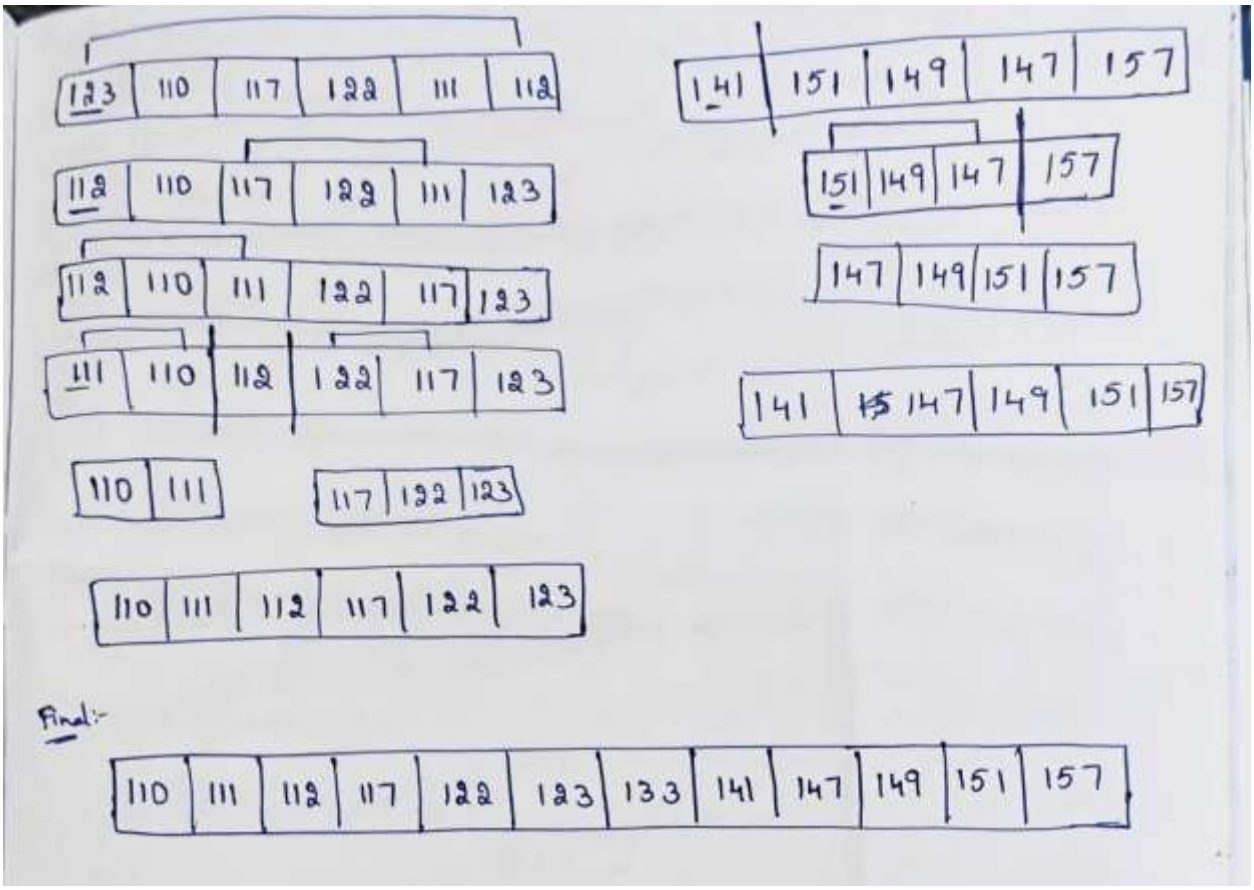
```c
//CH.SC.U4CSE24120
#include <stdio.h>
int Partition(int a[], int low, int high) {
    int pivot = a[low];
    int i = low + 1;
    int j = high;
    while (1) {
        while (i <= high && a[i] <= pivot)
            i++;
        while (a[j] > pivot)
            j--;
        if (i >= j)
            break;
        int temp = a[i];
        a[i] = a[j];
        a[j] = temp;
    }
    int temp = a[low];
    a[low] = a[j];
    a[j] = temp;
    return j;
}
void quickSort(int a[], int low, int high) {
    if (low < high) {
        int p = Partition(a, low, high);
        quickSort(a, low, p - 1);
        quickSort(a, p + 1, high);
    }
}
int main() {
    int a[] = {157, 110, 147, 122, 111, 149, 151, 141, 123, 112, 117, 133};
    int n = sizeof(a) / sizeof(a[0]);

    quickSort(a, 0, n - 1);

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

    return 0;
}
```

Output:

```
Sorted array:
110 111 112 117 122 123 133 141 147 149 151 157
```

Working:




123 110 117 122 111 112
112 110 117 122 111 123
112 110 111 122 117 123
111 110 112 122 117 123
110 111
117 122 123
110 111 112 117 122 123
141 151 149 147 157
151 149 147 157
147 149 151 157
141 147 149 151 157
Final:-
110 111 112 117 122 123 133 141 147 149 151 157

Quick Sort (Last element as pivot)

