AMRITA SCHOOL OF COMPUTING

DESIGN AND ANALYSIS OF ALGORITHMS

(23CSE211)

Name: CHANTATI SAI PURNISHA MAHI

Roll No.: CH.SC.U4CSE24156

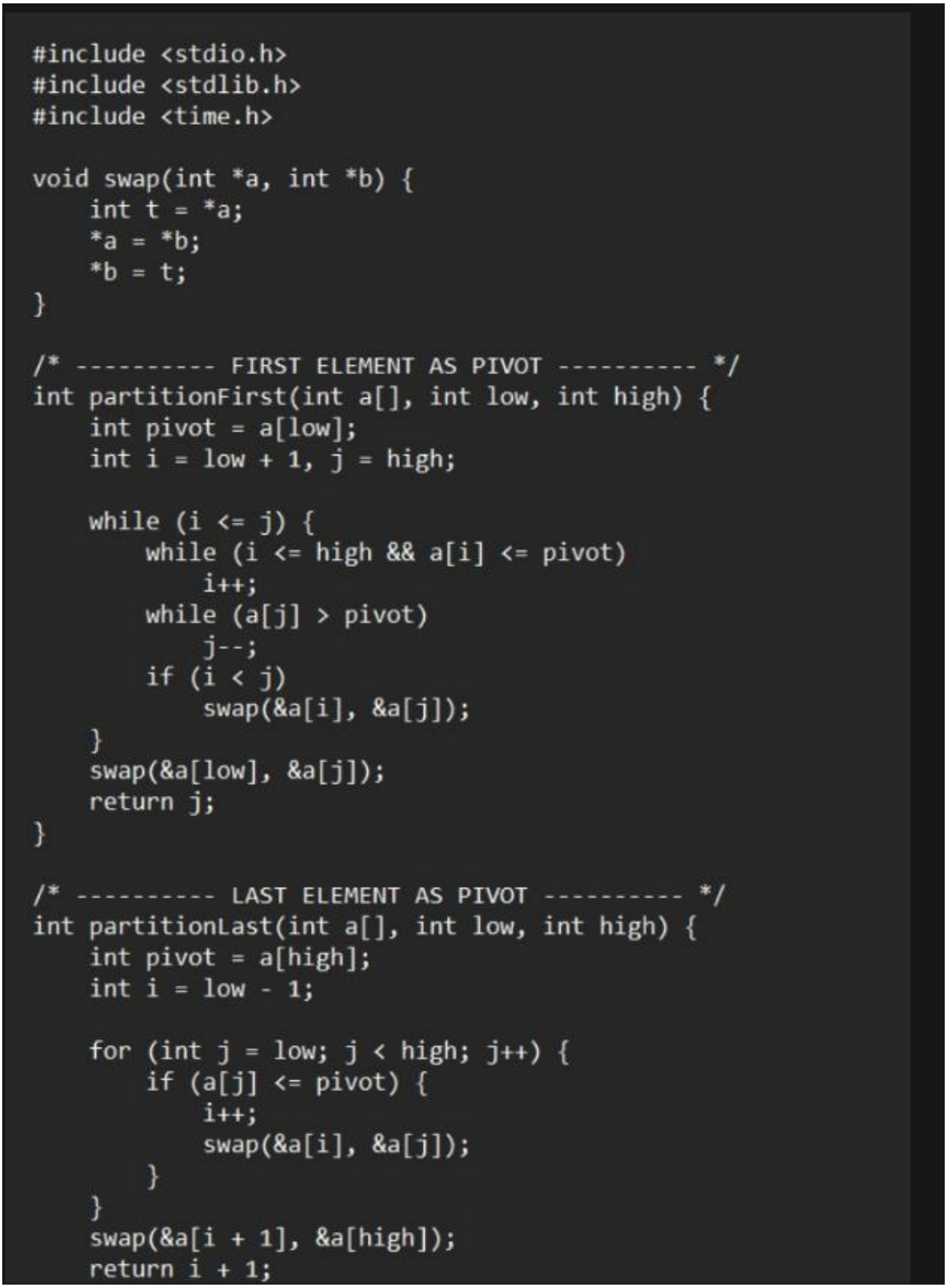
Class: BTech (CSE-B)

School: Amrita School of Computing,
Chennai Campus.

