DESIGN AND ANALYSIS OF ALGORITHMS

LAB WORKBOOK WEEK – 6

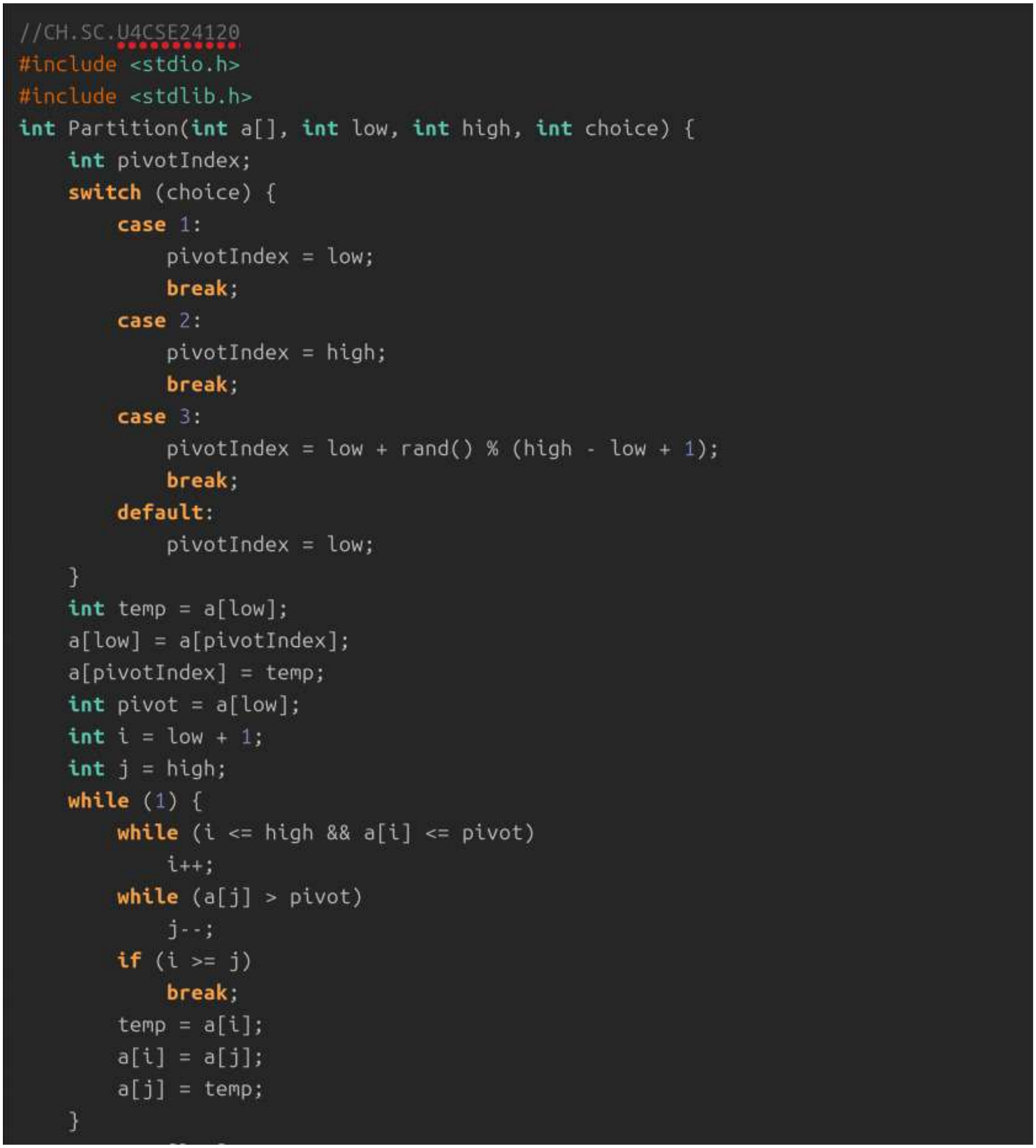
NAME: Konda Bhavani

