

**MINISTRY OF EDUCATION, CULTURE AND RESEARCH**

**OF THE REPUBLIC OF MOLDOVA**

**Technical University of Moldova**

**Faculty of Computers, Informatics and Microelectronics**

**Department of Software and Automation Engineering**

**Iamandii Ion student**

**Group: FAF-233**

**Report**

**Laboratory Work No.1**

***of the "Data Structures and Algorithms" course***

Checked:

Burlacu Natalia, PhD, Associate Professor

Department of Software and Automation Engineering,

FCIM Faculty, UTM

**Chisinau – 2024**

**Task:**

1. Solve the following problems in C, writing your own functions according to the given statements. Write the solution of the problem by procedural approach in two versions:

A. with the use of the method of transmitting the parametric functions by value;

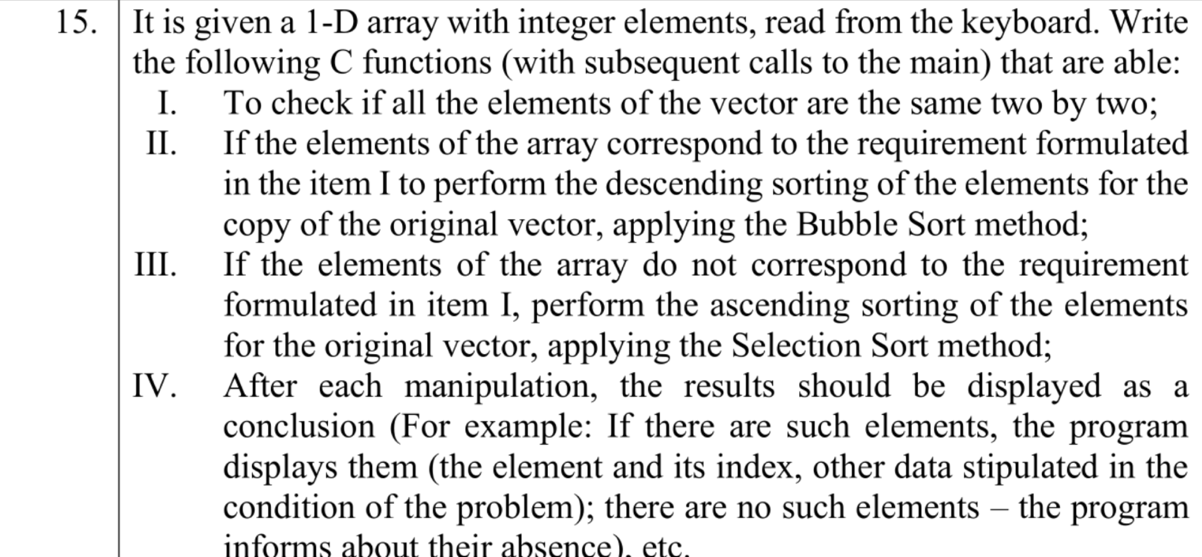
B. with the use of the method of passing parameters of functions by address/pointers (the formal parameter will be a pointer to the value of the corresponding object).

C. To draw the block diagram corresponding to the solved problem.

2. Modify the content of your problems emerging from the possibilities that are missing, but which can be brought as added value in the condition of the existing problem. Formulate and present in writing the modified condition; to solve in C your problem in the modified version, using functions developed by you

Because of the fact that in every problem in version 1, you should use two specified sorting methods, in version 2, of the problem proposed (modified) by you, you should use the sorting methods as Merge Sort & Insert Sort.

**Condition of the problem:**



**Fig 1.1** – Condition of problem

**1. The code of the program, with relevant comments in it, and the Block diagram;**

**Code:**

**-------------** **The version with passing function parameters by value -----------**

#include <stdio.h>

#include <stdbool.h>

// nr 15

// condition 1: check if all the elements of the vector are the same 2 by 2

// if condition 1 is true, then perform the descending sorting of the elements for the copy of the original vector, applying the Bubble Sort method

// if condition 1 is false, then perform the ascending sorting of the elements for the original vector, applying the Selection Sort method

// Bubble Sort

void bubbleSort(int arr[], int n);

// Selection Sort

void selectionSort(int arr[], int n);

int main() {

int n; // length of array

printf("n = ");

scanf("%d", &n);

int arr[n];

// input array

printf("Array:\n");

for (int i = 0; i < n; i++) {

printf("Element %d = ", i + 1);

scanf("%d", &arr[i]);

}

bool checkIfEqual2by2 = true;

for (int i = 0; i < n - 1; i += 2) {

if (arr[i] != arr[i + 1]) {

checkIfEqual2by2 = false;

break;

}

}

if (checkIfEqual2by2) {

printf("All the elements of the vector are the same 2 by 2");

} else {

printf("All the elements of the vector are NOT the same 2 by 2");

}

if (n == 0) {

printf("Empty array");

} else if (checkIfEqual2by2 && n % 2 == 0) { // array must meet condition 1, and the length of array must be even for it to work

// make a copy of the original vector

int copyArr[n];

for (int i = 0; i < n; i++) {

copyArr[i] = arr[i];

}

// sort descending with bubble sort

bubbleSort(copyArr, n);

int descendingArr[n];

for (int i = 0; i < n; i++) {

descendingArr[i] = copyArr[n-1-i];

}

// display array

printf("\nDescending sorted array with Bubble Sort:\n[");

for (int i = 0; i < n; ++i) {

if (i == n - 1) {

printf("%d]\n", descendingArr[i]);

} else {

printf("%d, ", descendingArr[i]);

}

}

} else { // condition 1 is not met or length of array is odd sort ascending with selection sort

selectionSort(arr, n);

// display array

printf("\nAscending sorted array with Selection Sort:\n[");

for (int i = 0; i < n; ++i) {

if (i == n - 1) {

printf("%d]\n", arr[i]);

} else {

printf("%d, ", arr[i]);

}

}

}

return 0;

}

// BUBBLE SORT

void bubbleSort(int arr[], int n) {

for (int i = 0; i < n - 1; i++) {

for (int j = 0; j < n - 1 - i; j++) {

if (arr[j] > arr[j + 1]) {

// swap elements

int temp = arr[j];

arr[j] = arr[j + 1];

arr[j + 1] = temp;

}

}

}

}

// SELECTION SORT

void selectionSort(int arr[], int n) {

for (int i = 0; i < n - 1; i++) {

int min = i;

for (int j = i + 1; j < n; j++) {

if (arr[min] > arr[j])

min = j;

}

// swap elements

int temp = arr[min];

arr[min] = arr[i];

arr[i] = temp;

}

}

**--------------** **The version with passing function parameters by pointers ---------------**

#include <stdio.h>

#include <stdbool.h>

// Bubble Sort

void bubbleSort(int \*arr, int n);

// Selection Sort

void selectionSort(int \*arr, int n);

int main() {

int n; // length of array

printf("n = ");

scanf("%d", &n);

int arr[n];