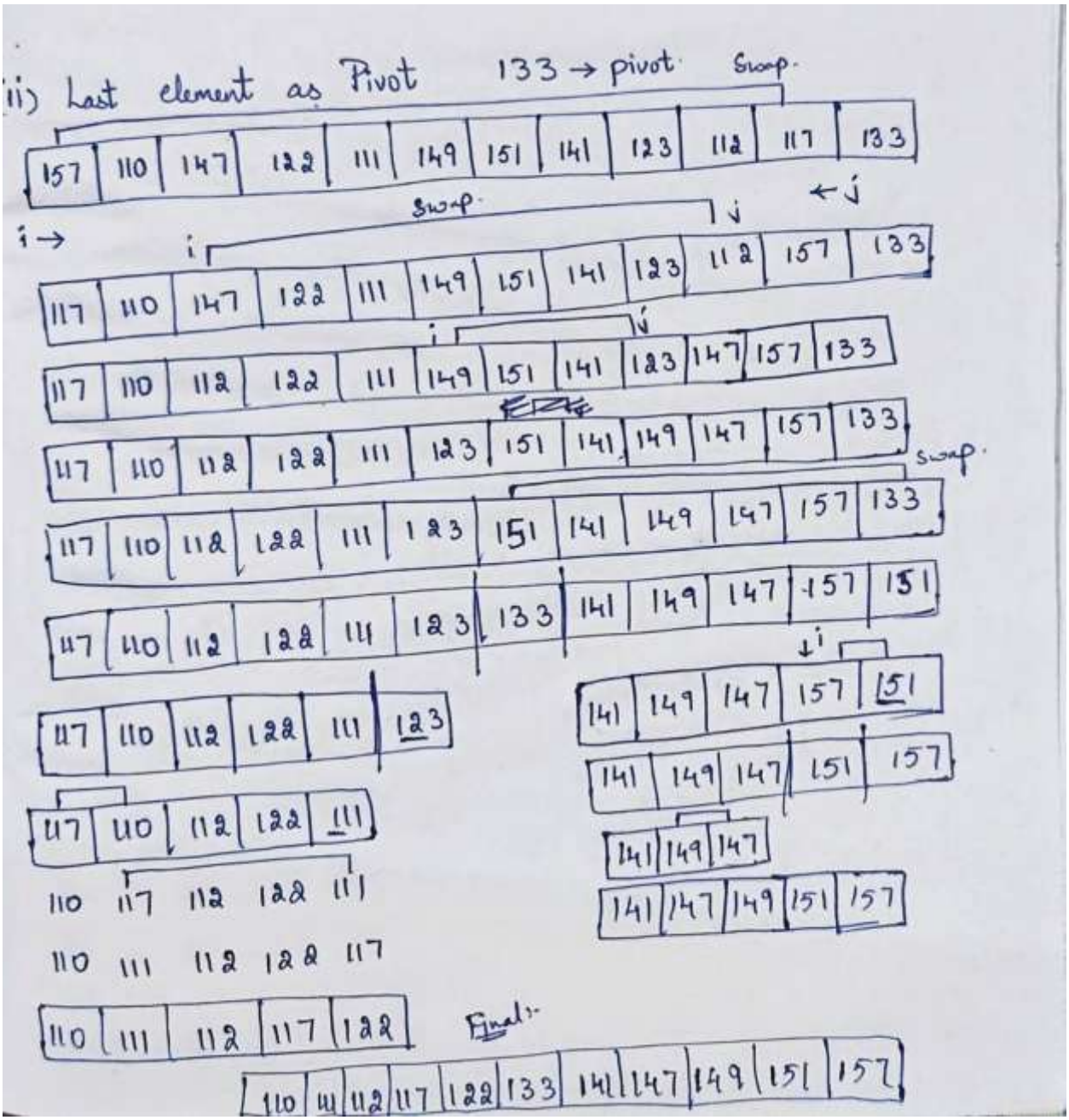
Code:

```c
//CH.SC.U4CSE24120
#include <stdio.h>
int PartitionLast(int a[], int low, int high) {
    int pivot = a[high];
    int i = low;
    int j = high - 1;
    while (1) {
        while (i <= j && a[i] <= pivot)
            i++;
        while (j >= i && a[j] > pivot)
            j--;
        if (i >= j)
            break;
        int temp = a[i];
        a[i] = a[j];
        a[j] = temp;
    }
    int temp = a[i];
    a[i] = a[high];
    a[high] = temp;
    return i;
}
void quickSort(int a[], int low, int high) {
    if (low < high) {
        int p = PartitionLast(a, low, high);
        quickSort(a, low, p - 1);
        quickSort(a, p + 1, high);
    }
}
int main() {
    int a[] = {157, 110, 147, 122, 111, 149, 151, 141, 123, 112, 117, 133};
    int n = sizeof(a) / sizeof(a[0]);
    quickSort(a, 0, n - 1);
    printf("Sorted array:\n");
    for (int i = 0; i < n; i++)
        printf("%d ", a[i]);

    return 0;
}
```