ROLL NUMBER: CH.SC.U4CSE24120

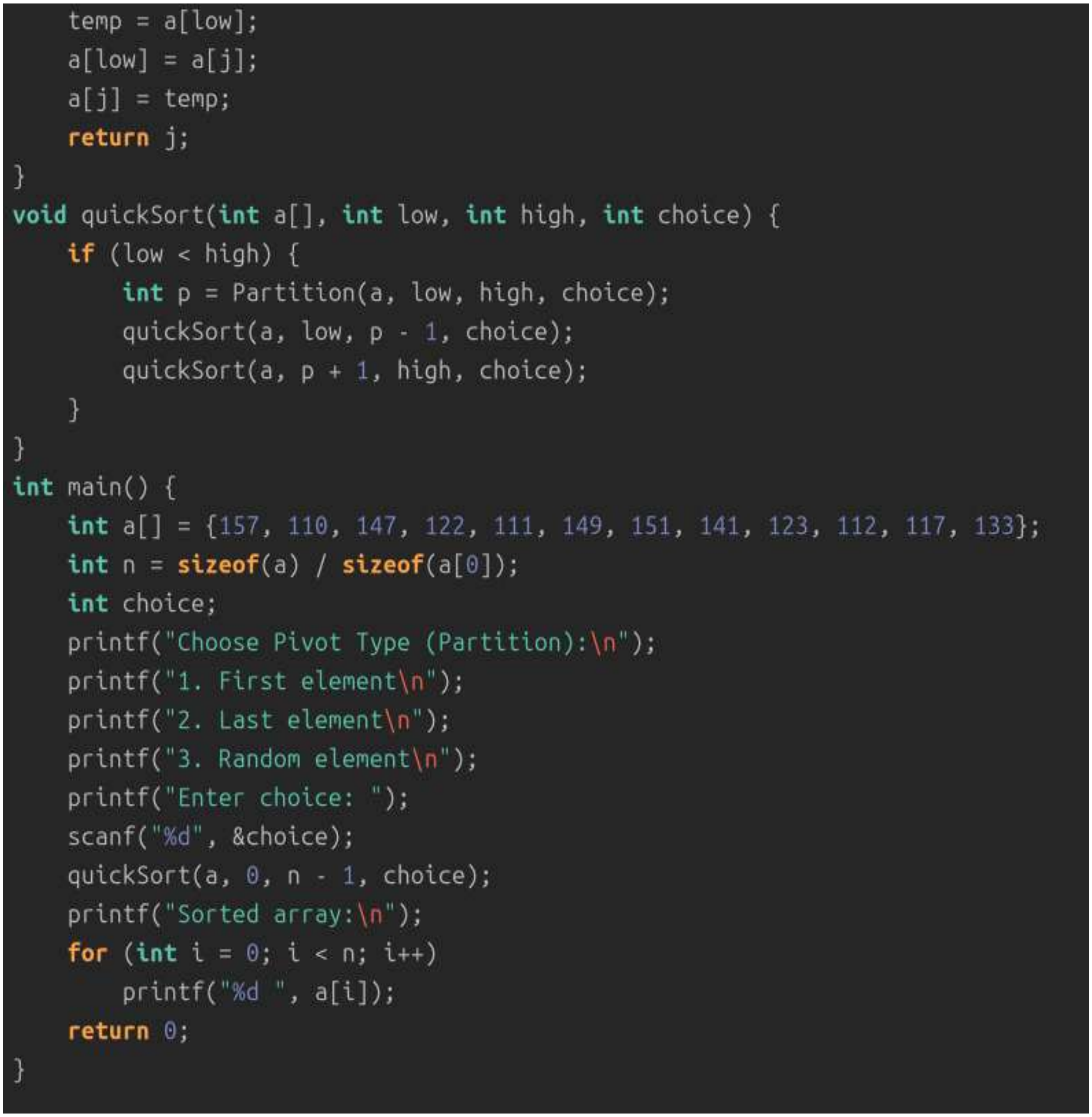
CLASS: CSE-B

Quick Sort (First element as pivot)

Code:

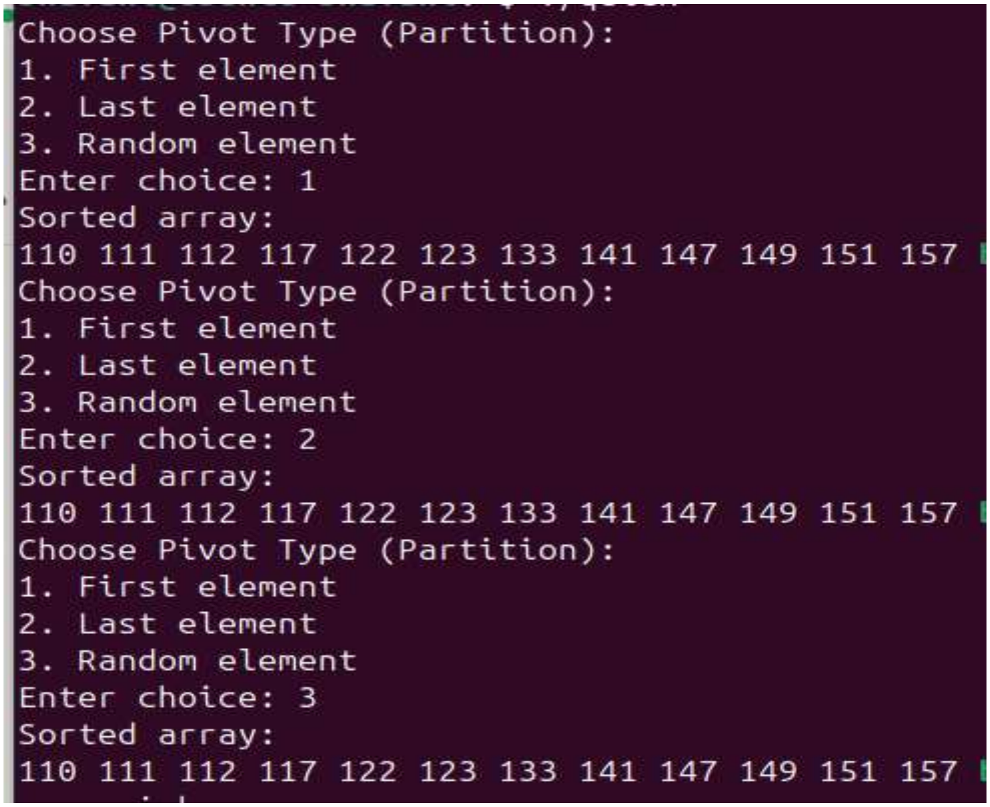


```
//CH.SC.U4CSE24120
#include <stdio.h>
#include <stdlib.h>
int Partition(int a[], int low, int high, int choice) {
    int pivotIndex;
    switch (choice) {
        case 1:
            pivotIndex = low;
            break;
        case 2:
            pivotIndex = high;
            break;
        case 3:
            pivotIndex = low + rand() % (high - low + 1);
            break;
        default:
            pivotIndex = low;
    }
    int temp = a[low];
    a[low] = a[pivotIndex];
    a[pivotIndex] = temp;
    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;
        temp = a[i];
        a[i] = a[j];
        a[j] = temp;
    }
```