// input array

printf("Array:\n");

for (int i = 0; i < n; i++) {

printf("Element %d = ", i + 1);

scanf("%d", &arr[i]);

}

bool checkIfEqual2by2 = true;

for (int i = 0; i < n - 1; i += 2) {

if (arr[i] != arr[i + 1]) {

checkIfEqual2by2 = false;

break;

}

}

if (checkIfEqual2by2) {

printf("All the elements of the vector are the same 2 by 2\n");

} else {

printf("All the elements of the vector are NOT the same 2 by 2\n");

}

if (n == 0) {

printf("Empty array");

} else if (checkIfEqual2by2 && n % 2 == 0) { // array must meet condition 1, and the length of array must be even for it to work

int copyArr[n];

int \*ptrCopyArr = copyArr; // points to address of copyArr

for (int i = 0; i < n; i++) {

\*ptrCopyArr++ = arr[i];

}

// sort descending with bubble sort

bubbleSort(copyArr, n);

int descendingArr[n];

int \*ptrDescendingArr = descendingArr;

for (int i = 0; i < n; i++) {

\*ptrDescendingArr++ = copyArr[n - 1 - i];

}

// display array

printf("Descending sorted array with Bubble Sort:\n[");

for (int i = 0; i < n; ++i) {

if (i == n - 1) {

printf("%d]\n", descendingArr[i]);

} else {

printf("%d, ", descendingArr[i]);

}

}

} else { // condition 1 is not met or length of array is odd

// sort ascending with selection sort

selectionSort(arr, n);

// display array

printf("\nAscending sorted array with Selection Sort: [");

for (int i = 0; i < n; ++i) {

if (i == n - 1) {

printf("%d]\n", arr[i]);

} else {

printf("%d, ", arr[i]);

}

}

}

return 0;

}

// BUBBLE SORT

void bubbleSort(int \*arr, int n) {

for (int i = 0; i < n - 1; i++) {

for (int j = 0; j < n - 1 - i; j++) {

if (\*(arr + j) > \*(arr + j + 1)) {

// swap elements

int temp = \*(arr + j);

\*(arr + j) = \*(arr + j + 1);

\*(arr + j + 1) = temp;

}

}

}

}

// SELECTION SORT

void selectionSort(int \*arr, int n) {

for (int i = 0; i < n - 1; i++) {

int min = i;

for (int j = i + 1; j < n; j++) {

if (\*(arr + min) > \*(arr + j))

min = j;

}

// swap elements

int temp = \*(arr + min);

\*(arr + min) = \*(arr + i);

\*(arr + i) = temp;

}

}

**-----------------------------------Modified version--------------------------------------**

#include <stdio.h>

#include <stdbool.h>

// nr 15

// condition 1: check if all the elements of the vector are the same 2 by 2

// condition 2: check if the array length is odd, if it is odd then add one more element by selecting all elements that are divisible by 2 and by 3 at the same time, and calculating their sum

// if condition 1 is true, then perform the descending sorting of the elements for the copy of the original vector, applying the Merge Sort method

// if condition 1 is false, then perform the ascending sorting of the elements for the original vector, applying the Insertion Sort method

// Merge Sort

void mergeSort(int arr[], int l, int r);

// Insertion Sort

void insertionSort(int arr[], int n);

int main() {

int n; // length of array

printf("n = ");

scanf("%d", &n);

int arr[n];

// input array

printf("Array:\n");

for (int i = 0; i < n; i++) {

printf("Element %d = ", i + 1);

scanf("%d", &arr[i]);

}

bool checkIfEqual2by2 = true;

for (int i = 0; i < n - 1; i += 2) {

if (arr[i] != arr[i + 1]) {

checkIfEqual2by2 = false;

break;

}

}

if (checkIfEqual2by2) {

printf("All the elements of the vector are the same 2 by 2");

} else {

printf("All the elements of the vector are NOT the same 2 by 2");

}

if (n == 0) {

printf("Empty array");

} else if (n % 2 == 1) {

int newElement = 0;

for (int i = 0; i < n; i++) {

if (arr[i] % 2 == 0 && arr[i] % 3 == 0) {

newElement+=arr[i];

}

}

// make a new array and add the new element (the sum of all elements divisible by 2 and 3) at the end

int newArr[n+1];

for (int i = 0; i < n; i++) {

newArr[i] = arr[i];

}

newArr[n] = newElement;

printf("\nSum of all elements divisible by 2 and 3 was added to the array since its length is odd");

insertionSort(newArr, n+1);

// display array

printf("\nAscending sorted array with Insertion Sort:\n[");

for (int i = 0; i < n+1; i++) {

if (i == n) {

printf("%d]\n", newArr[i]);

} else {

printf("%d, ", newArr[i]);

}

}

} else if (checkIfEqual2by2) { // array must meet condition 1, and the length of array must be even for it to work

// make a copy of the original vector

int copyArr[n];

for (int i = 0; i < n; i++) {

copyArr[i] = arr[i];

}

// sort descending with bubble sort

mergeSort(copyArr, 0, n-1);

int descendingArr[n];

for (int i = 0; i < n; i++) {

descendingArr[i] = copyArr[n-1-i];

}

// display array

printf("\nDescending sorted array with Merge Sort:\n[");

for (int i = 0; i < n; ++i) {

if (i == n - 1) {

printf("%d]\n", descendingArr[i]);

} else {

printf("%d, ", descendingArr[i]);

}

}

} else { // condition 1 is not met or length of array is odd sort ascending with selection sort

insertionSort(arr, n);

// display array

printf("\nAscending sorted array with Insertion Sort:\n[");

for (int i = 0; i < n; ++i) {

if (i == n - 1) {

printf("%d]\n", arr[i]);

} else {

printf("%d, ", arr[i]);

}

}

}

return 0;

}

// MERGE SORT

void merge(int arr[], int l, int m, int r) {

int i, j, k;

int n1 = m - l + 1;

int n2 = r - m;

// temporary arrays left and right

int L[n1], R[n2];

// copy data to temp arrays

for (i = 0; i < n1; i++)

L[i] = arr[l + i];

for (j = 0; j < n2; j++)

R[j] = arr[m + 1 + j];

// merge the temp arrays back

i = 0;

j = 0;

k = l;

while (i < n1 && j < n2) {

if (L[i] <= R[j]) {

arr[k] = L[i];

i++;

}

else {

arr[k] = R[j];

j++;

}

k++;

}

// copy the remaining elements of L if there are any

while (i < n1) {

arr[k] = L[i];

i++;

k++;

}

// Copy the remaining elements of R[],

// if there are any

while (j < n2) {

arr[k] = R[j];

j++;

k++;

}

}

// l is for left index and r is right index of the sub-array of arr to be sorted

void mergeSort(int arr[], int l, int r) {

if (l < r) {

int m = l + (r - l) / 2;