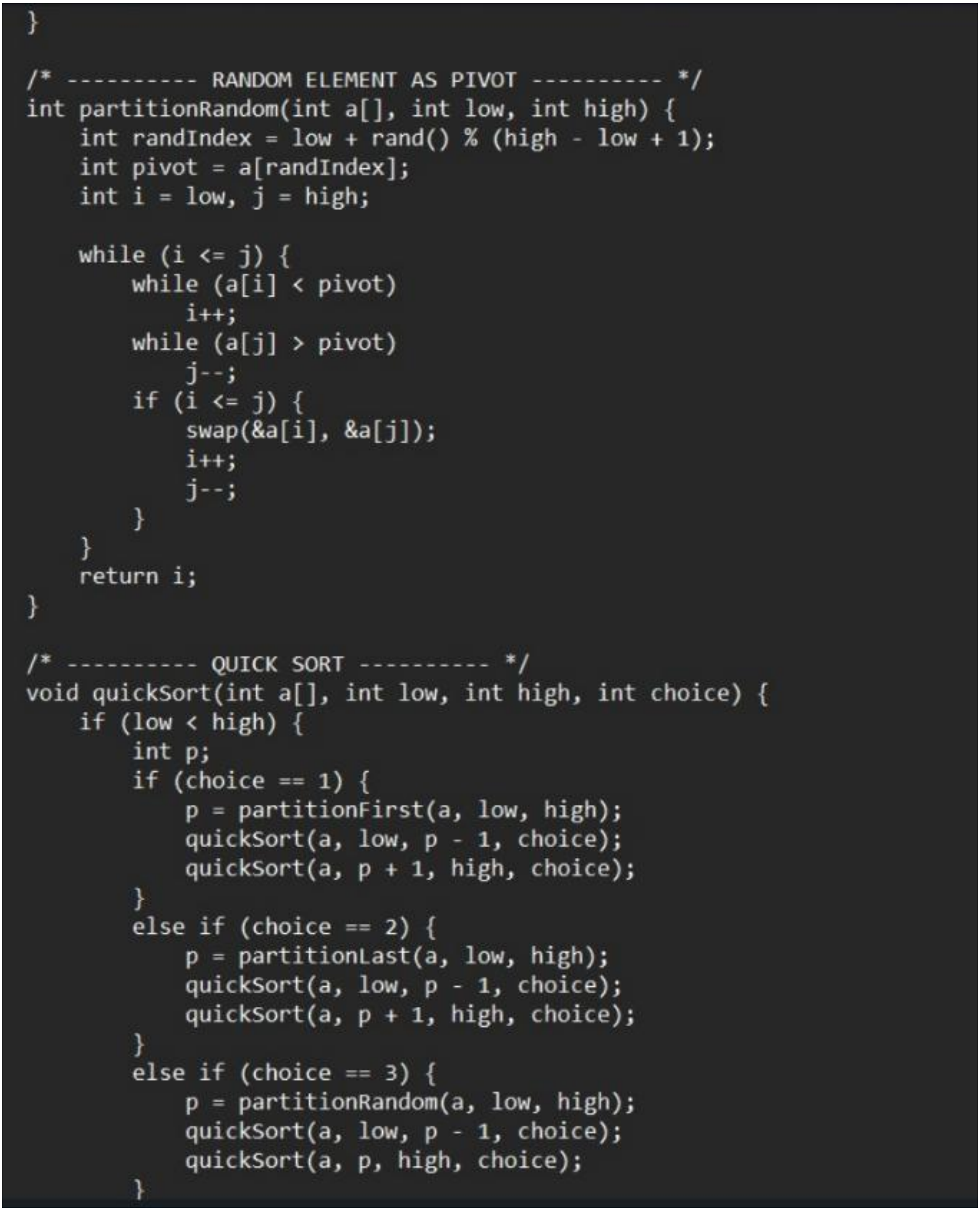
WEEK -6:-

1) Quick Sort using first, last, and random pivot selection methods. Design a menu-driven program that allows the user to choose any method, prints the randomly selected pivot.