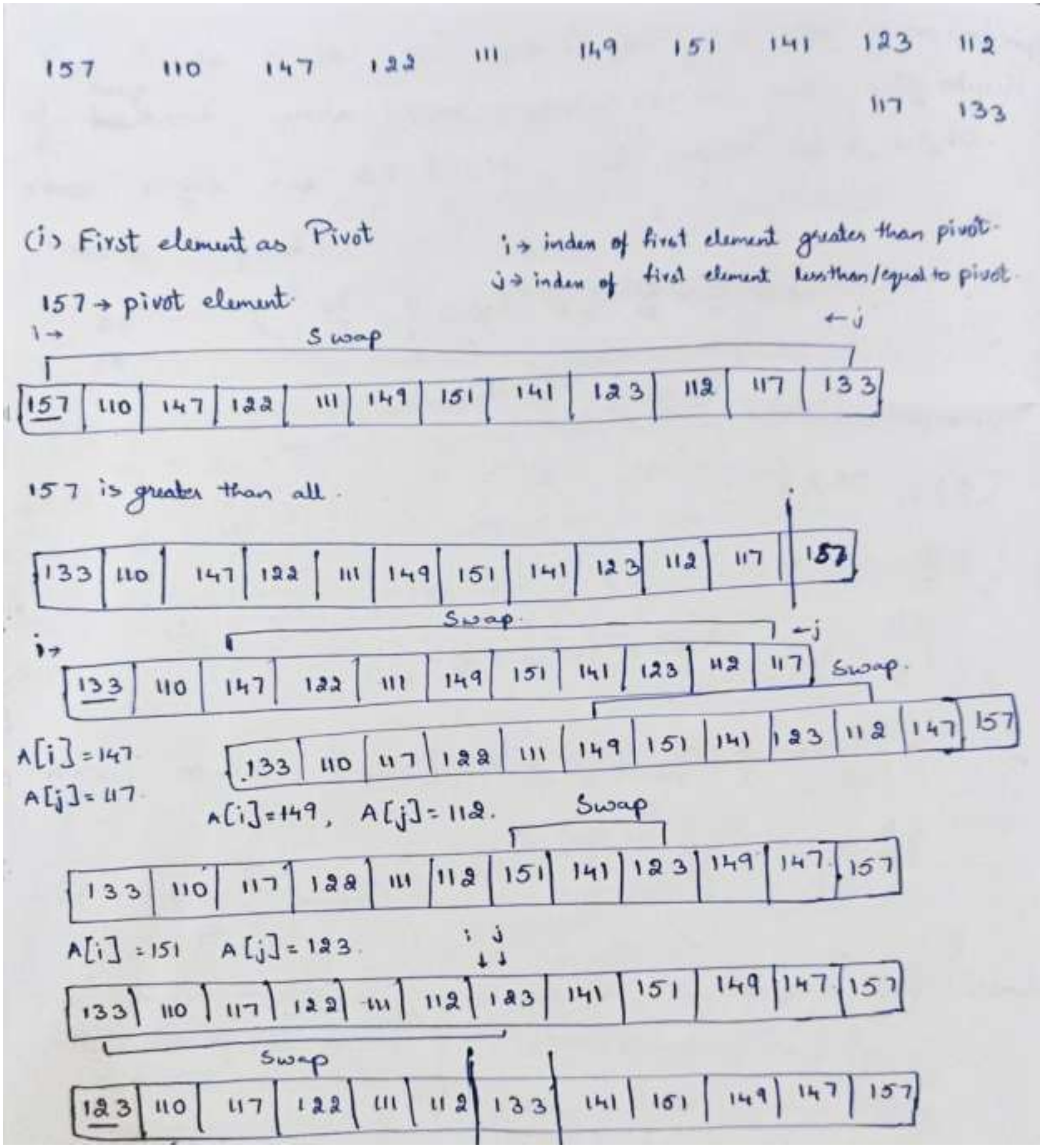
```
    temp = a[low];
    a[low] = a[j];
    a[j] = temp;
    return j;
}
void quickSort(int a[], int low, int high, int choice) {
    if (low < high) {
        int p = Partition(a, low, high, choice);
        quickSort(a, low, p - 1, choice);
        quickSort(a, p + 1, high, choice);
    }
}
int main() {
    int a[] = {157, 110, 147, 122, 111, 149, 151, 141, 123, 112, 117, 133};
    int n = sizeof(a) / sizeof(a[0]);
    int choice;
    printf("Choose Pivot Type (Partition):\n");
    printf("1. First element\n");
    printf("2. Last element\n");
    printf("3. Random element\n");
    printf("Enter choice: ");
    scanf("%d", &choice);
    quickSort(a, 0, n - 1, choice);
    printf("Sorted array:\n");
    for (int i = 0; i < n; i++)
        printf("%d ", a[i]);
    return 0;
}
```

Output:



```
Choose Pivot Type (Partition):
1. First element
2. Last element
3. Random element
Enter choice: 1
Sorted array:
110 111 112 117 122 123 133 141 147 149 151 157
Choose Pivot Type (Partition):
1. First element
2. Last element
3. Random element
Enter choice: 2
Sorted array:
110 111 112 117 122 123 133 141 147 149 151 157
Choose Pivot Type (Partition):
1. First element
2. Last element
3. Random element
Enter choice: 3
Sorted array:
110 111 112 117 122 123 133 141 147 149 151 157
```

Working:



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

(i) First element as Pivot

i → index of first element greater than pivot.
j → index of first element lessthan/equal to pivot.

