

City University

Department of CSE

Course Code : CSE 3116

Course Title : Algorithms Laboratory

Submitted To

Debobrata Chakraborty

Lecturer

Department of CSE

City University, Khagan, Savar, Dhaka

Submitted By

Name : Fariha Ahmed Riya

ID : 0272320005101296

Batch & Section : 63 C

Department : CSE

City University, Khagan, Savar, Dhaka

Linear Search

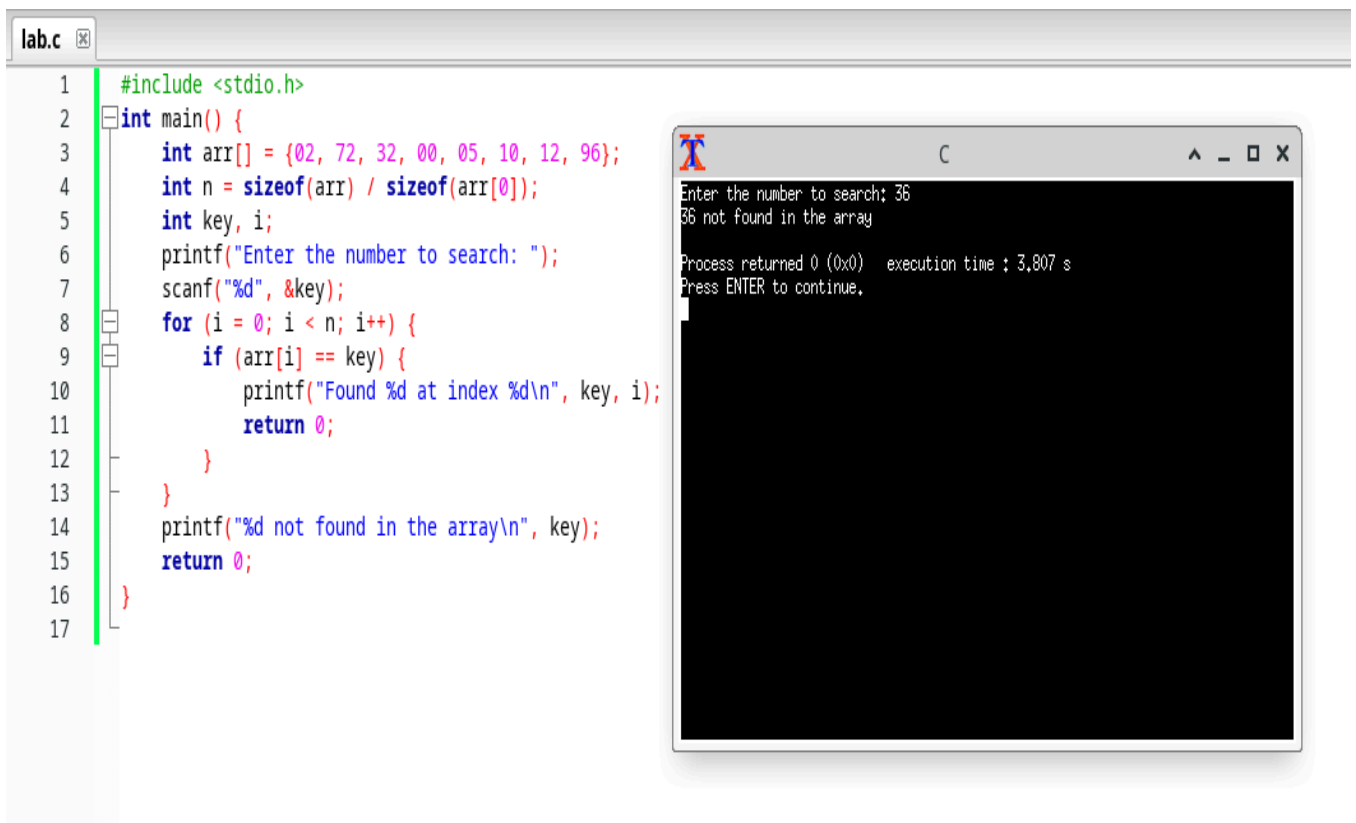
The code iterates through the array and checks for a match with the key. It prints the index if found and breaks. This is a standard linear search implementation.