// sort first and second halves

mergeSort(arr, l, m);

mergeSort(arr, m + 1, r);

merge(arr, l, m, r);

}

}

// INSERTION SORT

void insertionSort(int arr[], int n) {

int i, a, j; // i - the right element, j - the left element

for (i = 1; i < n; i++) { // iterate through array

a = arr[i];

j = i - 1;

// swap the right and left elements if they are not in order

while (j >= 0 && arr[j] > a) {

arr[j+1] = arr[j];

j = j - 1;

}

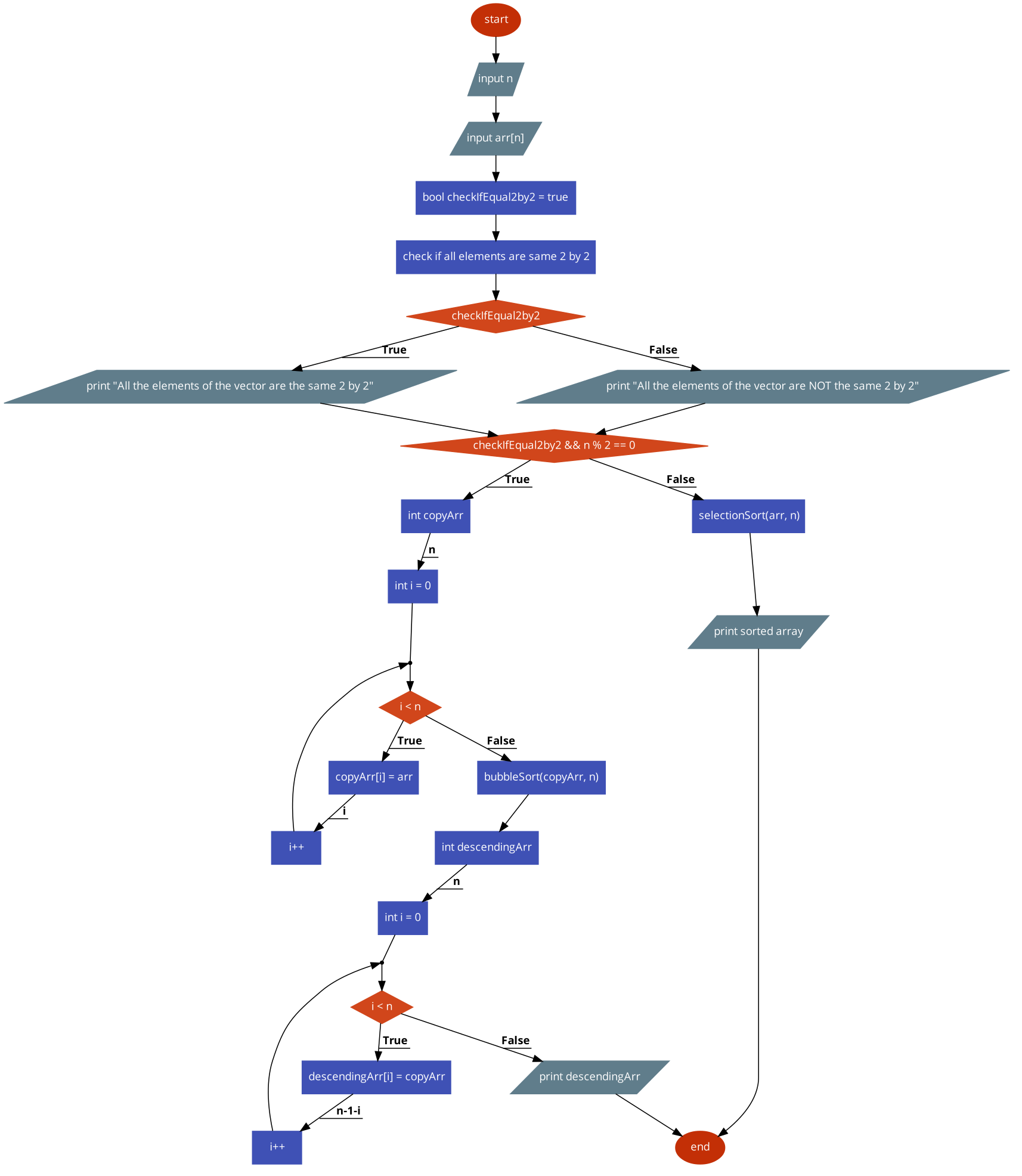
arr[j+1] = a;