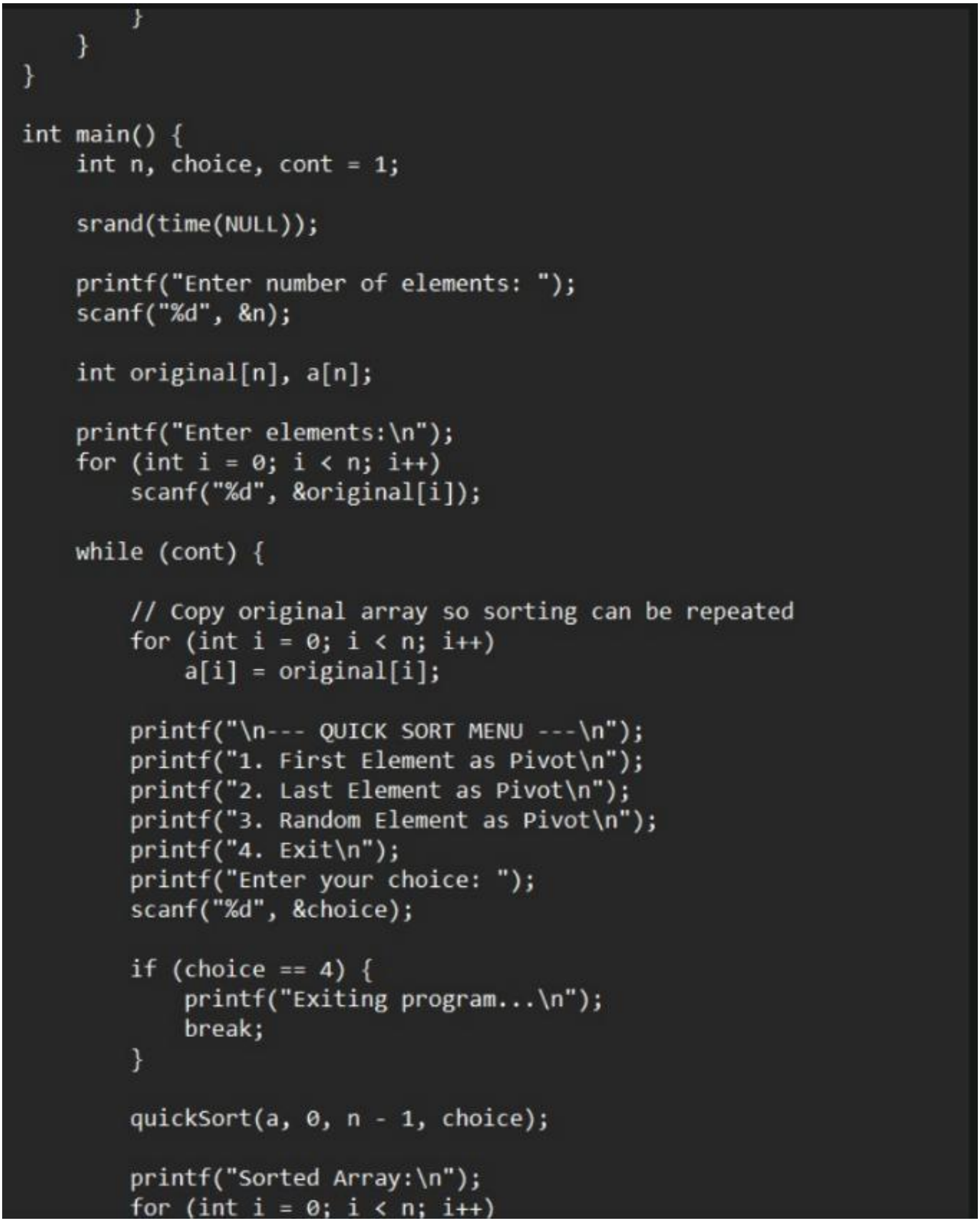
Code:



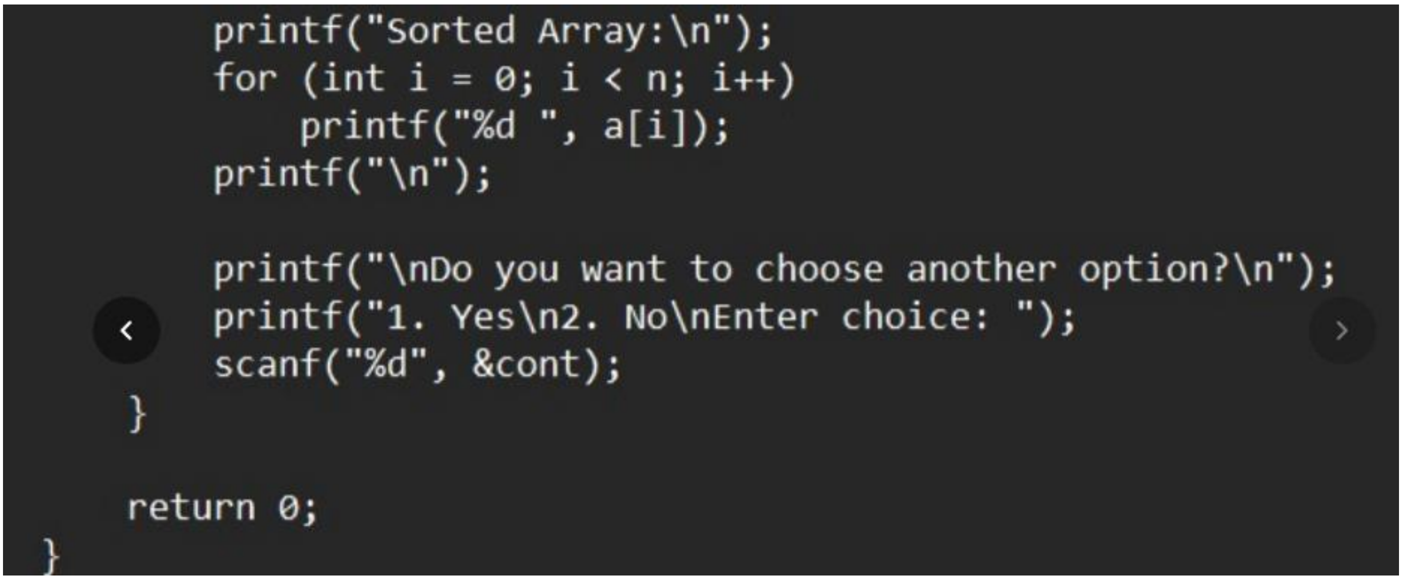
```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

void swap(int *a, int *b) {
    int t = *a;
    *a = *b;
    *b = t;
}

/* ---------- FIRST ELEMENT AS PIVOT ---------- */
int partitionFirst(int a[], int low, int high) {
    int pivot = a[low];
    int i = low + 1, j = high;

    while (i <= j) {
        while (i <= high && a[i] <= pivot)
            i++;
        while (a[j] > pivot)
            j--;
        if (i < j)
            swap(&a[i], &a[j]);
    }
    swap(&a[low], &a[j]);
    return j;
}

/* ---------- LAST ELEMENT AS PIVOT ---------- */
int partitionLast(int a[], int low, int high) {
    int pivot = a[high];
    int i = low - 1;

    for (int j = low; j < high; j++) {
        if (a[j] <= pivot) {
            i++;
            swap(&a[i], &a[j]);
        }
    }
    swap(&a[i + 1], &a[high]);
    return i + 1;
```



```
}

/* ---------- RANDOM ELEMENT AS PIVOT ---------- */
int partitionRandom(int a[], int low, int high) {
    int randIndex = low + rand() % (high - low + 1);
    int pivot = a[randIndex];
    int i = low, j = high;

    while (i <= j) {
        while (a[i] < pivot)
            i++;
        while (a[j] > pivot)
            j--;
        if (i <= j) {
            swap(&a[i], &a[j]);
            i++;
            j--;
        }
    }
    return i;
}

/* ---------- QUICK SORT ---------- */
void quickSort(int a[], int low, int high, int choice) {
    if (low < high) {
        int p;
        if (choice == 1) {
            p = partitionFirst(a, low, high);
            quickSort(a, low, p - 1, choice);
            quickSort(a, p + 1, high, choice);
        }
        else if (choice == 2) {
            p = partitionLast(a, low, high);
            quickSort(a, low, p - 1, choice);
            quickSort(a, p + 1, high, choice);
        }
        else if (choice == 3) {
            p = partitionRandom(a, low, high);
            quickSort(a, low, p - 1, choice);
            quickSort(a, p, high, choice);
        }
```