The Source Code :

```
#include <stdio.h>

int main() {
    int arr[] = {02, 72, 32, 00, 05, 10, 12, 96};
    int n = sizeof(arr) / sizeof(arr[0]);
    int key, i;
    printf("Enter the number to search: ");
    scanf("%d", &key);
    for (i = 0; i < n; i++) {
        if (arr[i] == key) {
            printf("Found %d at index %d\n", key, i);
            return 0;
        }
    }
    printf("%d not found in the array\n", key);
    return 0;
}
```

Screenshot of the Linear Search Code:



The image shows a C program in a code editor and its execution output in a terminal window.

Code Editor (lab.c):

```
1  #include <stdio.h>
2  int main() {
3      int arr[] = {02, 72, 32, 00, 05, 10, 12, 96};
4      int n = sizeof(arr) / sizeof(arr[0]);
5      int key, i;
6      printf("Enter the number to search: ");
7      scanf("%d", &key);
8      for (i = 0; i < n; i++) {
9          if (arr[i] == key) {
10             printf("Found %d at index %d\n", key, i);
11             return 0;
12         }
13     }
14     printf("%d not found in the array\n", key);
15     return 0;
16 }
17
```

Terminal Window (C):

```
Enter the number to search: 36
36 not found in the array

Process returned 0 (0x0)   execution time : 3.807 s
Press ENTER to continue.
```

Binary Search

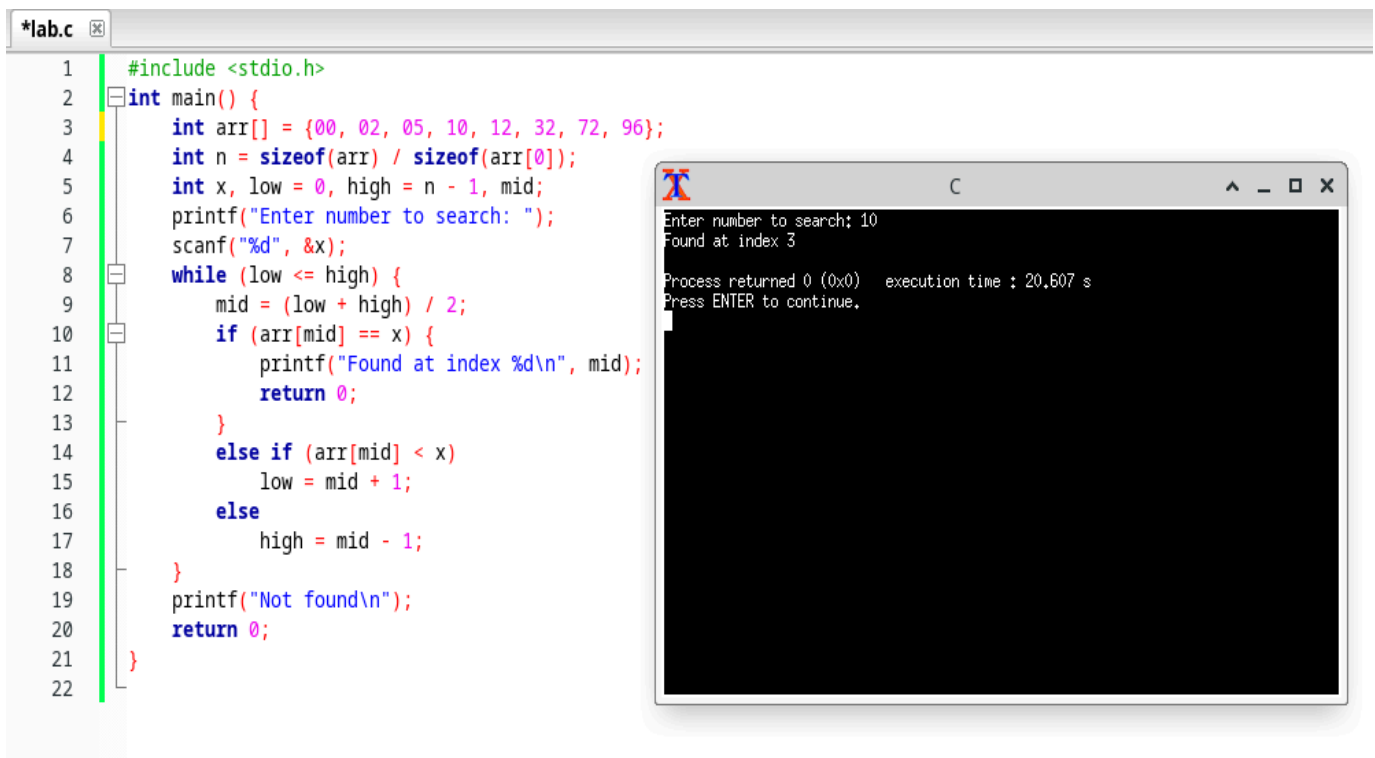
This assumes the array is sorted. The provided code uses a sorted version ([00, 02, 05, 10, 12, 32, 72, 96]). It correctly implements the binary search algorithm with a while loop to find the target value. Note that for the original unsorted array, you need to sort it first or use Linear Search instead.