}

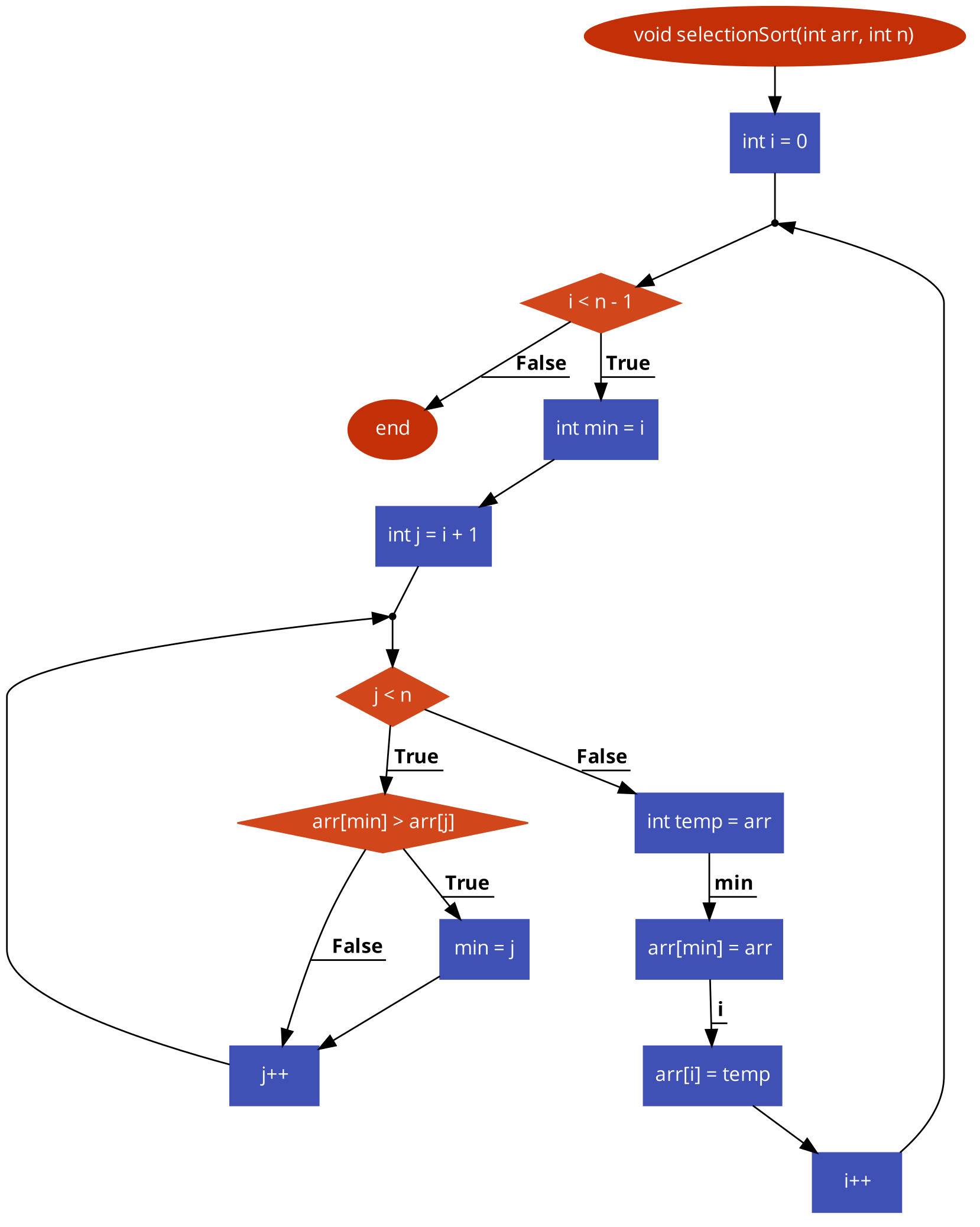
}

In this version I added an algorithm for checking if the array length is odd, and if it is odd, then add one more element by selecting all elements that are divisible by 2 and by 3 at the same time, and calculating their sum, then pushing this element to the end of the array