157 → pivot element.

i → Swap ← j

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

157 is greater than all.

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

i → Swap. ← j

133	110	147	122	111	149	151	141	123	112	117

Swap.

A[i] = 147
A[j] = 117

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

A[i] = 149, A[j] = 112. Swap

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

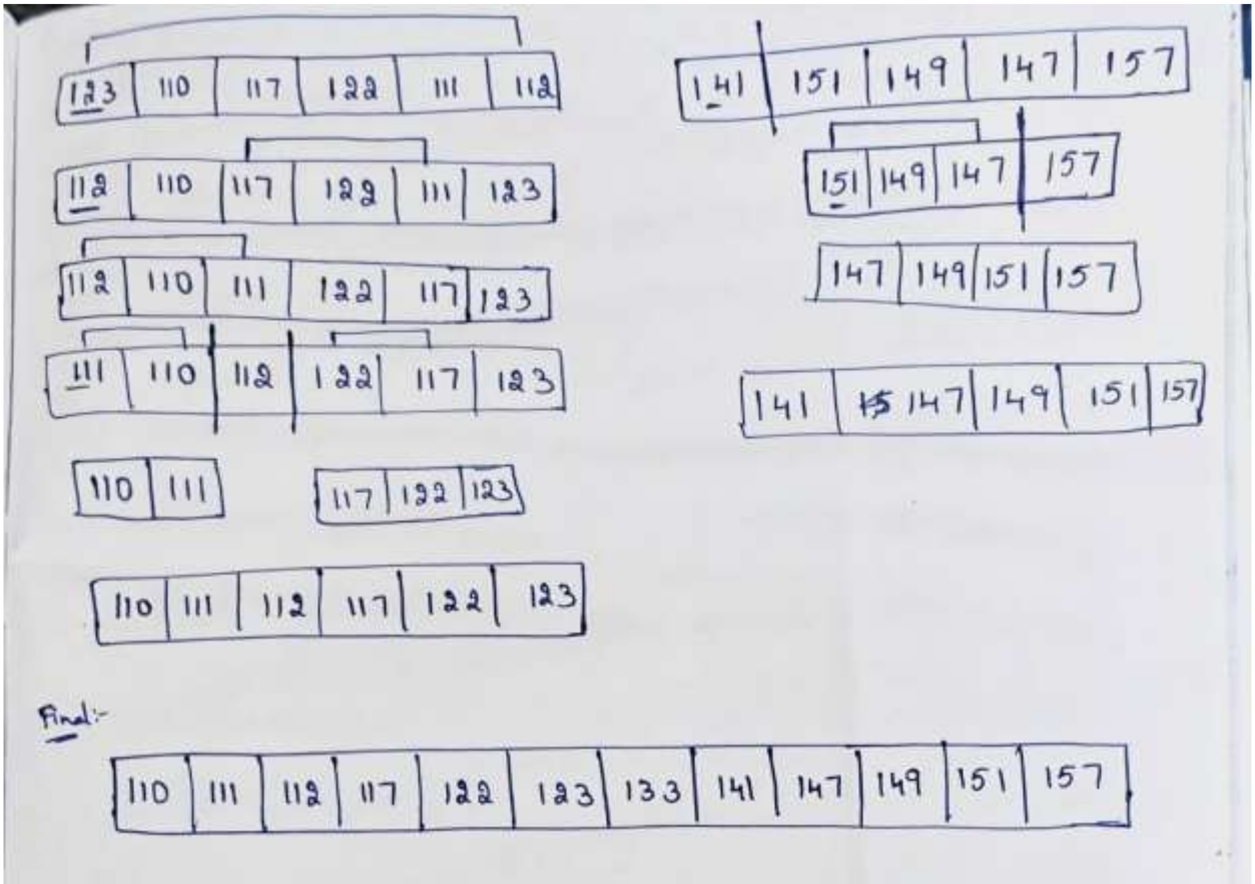
A[i] = 151 A[j] = 123.