Source Code :

```
#include <stdio.h>

int main() {
    int arr[] = {00, 02, 05, 10, 12, 32, 72, 96}; // Assuming sorted array
    int n = sizeof(arr) / sizeof(arr[0]);
    int x, low = 0, high = n - 1, mid;
    printf("Enter number to search: ");
    scanf("%d", &x);
    while (low <= high) {
        mid = (low + high) / 2;
        if (arr[mid] == x) {
            printf("Found at index %d\n", mid);
            return 0;
        }
        else if (arr[mid] < x)
            low = mid + 1;
        else
            high = mid - 1;
    }
    printf("Not found\n");
    return 0;
}
```

Screenshot of the Binary Search Code:



The image shows a C program in a code editor and its execution output in a terminal window. The code implements a binary search algorithm on a sorted array.

```
1  #include <stdio.h>
2  int main() {
3      int arr[] = {00, 02, 05, 10, 12, 32, 72, 96};
4      int n = sizeof(arr) / sizeof(arr[0]);
5      int x, low = 0, high = n - 1, mid;
6      printf("Enter number to search: ");
7      scanf("%d", &x);
8      while (low <= high) {
9          mid = (low + high) / 2;
10         if (arr[mid] == x) {
11             printf("Found at index %d\n", mid);
12             return 0;
13         }
14         else if (arr[mid] < x)
15             low = mid + 1;
16         else
17             high = mid - 1;
18     }
19     printf("Not found\n");
20     return 0;
21 }
22
```

The terminal window shows the execution of the program. It prompts the user to enter a number to search for, and the user enters 10. The program finds the number at index 3 and displays the message "Found at index 3". It also shows the process returned 0 (0x0) and the execution time was 20.607 s. The prompt "Press ENTER to continue." is visible at the bottom of the terminal window.