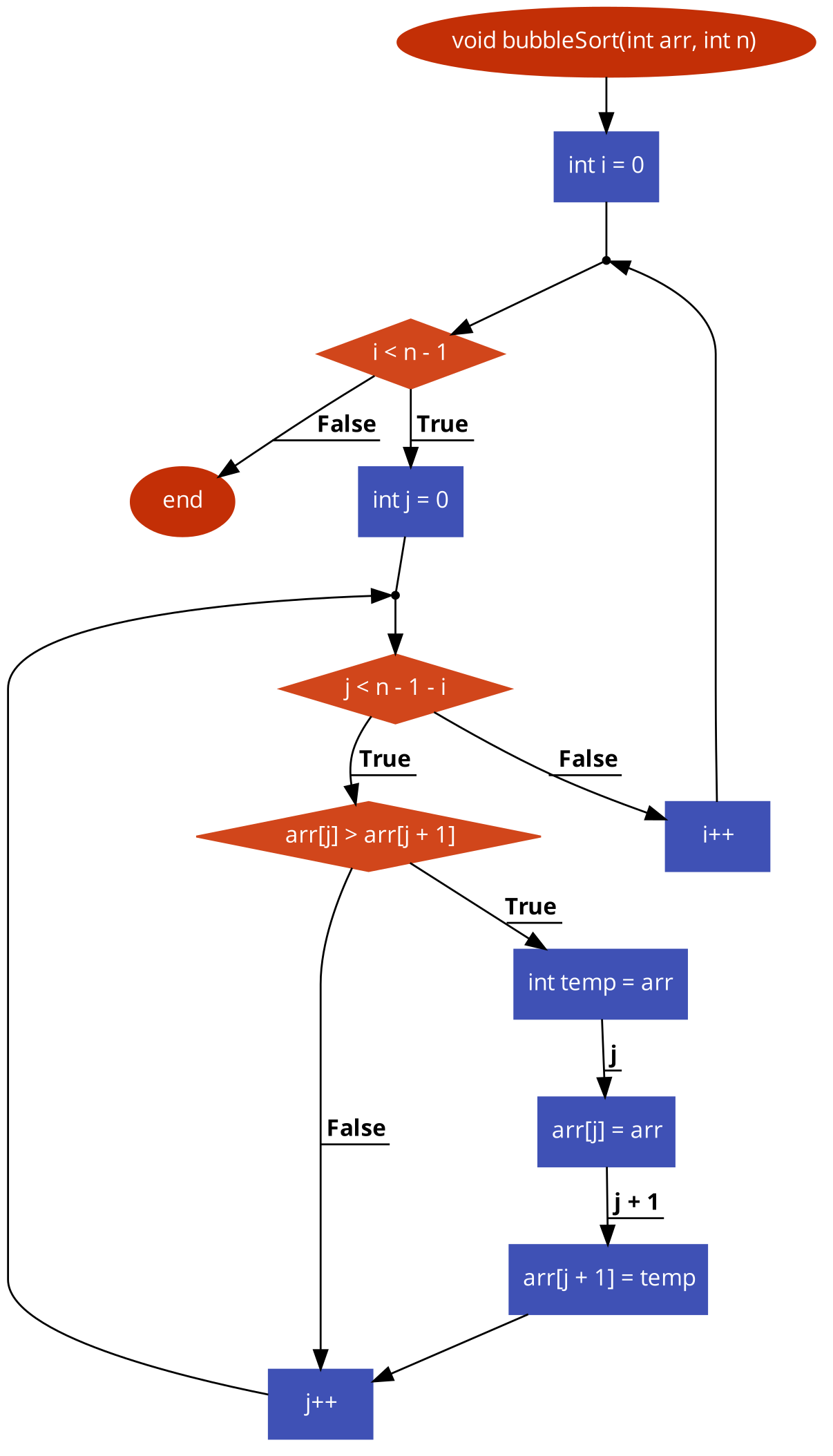
**Flowchart for first version:**



**Fig 2.1** – main function

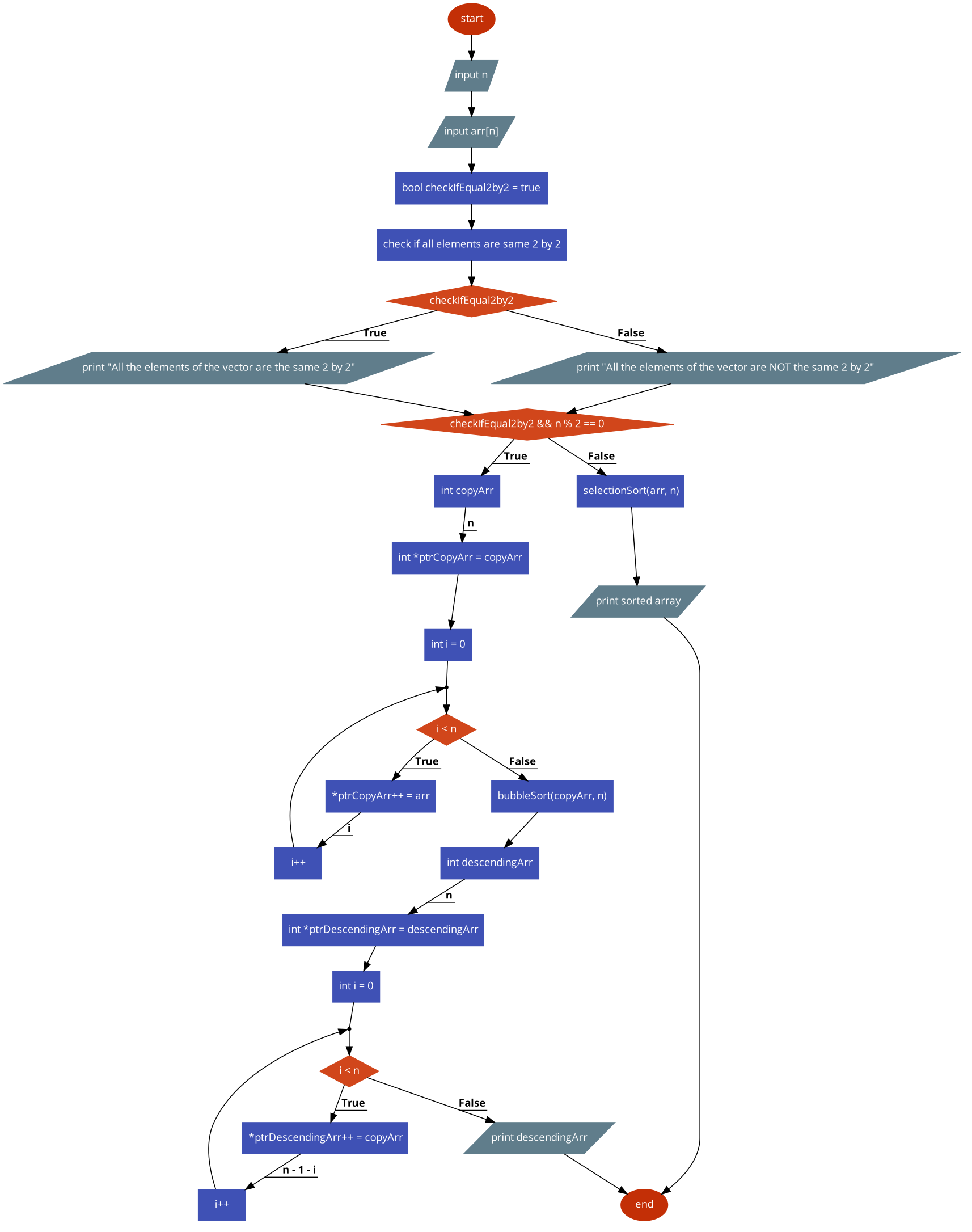


**Fig 2.2** –selection sort function