i j

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

Swap

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



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

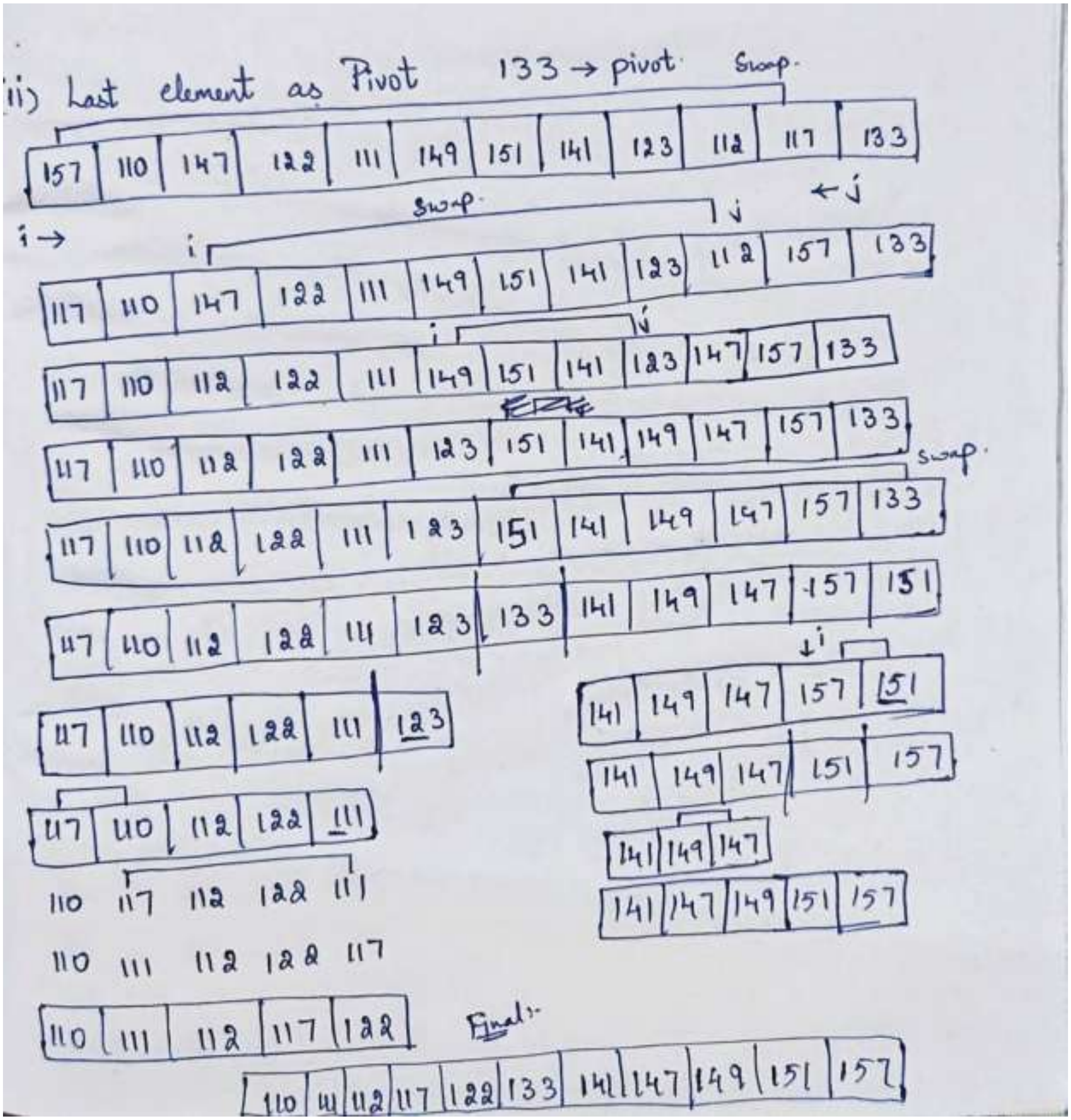
151 | 149 | 147 | 157

147 | 149 | 151 | 157

141 | ~~15~~ 147 | 149 | 151 | 157

Final:-

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



(ii) Last element as Pivot   133 → pivot   Swap.

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

i →   j   ← j

swap.

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

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

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

swap.