Insertion Sort

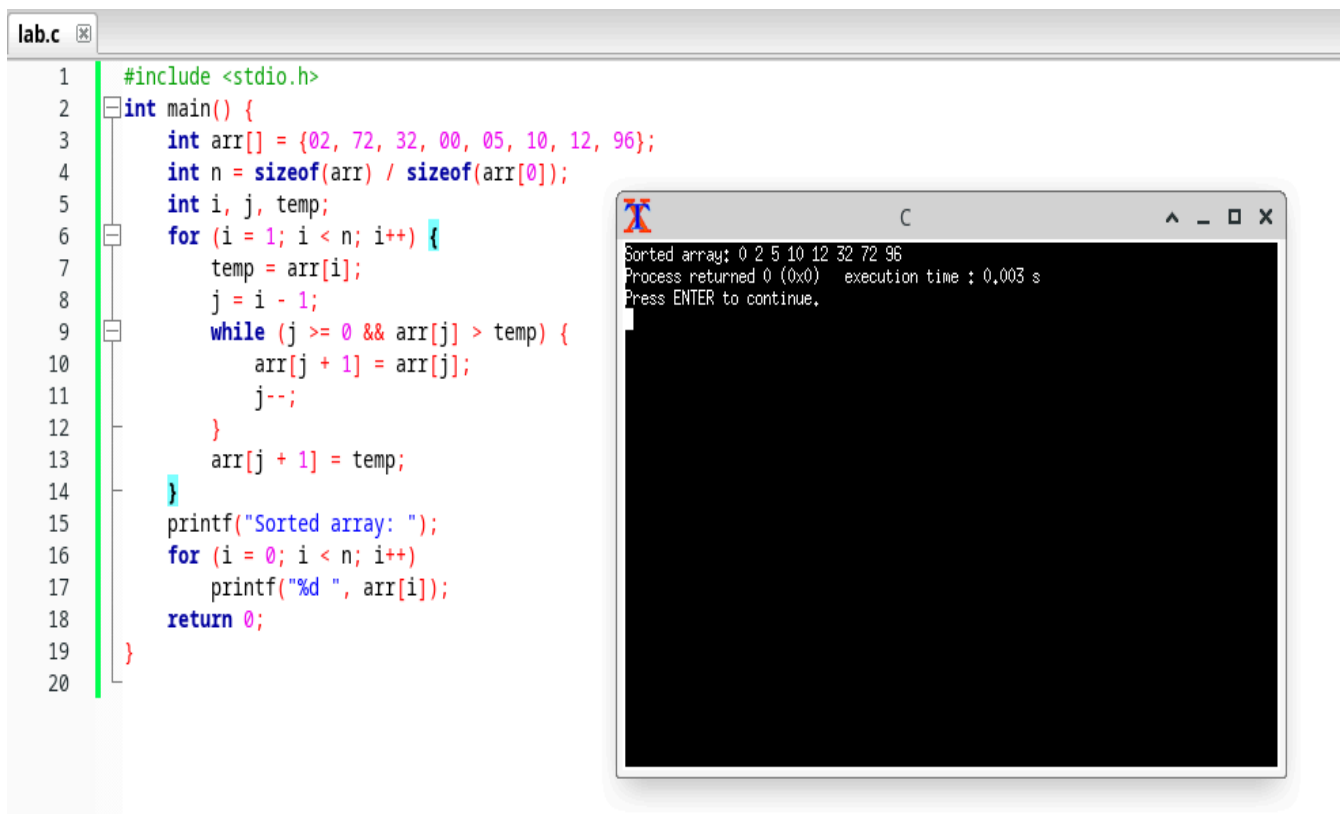
The code uses nested loops to shift elements and insert the current element in its correct position. It sorts the array correctly from the example runs.

Source Code :

```
#include <stdio.h>

int main() {
    int arr[] = {02, 72, 32, 00, 05, 10, 12, 96};
    int n = sizeof(arr) / sizeof(arr[0]);
    int i, j, temp;
    for (i = 1; i < n; i++) {
        temp = arr[i];
        j = i - 1;
        while (j >= 0 && arr[j] > temp) {
            arr[j + 1] = arr[j];
            j--;
        }
        arr[j + 1] = temp;
    }
    printf("Sorted array: ");
    for (i = 0; i < n; i++)
        printf("%d ", arr[i]);
    return 0;
}
```

Screenshot of the Insertion Sort Code:



```
1  #include <stdio.h>
2  int main() {
3      int arr[] = {02, 72, 32, 00, 05, 10, 12, 96};
4      int n = sizeof(arr) / sizeof(arr[0]);
5      int i, j, temp;
6      for (i = 1; i < n; i++) {
7          temp = arr[i];
8          j = i - 1;
9          while (j >= 0 && arr[j] > temp) {
10             arr[j + 1] = arr[j];
11             j--;
12         }
13         arr[j + 1] = temp;
14     }
15     printf("Sorted array: ");
16     for (i = 0; i < n; i++)
17         printf("%d ", arr[i]);
18     return 0;
19 }
20
```

Sorted array: 0 2 5 10 12 32 72 96
Process returned 0 (0x0) execution time : 0.003 s
Press ENTER to continue.