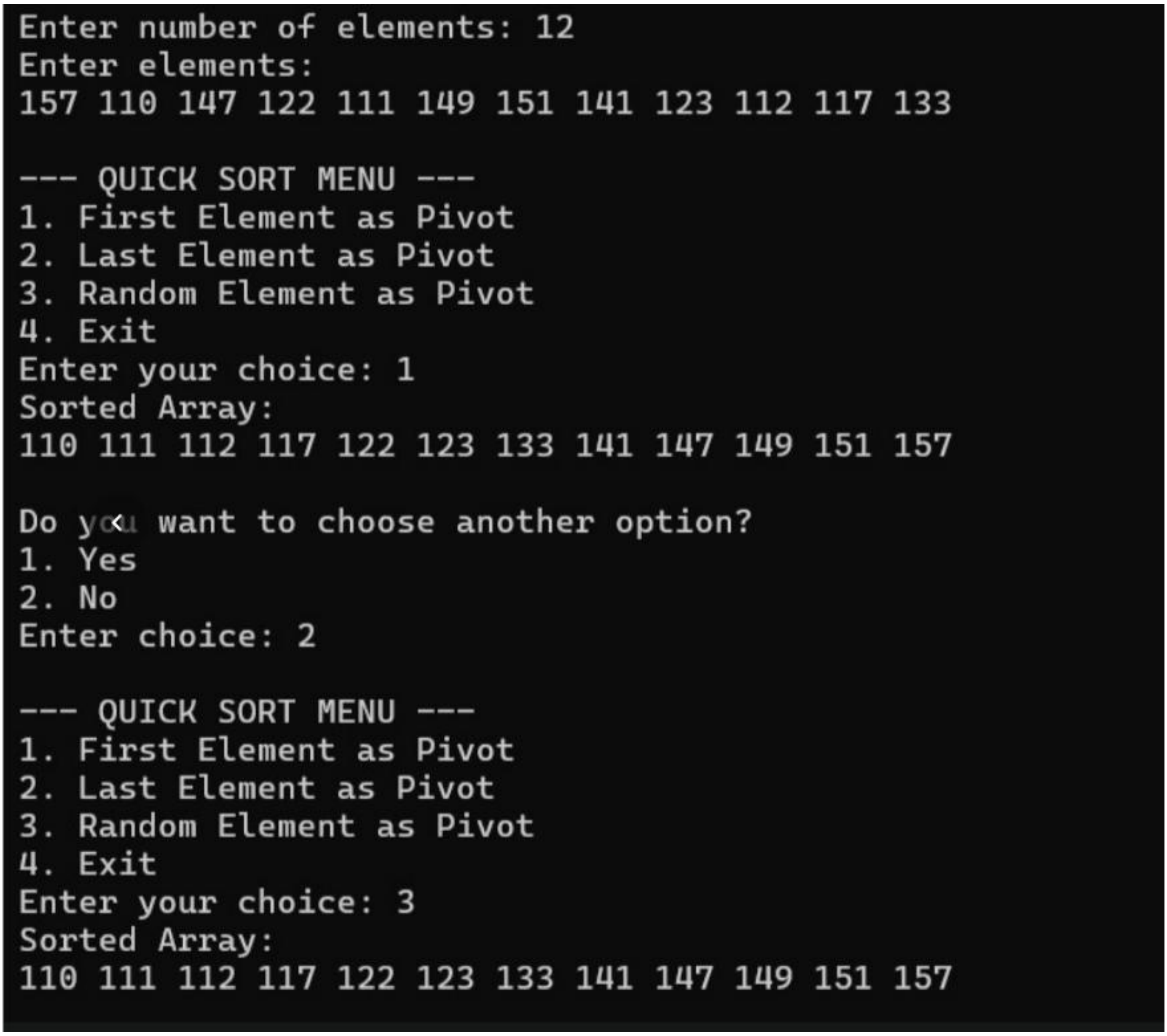


```
        }
    }
}

int main() {
    int n, choice, cont = 1;

    srand(time(NULL));

    printf("Enter number of elements: ");
    scanf("%d", &n);

    int original[n], a[n];

    printf("Enter elements:\n");
    for (int i = 0; i < n; i++)
        scanf("%d", &original[i]);

    while (cont) {

        // Copy original array so sorting can be repeated
        for (int i = 0; i < n; i++)
            a[i] = original[i];

        printf("\n--- QUICK SORT MENU ---\n");
        printf("1. First Element as Pivot\n");
        printf("2. Last Element as Pivot\n");
        printf("3. Random Element as Pivot\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        if (choice == 4) {
            printf("Exiting program...\n");
            break;
        }

        quickSort(a, 0, n - 1, choice);

        printf("Sorted Array:\n");
        for (int i = 0; i < n; i++)
```