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

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

117 | 110 | 112 | 122 | 111 | 123

117 | 110 | 112 | 122 | 111

110 117 112 122 111

110 111 112 122 117

110 | 111 | 112 | 117 | 122

i

141 | 149 | 147 | 157 | 151

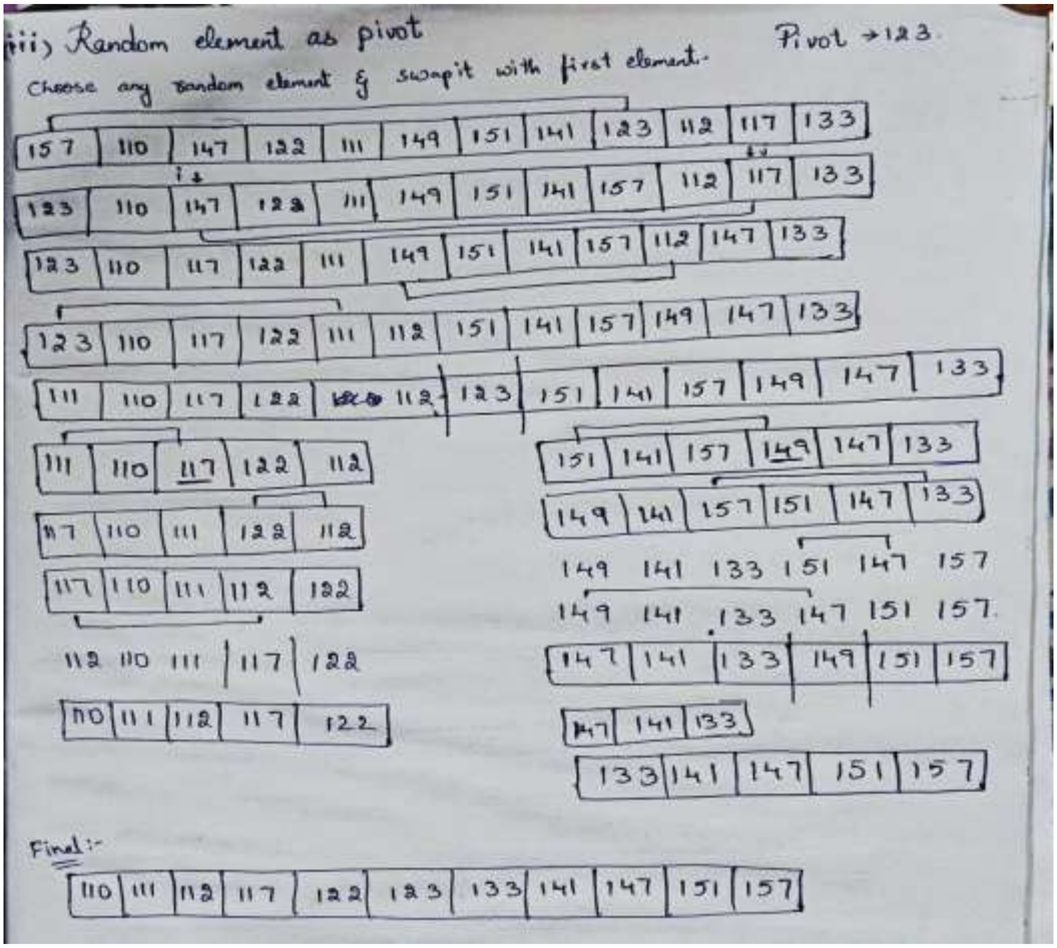
141 | 149 | 147 | 151 | 157

141 | 149 | 147

141 | 147 | 149 | 151 | 157

Final:-

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



iii) Random element as pivot Pivot → 123.

Choose any random element & swap it with first element.

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

i↓ ↓↓

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

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

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

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

111 | 110 | 117 | 122 | 112

117 | 110 | 111 | 122 | 112

117 | 110 | 111 | 112 | 122

112 110 111 | 117 | 122

110 | 111 | 112 | 117 | 122

151 | 141 | 157 | 149 | 147 | 133

149 | 141 | 157 | 151 | 147 | 133

149 141 133 151 147 157

149 141 133 147 151 157.

147 | 141 | 133 | 149 | 151 | 157

147 | 141 | 133

133 | 141 | 147 | 151 | 157

Final:-

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

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 = **$O(n^2)$**

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)**