Bubble Sort

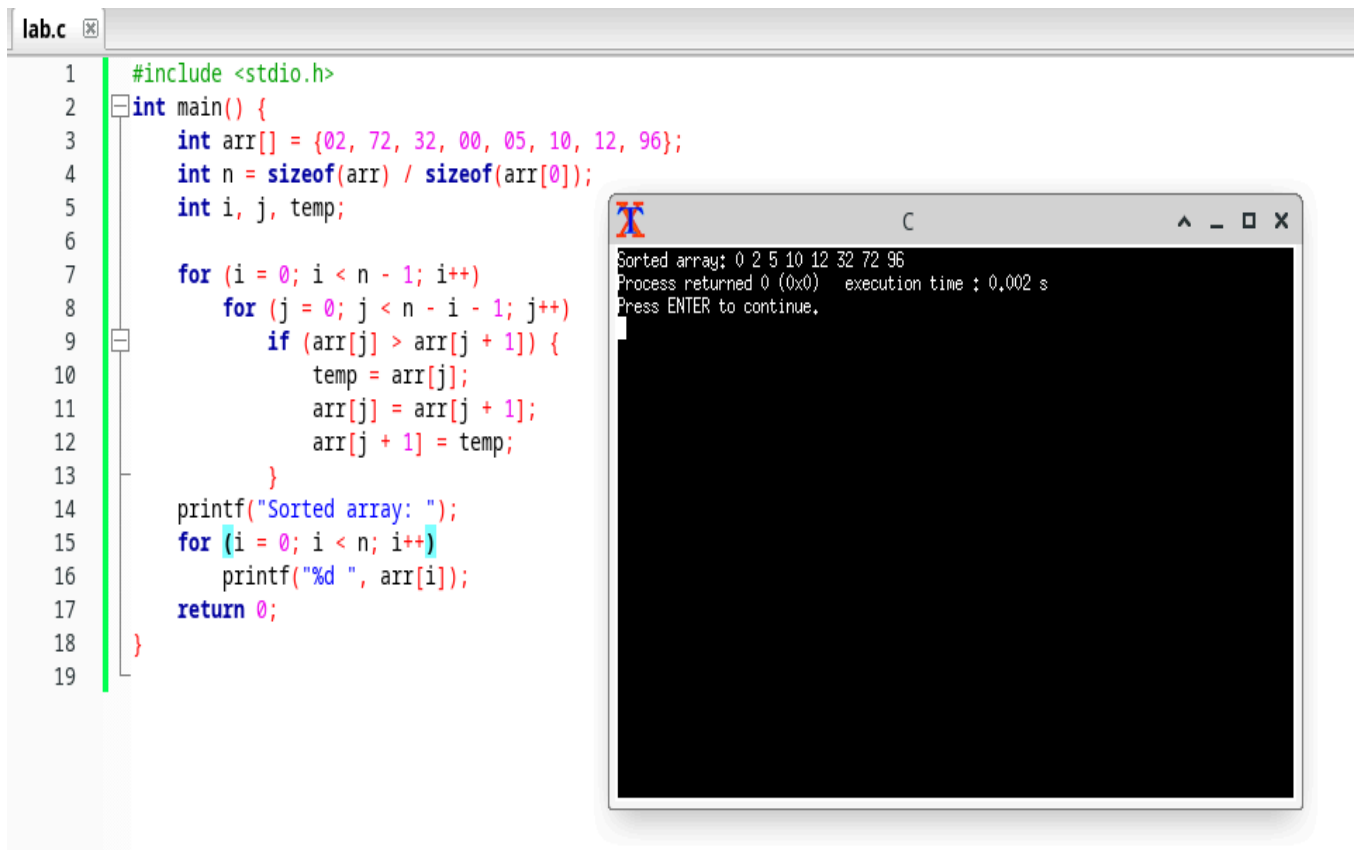
The code uses nested loops to compare adjacent elements and swap them if they are in the wrong order. It successfully sorts the array as expected.

Source Code :

```
#include <stdio.h>

int main() {
    int arr[] = {02, 72, 32, 00, 05, 10, 12, 96};
    int n = sizeof(arr) / sizeof(arr[0]);
    int i, j, temp;
    for (i = 0; i < n - 1; i++)
        for (j = 0; j < n - i - 1; j++)
            if (arr[j] > arr[j + 1]) {
                temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
    printf("Sorted array: ");
    for (i = 0; i < n; i++)
        printf("%d ", arr[i]);
    return 0;
}
```


Screenshot of the Bubble Sort Code:



The image shows a code editor window titled 'lab.c' and a terminal window titled 'C'. The code in the editor implements a bubble sort algorithm on an array of integers. The terminal shows the output of the program, which is a sorted array and the execution time.

```
1  #include <stdio.h>
2  int main() {
3      int arr[] = {02, 72, 32, 00, 05, 10, 12, 96};
4      int n = sizeof(arr) / sizeof(arr[0]);
5      int i, j, temp;
6
7      for (i = 0; i < n - 1; i++)
8          for (j = 0; j < n - i - 1; j++)
9              if (arr[j] > arr[j + 1]) {
10                 temp = arr[j];
11                 arr[j] = arr[j + 1];
12                 arr[j + 1] = temp;
13             }
14     printf("Sorted array: ");
15     for (i = 0; i < n; i++)
16         printf("%d ", arr[i]);
17     return 0;
18 }
19
```

Sorted array: 0 2 5 10 12 32 72 96
Process returned 0 (0x0) execution time : 0.002 s
Press ENTER to continue.