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

        printf("\nDo you want to choose another option?\n");
        printf("1. Yes\n2. No\nEnter choice: ");
        scanf("%d", &cont);
    }

    return 0;
}
```

OUTPUT:-



```
Enter number of elements: 12
Enter elements:
157 110 147 122 111 149 151 141 123 112 117 133

--- QUICK SORT MENU ---
1. First Element as Pivot
2. Last Element as Pivot
3. Random Element as Pivot
4. Exit
Enter your choice: 1
Sorted Array:
110 111 112 117 122 123 133 141 147 149 151 157

Do you want to choose another option?
1. Yes
2. No
Enter choice: 2

--- QUICK SORT MENU ---
1. First Element as Pivot
2. Last Element as Pivot
3. Random Element as Pivot
4. Exit
Enter your choice: 3
Sorted Array:
110 111 112 117 122 123 133 141 147 149 151 157
```

TIME COMPLEXITY:- O(n log n)

JUSTIFICATION:-

Quick Sort divides the array into two parts at each partition step. On average, the pivot divides the array into nearly equal halves. Each partition operation takes O(n) time to compare elements. The recursion depth is log n. Therefore, total time complexity = n × log n = O(n log n).

SPACE COMPLEXITY:- O(log n)

JUSTIFICATION:-

Quick Sort uses recursion stack memory.

Each recursive call stores:
- int low → 4 bytes
- int high → 4 bytes
- int p (partition index) → 4 bytes
- return address & control data → approx 8 bytes

Total per call ≈ 20 bytes

Maximum recursion depth (average case) = log n

Total stack memory = 20 × log n bytes

Hence space complexity = O(log n).

Random pivot is the most efficient.

1. **Avoids worst-case performance:**
   Choosing first or last element as pivot can cause worst-case time complexity **O(n^2)** for already sorted or reverse-sorted arrays.
2. **Gives balanced partitions:**
   Random pivot usually divides the array into nearly equal halves, improving efficiency.
3. **Improves average time complexity:**
   With random pivot, Quick Sort works close to its best case **O(n log n)**.
4. **Input independent:**
   Performance does not depend on input order (sorted, reverse, or random).

Quick sort

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

i, First element pivot

ii, Last element pivot

iii. Random element

i Method:-

Pivot = first element

i = left + 1 index

j = right index

Move i ⟶ right until A[i] >pivot

Move j ⟶ left until A[j] <pivot

If i<j ⟶ swap A[i], A[j]

If i≥j ⟶ swap pivot with A[j]

Given Array:-

157	110	147	122	111	149	151	141	123	112	117	133
0	1	2	3	4	5	6	7	8	9	10	11

Step1:- For array [0, 11]

Pass 1:- Pivot element = 157

i=1 j=11

As no element greater than 157, i=11 & j=11 i=j .

∴ swap 157 ⟷ 133.

133	110	147	122	111	149	151	141	123	112	117	157
0	1	2	3	4	5	6	7	8	9	10	11

Pass 2:- Pivot element = 133 i=1 j=10 Step 2:- For array [0,10]

As 147 > 133 i=2

As 133 > 117 j=10 As i<j swap 147 ⟷ 117