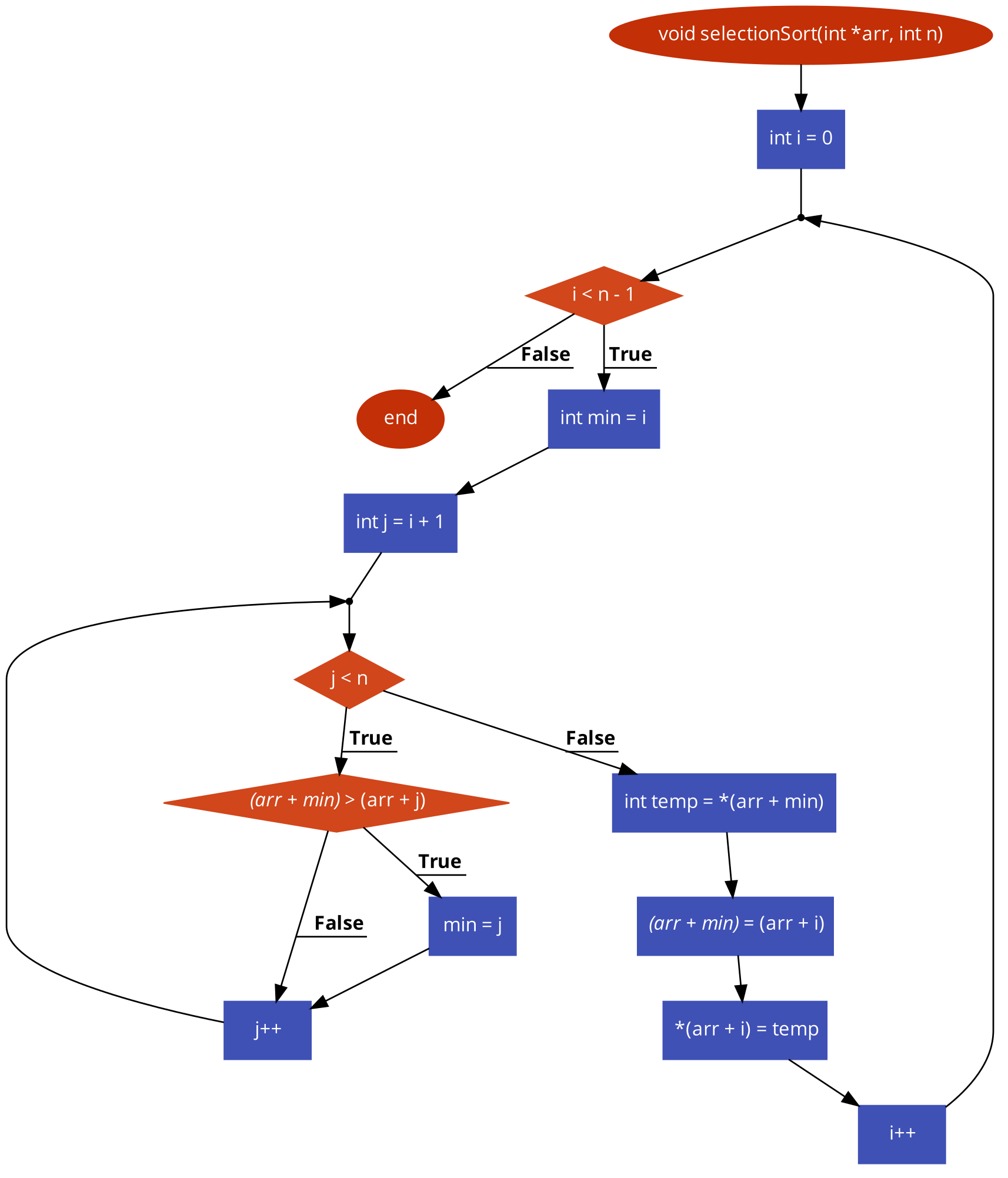


**Fig 2.3** – bubble sort function

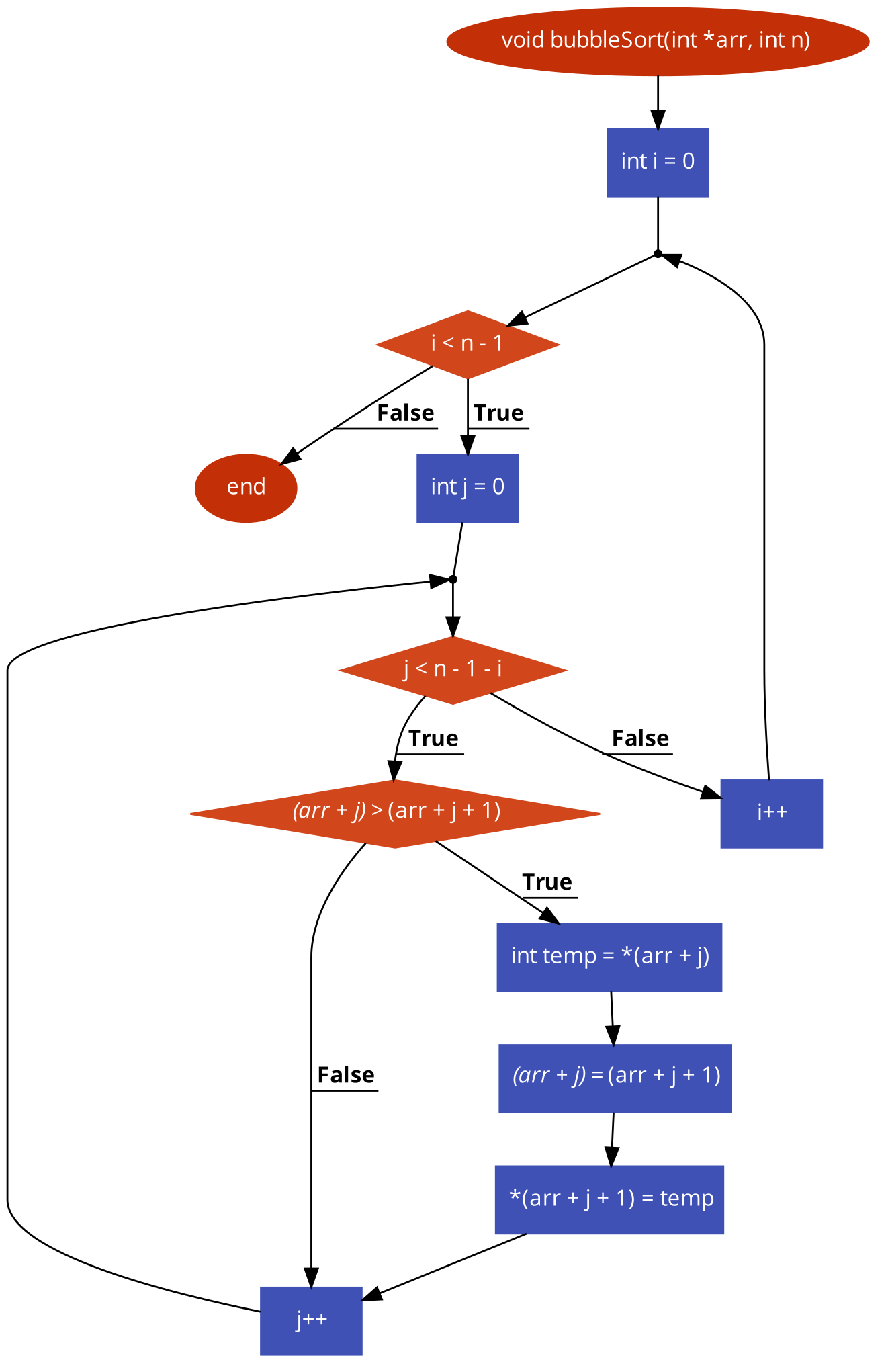
**Flowchart for second version:**



**Fig 2.4** – main function

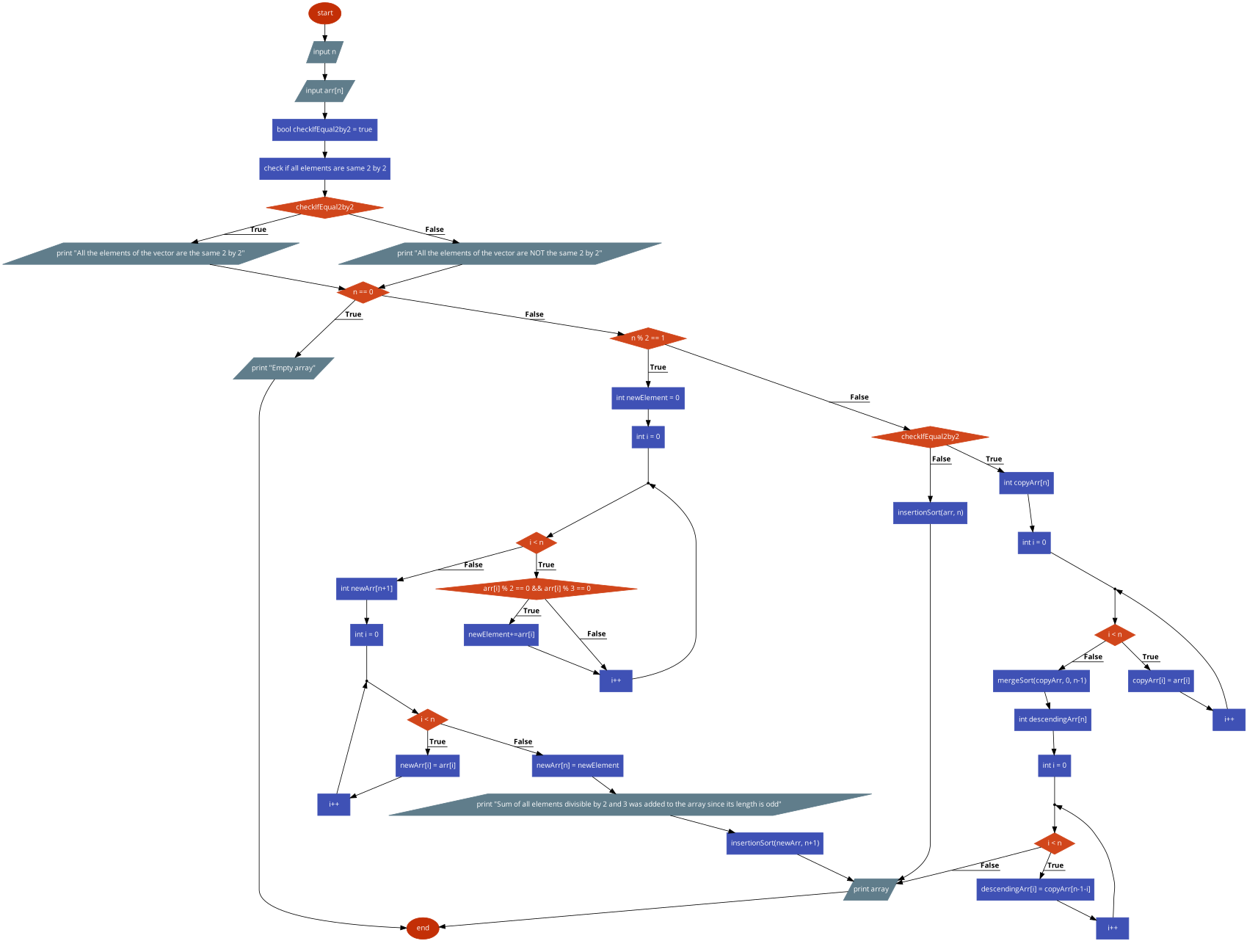