Output:

```
Sorted array:
110 111 112 117 122 123 133 141 147 149 151 157
```

Working:



Quick Sort (Random element as pivot)

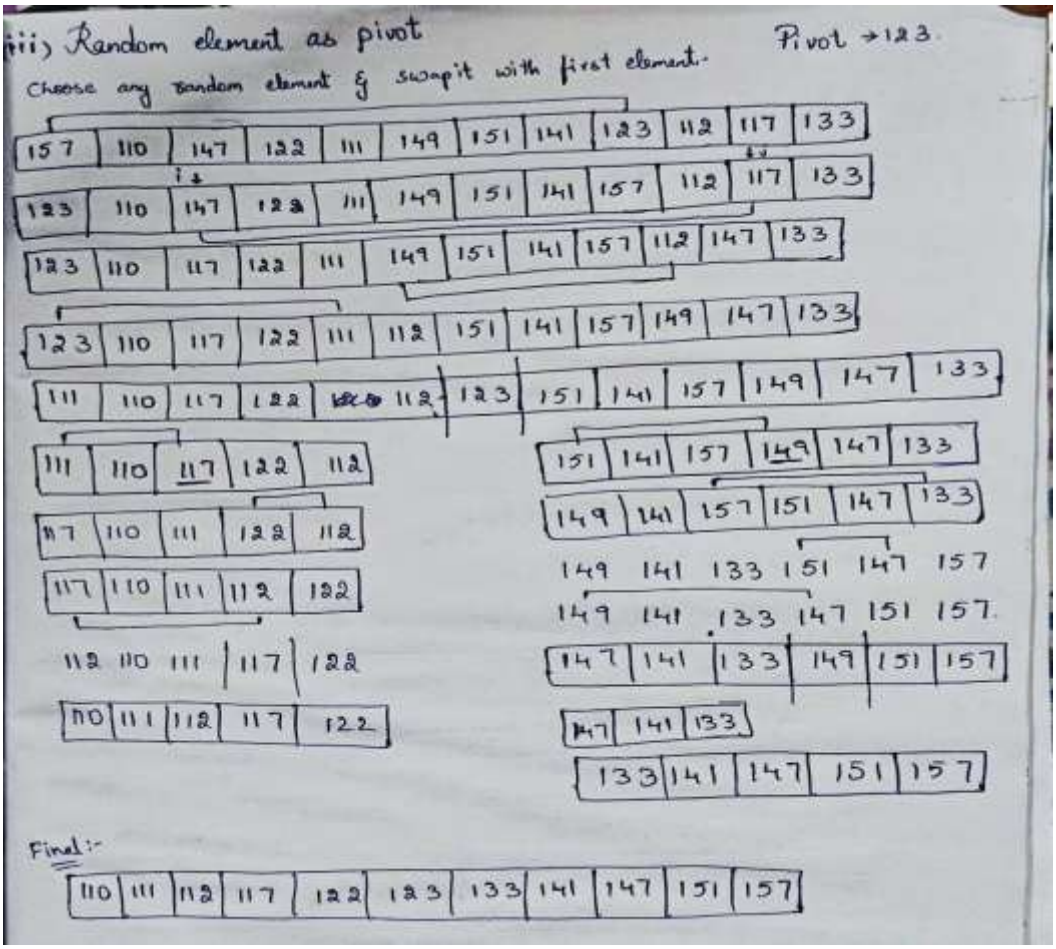
Code:

```c
//CH.SC.U4CSE24120
#include <stdio.h>
int PartitionLast(int a[], int low, int high) {
    int pivot = a[high];
    int i = low;
    int j = high - 1;
    while (1) {
        while (i <= j && a[i] <= pivot)
            i++;
        while (j >= i && a[j] > pivot)
            j--;
        if (i >= j)
            break;
        int temp = a[i];
        a[i] = a[j];
        a[j] = temp;
    }
    int temp = a[i];
    a[i] = a[high];
    a[high] = temp;
    return i;
}
void quickSort(int a[], int low, int high) {
    if (low < high) {
        int p = PartitionLast(a, low, high);
        quickSort(a, low, p - 1);
        quickSort(a, p + 1, high);
    }
}
int main() {
    int a[] = {157, 110, 147, 122, 111, 149, 151, 141, 123, 112, 117, 133};
    int n = sizeof(a) / sizeof(a[0]);
    quickSort(a, 0, n - 1);
    printf("Sorted array:\n");
    for (int i = 0; i < n; i++)
        printf("%d ", a[i]);

    return 0;
}
```

Output:

```
Sorted array:
110 111 112 117 122 123 133 141 147 149 151 157
```

Working:



Time Complexity:
Recursive partitioning divides the array into subarrays.
The recursion depth depends on how balanced the partitions are.
Balanced partitions give logarithmic depth.
Unbalanced partitions lead to linear depth.

- Best / Average Case = **O(n log n)**
- Worst Case = **O(n²)**

Space Complexity:
Recursion is used for partitioning.
Recursion depth depends on the height of the recursion tree.
Average recursion depth is logarithmic.

- Average Case = **O(log n)**
- Worst Case = **O(n)**