133	110	117	122	111	149	151	141	123	112	147	157
0	1	2	3	4	5	6	7	8	9	10	11

Pass 2:- Pivot element = 133 i=2 j= 10

AS 149>133 i= 5

i<j swap 149 ⟷ 112

AS 133 > 112 j = 9

133	110	117	122	111	112	151	141	123	149	147	157
0	1	2	3	4	5	6	7	8	9	10	11

Pass 3:- Pivot element 133 i= 5 j= 9

AS 151 > 133 i= 6

i<j swap 151 ⟷ 123

AS 133 > 123 j =8

133	110	117	122	111	112	123	141	151	149	147	157
0	1	2	3	4	5	6	7	8	9	10	11

Pass 4:- Pivot element 133 i= 6 j=8

AS 141> 133 i = 7

i ≥ j swap 133 ⟷ 123

AS 133 > 123 j = 6

123	110	117	122	111	112	133	141	151	149	147	157
0	1	2	3	4	5	6	7	8	9	10	11

Step 3:- Pivot element = 123 Parition [0, 5]

Pass 1:- i= 1 j = 5

AS no element is greater 123 in [0, 5] array

i= 5 j = 5 i=j swap 123 ⟷ 112

112	110	117	122	111	123
0	1	2	3	4	5

Step 4:- Pivot element = 112 Parition [0 4] array

Pass 1:- $i = 1$ $j = 4$

AS $117 > 112$ $i = 2$ $i \nless j$ swap 117 ⟷ 111

AS $111 < 112$ $j = 4$

112	110	111	122	117
0	1	2	3	4

Pass 2:- $i = 2$ $j = 4$

AS $122 > 112$ $i = 3$ $i \geq j$ swap 112 ⟷ 111

$111 < 112$ $j = 2$

111	110	112	122	117
0	1	2	3	4

Step5:- Pivot element = 111 Parition [0 1] array

Pass 1 :- $i = 1$ $j = 1$

AS $i = j$ swap 111 ⟷ 110

110	111
0	1

[Sorted]

Step6:- Pivot elenent = 122 Parition [3 4] array.

Pass 1:- $i = 4$ $j = 4$

AS $i = j$ swap 122 ⟷ 117.

122

110	111	112	117	122
0	1	2	3	4

Step 7:- Pivot element = 141 Parition [7 10] array.

Pass 1:- $i = 8$ $j = 10$

AS no element lesser than 141 $j = 7$ means cannot swap with element 141 is already sorted.

133 / 110 / 117 122 / 4·

110	111	112	117	122	123	193	141
0	1	2	3	4	5	6	7

Step 8:- Piviot element = 151 [8, 10] parition

i = 9 j = 10

As no eloment greater i = 10 j = 10

swap 151 ⟷ 147

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

iii, Last element:-

157	110	147	122	111	144	151	141	123	112	117	133
0	1	2	3	4	5	6	7	8	9	10	11

Method:-

1) Pivot = A [high]

2) i = first / inde low

3) j = high - 1

4) Move i → right while A[I] < pivot

5) Move j → left while A[j] > pivot

6) If i < j → swap A (i), A [j]

7) If i ≥ j → swap A[j], pivot

Step 1:- Pivot = 133 [0 11] array
Pass 1:-

i = 0 j = 10

157 > 133 i = 0

j stops at 117 (j = 10) swap 157 ⟷ 117

117	110	147	122	111	149	151	141	123	112	157	133
0	1	2	3	4	5	6	7	8	9	10	11

Pass 2:-

i → stops at 147 (i=2)
j → stops at 112 (j= 9)

swap 147 ⟷ 112

117	110	112	122	111	149	151	141	123	147	157	133
0	1	2	3	4	5	6	7	8	9	10	11

Pass 3:-

i → stops at 149 (i=5)
j → stops at 123 (j=8)

swap 149 ⟷ 123

117	110	112	122	111	123	151	141	149	147	151	133
0	1	2	3	4	5	6	7	8	9	10	11

Pass 4:-

i → stops at 151 (i=6)
j → stop at 123 (j=5)

swap pivot 133 ⟷ A[i] 151

117	110	112	122	111	123	133	141	149	147	157	151
0	1	2	3	4	5	6	7	8	9	10	11

Step 2:- Pivot element = 123 [0 5] array

Pass 1:-

i → stops at 123 (i=5)
j → stops at 111 (j=4)

swap 123 ⟷ 123