**Fig 2.5** – selection sort function

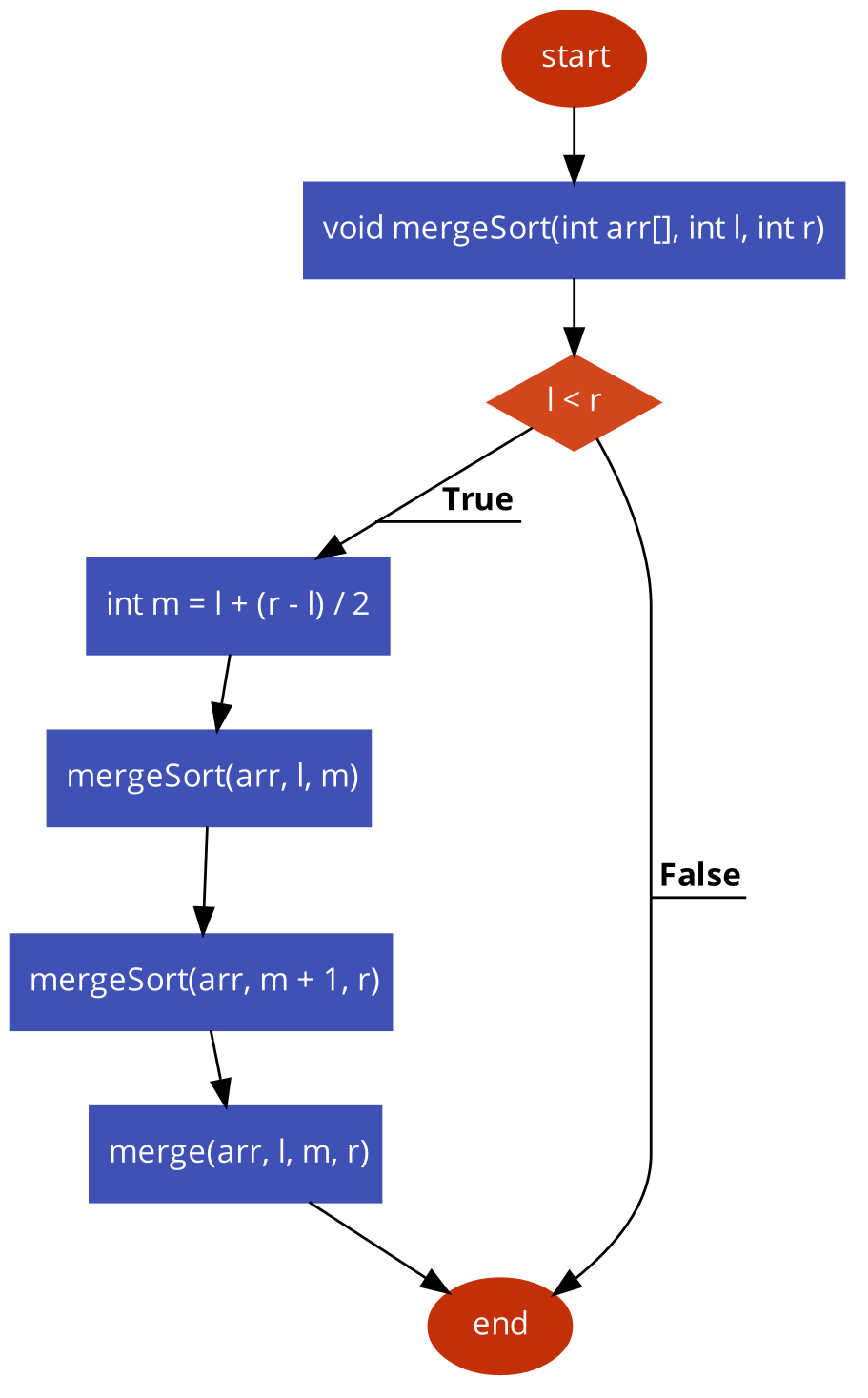


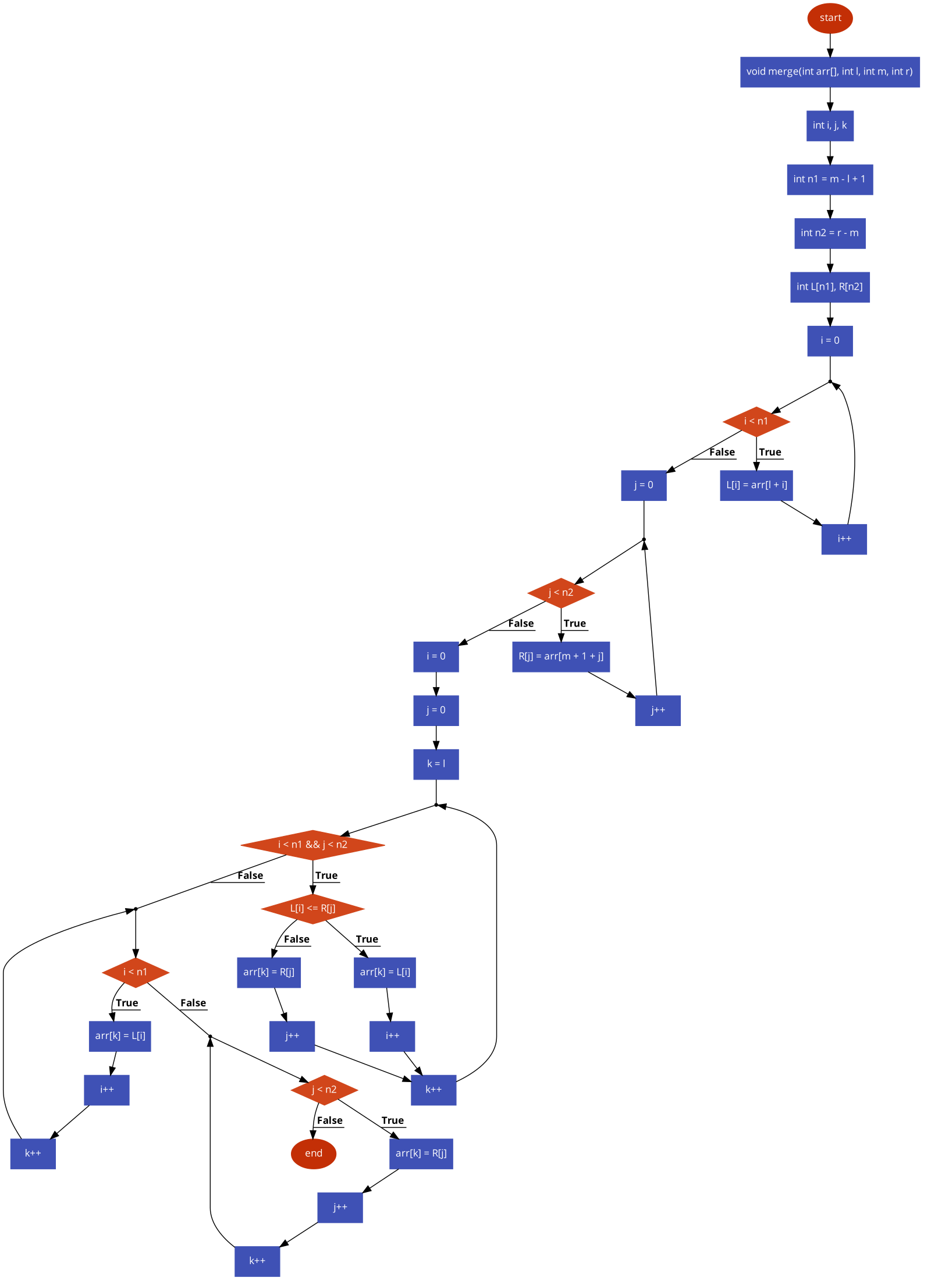
**Fig 2.6** – bubble sort function

**Flowchart for