Merge Sort

The recursive divide-and-conquer approach with a merge function is correctly implemented. It sorts the array by splitting it into halves, sorting them, and merging them back.

Source Code :

```
#include <stdio.h>

void merge(int arr[], int l, int m, int r) {
    int i, j, k;
    int n1 = m - l + 1;
    int n2 = r - m;
    int L[n1], R[n2];
    for (i = 0; i < n1; i++)
        L[i] = arr[l + i];
    for (j = 0; j < n2; j++)
        R[j] = arr[m + 1 + j];
    i = 0; j = 0; k = l;
    while (i < n1 && j < n2) {
        if (L[i] <= R[j])
            arr[k++] = L[i++];
        else
            arr[k++] = R[j++];
    }
}
```

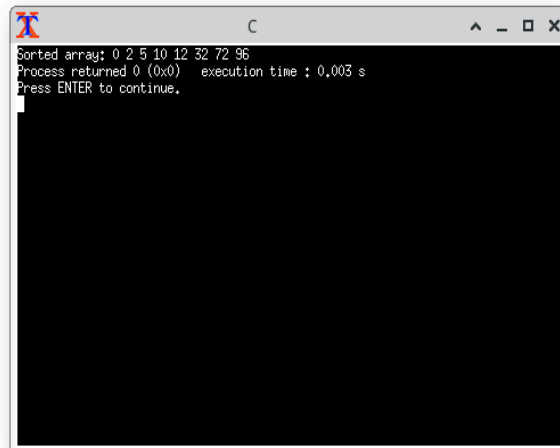
```

while (i < n1)
    arr[k++] = L[i++];
while (j < n2)
    arr[k++] = R[j++];
}
void mergeSort(int arr[], int l, int r) {
    if (l < r) {
        int m = l + (r - l) / 2;
        mergeSort(arr, l, m);
        mergeSort(arr, m + 1, r);
        merge(arr, l, m, r);
    }
}
int main() {
    int arr[] = {02, 72, 32, 00, 05, 10, 12, 96};
    int n = sizeof(arr) / sizeof(arr[0]);
    mergeSort(arr, 0, n - 1);
    printf("Sorted array: ");
    for (int i = 0; i < n; i++)
        printf("%d ", arr[i]);
    return 0;
}

```

Screenshot of the Merge Sort Code:

```
*lab.c
1  #include <stdio.h>
2  void merge(int arr[], int l, int m, int r) {
3      int i, j, k;
4      int n1 = m - l + 1;
5      int n2 = r - m;
6      int L[n1], R[n2];
7      for (i = 0; i < n1; i++)
8          L[i] = arr[l + i];
9      for (j = 0; j < n2; j++)
10         R[j] = arr[m + 1 + j];
11     i = 0; j = 0; k = l;
12     while (i < n1 && j < n2) {
13         if (L[i] <= R[j])
14             arr[k++] = L[i++];
15         else
16             arr[k++] = R[j++];
17     }
18     while (i < n1)
19         arr[k++] = L[i++];
20     while (j < n2)
21         arr[k++] = R[j++];
22 }
23 void mergeSort(int arr[], int l, int r) {
24     if (l < r) {
25         int m = l + (r - l) / 2;
26         mergeSort(arr, l, m);
27         mergeSort(arr, m + 1, r);
28         merge(arr, l, m, r);
29     }
30 }
31 int main() {
32     int arr[] = {02, 72, 32, 00, 05, 10, 12, 96};
33     int n = sizeof(arr) / sizeof(arr[0]);
```



```
32     int main() {
33         int arr[] = {02, 72, 32, 00, 05, 10, 12, 96};
34         int n = sizeof(arr) / sizeof(arr[0]);
35         mergeSort(arr, 0, n - 1);
36         printf("Sorted array: ");
37         for (int i = 0; i < n; i++)
38             printf("%d ", arr[i]);
39         return 0;
40     }
41 }
```