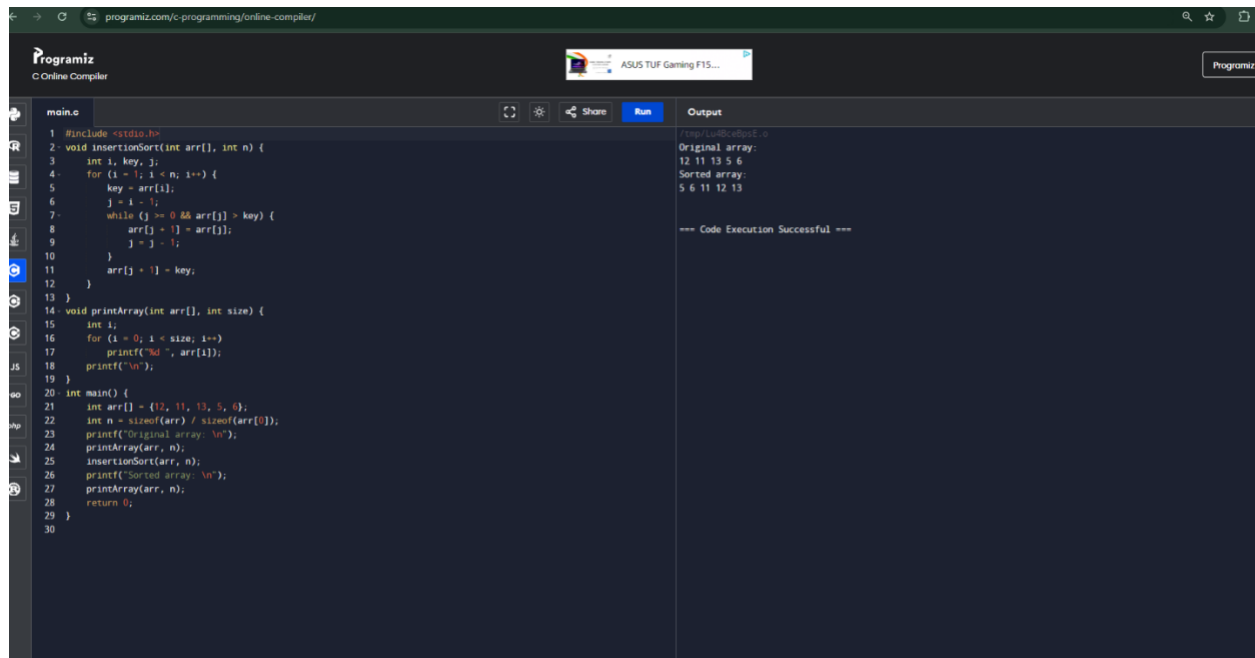


1.c code for insertion sorting



The screenshot shows the Programiz C Online Compiler interface. The code editor on the left contains the following C code for insertion sort:

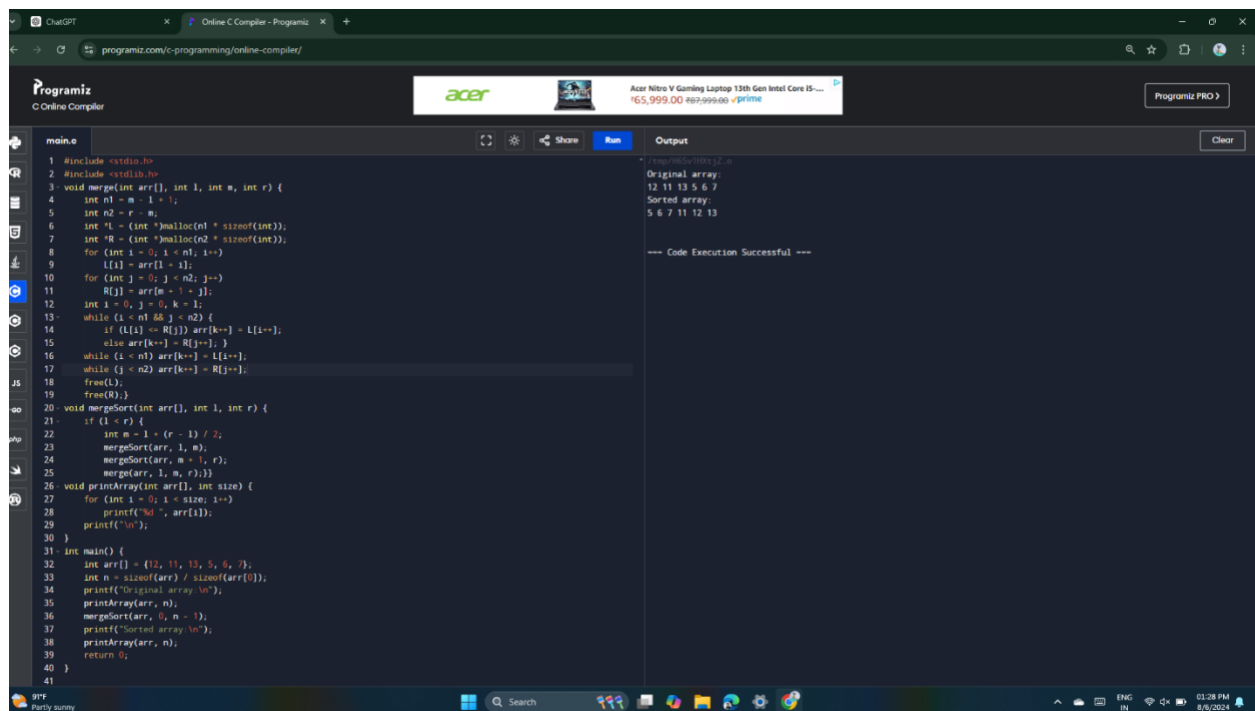
```
1 #include <stdio.h>
2 void insertionSort(int arr[], int n) {
3     int i, key, j;
4     for (i = 1; i < n; i++) {
5         key = arr[i];
6         j = i - 1;
7         while (j >= 0 && arr[j] > key) {
8             arr[j + 1] = arr[j];
9             j = j - 1;
10        }
11        arr[j + 1] = key;
12    }
13 }
14 void printArray(int arr[], int size) {
15     int i;
16     for (i = 0; i < size; i++)
17         printf("%d ", arr[i]);
18     printf("\n");
19 }
20 int main() {
21     int arr[] = {12, 11, 13, 5, 6};
22     int n = sizeof(arr) / sizeof(arr[0]);
23     printf("Original array: \n");
24     printArray(arr, n);
25     insertionSort(arr, n);
26     printf("Sorted array: \n");
27     printArray(arr, n);
28     return 0;
29 }
30
```

The output panel on the right shows the following results:

```
Original array:
12 11 13 5 6
Sorted array:
5 6 11 12 13

--- Code Execution Successful ---
```

2. c code for merge sorting



The screenshot shows the Programiz C Online Compiler interface. The code editor on the left contains the following C code for merge sort:

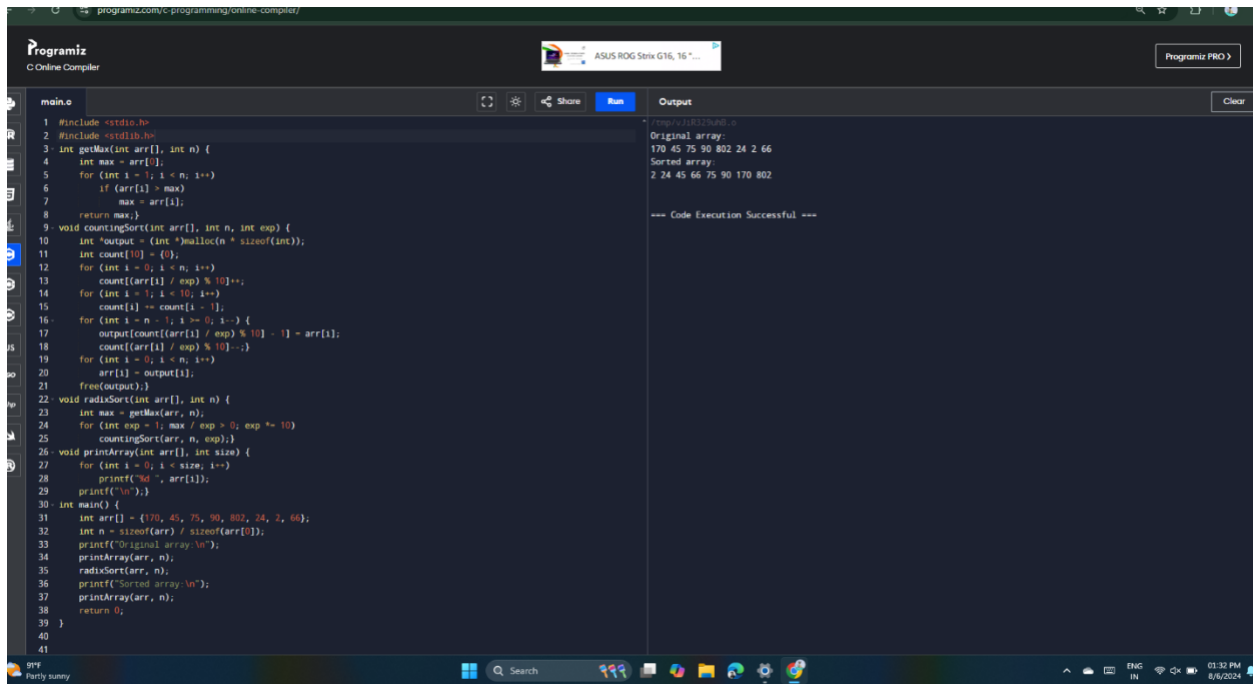
```
1 #include <stdio.h>
2 #include <stdlib.h>
3 void merge(int arr[], int l, int m, int r) {
4     int n1 = m - l + 1;
5     int n2 = r - m;
6     int *L = (int *)malloc(n1 * sizeof(int));
7     int *R = (int *)malloc(n2 * sizeof(int));
8     for (int i = 0; i < n1; i++)
9         L[i] = arr[l + i];
10    for (int j = 0; j < n2; j++)
11        R[j] = arr[m + 1 + j];
12    int i = 0, j = 0, k = l;
13    while (i < n1 && j < n2) {
14        if (L[i] <= R[j]) arr[k++] = L[i++];
15        else arr[k++] = R[j++];
16    }
17    while (i < n1) arr[k++] = L[i++];
18    while (j < n2) arr[k++] = R[j++];
19    free(L);
20    free(R);
21 }
22 void mergeSort(int arr[], int l, int r) {
23     if (l < r) {
24         int m = l + (r - l) / 2;
25         mergeSort(arr, l, m);
26         mergeSort(arr, m + 1, r);
27         merge(arr, l, m, r);
28     }
29 }
30 void printArray(int arr[], int size) {
31     for (int i = 0; i < size; i++)
32         printf("%d ", arr[i]);
33     printf("\n");
34 }
35 int main() {
36     int arr[] = {12, 11, 13, 5, 6, 7};
37     int n = sizeof(arr) / sizeof(arr[0]);
38     printf("Original array: \n");
39     printArray(arr, n);
40     mergeSort(arr, 0, n - 1);
41     printf("Sorted array: \n");
42     printArray(arr, n);
43     return 0;
44 }
45
```

The output panel on the right shows the following results:

```
Original array:
12 11 13 5 6 7
Sorted array:
5 6 7 11 12 13

--- Code Execution Successful ---
```

3. c code for Radix sorting



The screenshot displays the Programiz C Online Compiler interface. The left pane shows the source code for a C program that implements Radix Sort. The right pane shows the output of the program, which prints the original array, the sorted array, and a success message.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 int getMax(int arr[], int n) {
4     int max = arr[0];
5     for (int i = 1; i < n; i++)
6         if (arr[i] > max)
7             max = arr[i];
8     return max;
9 void countingSort(int arr[], int n, int exp) {
10    int *output = (int *)malloc(n * sizeof(int));
11    int count[10] = {0};
12    for (int i = 0; i < n; i++)
13        count[(arr[i] / exp) % 10]++;
14    for (int i = 1; i < 10; i++)
15        count[i] += count[i - 1];
16    for (int i = n - 1; i >= 0; i--) {
17        output[count[(arr[i] / exp) % 10] - 1] = arr[i];
18        count[(arr[i] / exp) % 10]--;
19    }
20    for (int i = 0; i < n; i++)
21        arr[i] = output[i];
22    free(output);
23 void radixSort(int arr[], int n) {
24    int max = getMax(arr, n);
25    for (int exp = 1; max / exp > 0; exp *= 10)
26        countingSort(arr, n, exp);
27 void printArray(int arr[], int size) {
28    for (int i = 0; i < size; i++)
29        printf("%d ", arr[i]);
30    printf("\n");
31 int main() {
32    int arr[] = {170, 45, 75, 90, 802, 24, 2, 66};
33    int n = sizeof(arr) / sizeof(arr[0]);
34    printf("Original array\n");
35    printArray(arr, n);
36    radixSort(arr, n);
37    printf("Sorted array\n");
38    printArray(arr, n);
39    return 0;
40 }
41
```

Output:

```
Original array:
170 45 75 90 802 24 2 66
Sorted array:
2 24 45 66 75 90 170 802
=== Code Execution Successful ===
```