cannot happen pivot already sortea.

117	110	112	122	117	123
0	1	2	3	4	5

Step3:- Pivot element = 111 [0 4] array

Pass 1:-

i → stops at i = 0

j → stops at j = 1

swap 117 → 110

i	j			p
110	117	112	122	111
0	1	2	3	4

Pass 2:-

i → stop at i = 1

j → stop at j = 1

swap 117 ↔ 111

110	111	112	122	117
0	1	2	3	4

Step4:- Pivot element = 117 [2.4] array.

i → stops at i = 3

j → stops at i = 2

Swap 122 ↔ 117

110	111	112	117	122
0	1	2	3	4

Steps:- Pivot element = 151 [7 11] array.

i → stops at 9

j → stop at at 90

swap 157 ↔ 151

110	111	112	117	122	123	133	141	149	147	151	157
0	1	2	3	4	5	6	7	8	9	10	11

Step 6:- Pivot element = 147 [7 9] array.

i → stops at 8

j → Stop at 7 Swap 149 <-> 147

110	111	112	117	122	123	133	141	147	149	151	157
0	1	2	3	4	5	6	7	8	9	10	11

iii) Random Element as pivot element.

Method/Logic:-

1) choose random index

2) Swap with first element

3) Use same method used in first element.

Sol:-

157	110	147	122	111	149	151	141	123	112	117	133.
0	1	2	3	4	5	6	7	8	9	10	11

Step 1:- Take 141 as pivot element

Swap with first element.

141	110	147	122	111	149	151	157	123	112	117	133.
0	1	2	3	4	5	6	7	8	9	10	11

Pass 1:-

i stops at 147

j stops at 133 Swap 147 133

141	110	133	122	111	149	151	157	123	112	117	147
0	1	2	3	4	5	6	7	8	9	10	11

Pass 2:-

i stops at 149

j stops at 133 Swaps 149 117

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

Pass 3:-

i stops at 151

j stops at 112

swap 151 ↔ 112

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

Pass 4:-

i stops at 157

j stops at 123

swap 157 ↔ 123

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

Pivot index = 7

Left subbarray [0 6]

Step 2:- Let subarray [0 6].

Pass 1:-

123	110	133	122	111	117	112

Random Pivot = 117

After swap:-

117	110	133	122	111	123	112

i stops at 1

j stops at 6

swap 133 ↔ 112

117	110	112	122	111	123	133

Pass 2:-

i stops at 122

j stops at 111

swap6 122 ↔ 111

117	110	112	111	122	123	133

Pass 3:

i = 4

j = 3

i ≥ j → STOP

Swap 111 ⟷ 117.

Pivot index / 3.

111	110	112	117	122	123	133

left subbarray [0 2]

Step 3:- subbarray [0, 2] left of 117.

Take pivot = 110

swap with 111

110	111	112.

Pass 1:-

i = 1 j = 2 ⇒ i ≥ j ⇒ stop.

Already sorted.

Righ of 117 ⇒ already sorted ✓

Right of 141 ⇒

157	151	149	147.

Step 4:- subbarray [8, 11]

157	151	149	147

Take pivot = 149

swap with 157.

149	151	157	147.

Pass 1:-

i = 9

j = 11

Swaps 147 ⟷ 151

149	147	157	151

Pass 2:-

i = 10

j = 9

Swap 149 ⟷ 147

147	149	157	151

Steps:- Right of 149 [10, 11]

157 151

Pivot = 151

After sawap 151 157.

sorted ✓

Finally:

110	111	112	117	122	123	133	141	149	144	151	157.

At last Random pivot is most efficient thant first or last.

1. Avoids worst-case performance.
2. Gives balanced paritions.
3. Improve average time complexity.
4. Better practical performance.