third version:**

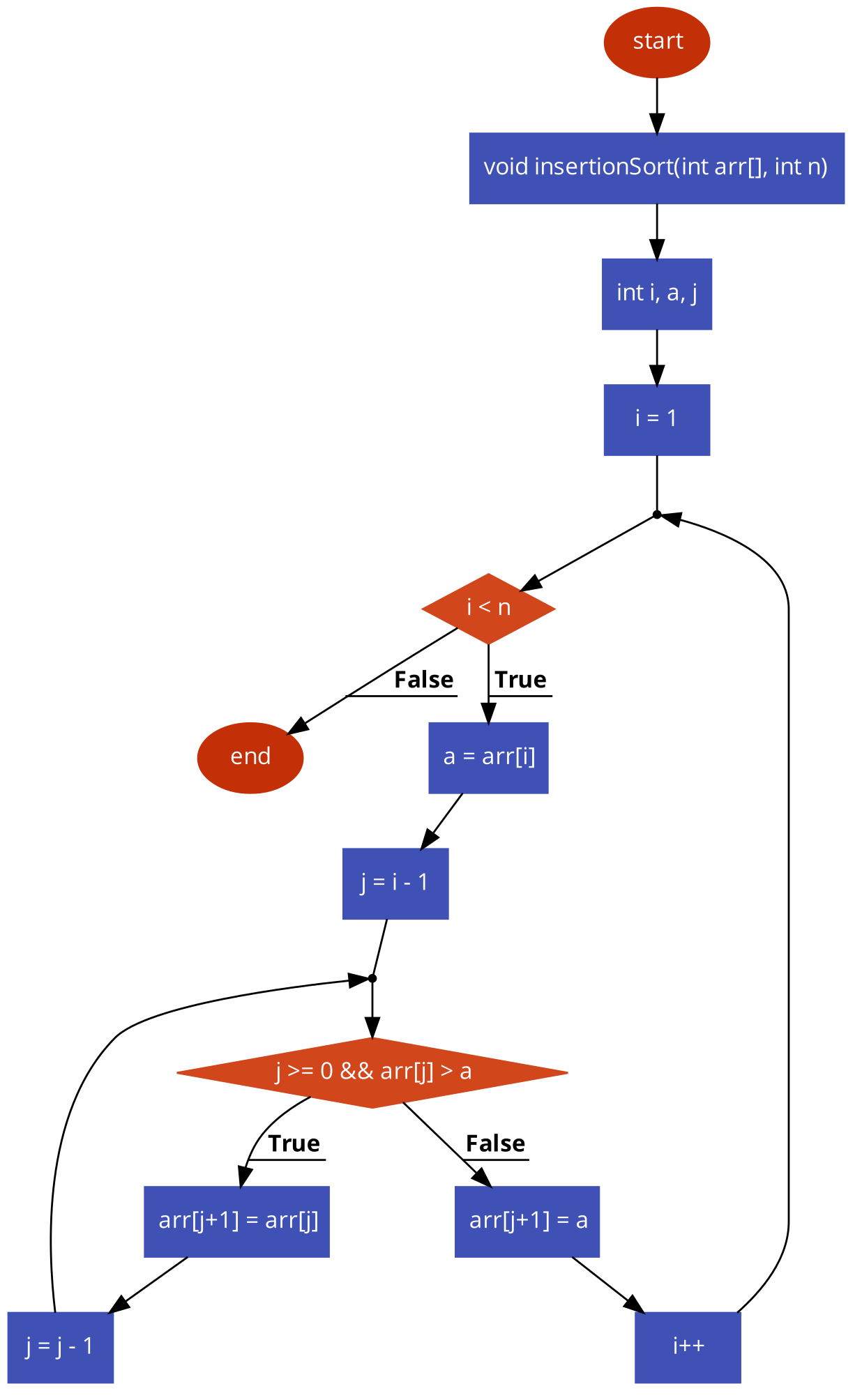


**Fig 2.7** – main function



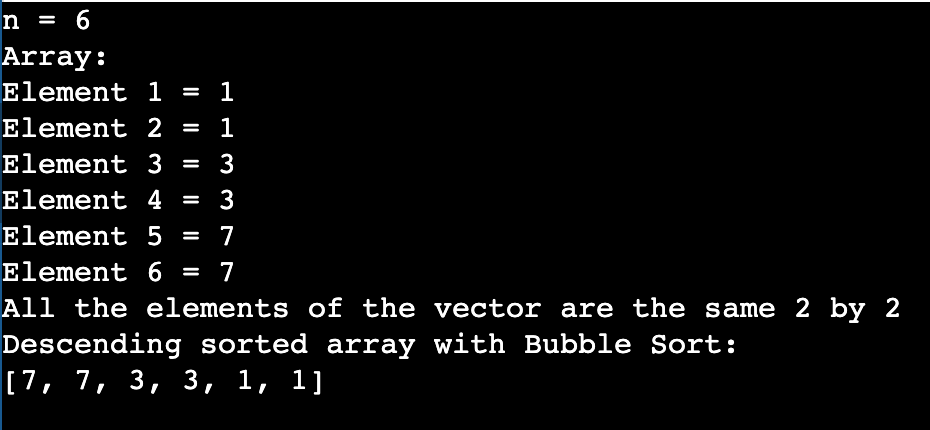
**Fig 2.8** – merge sort function

**Fig 2.9** – merge function

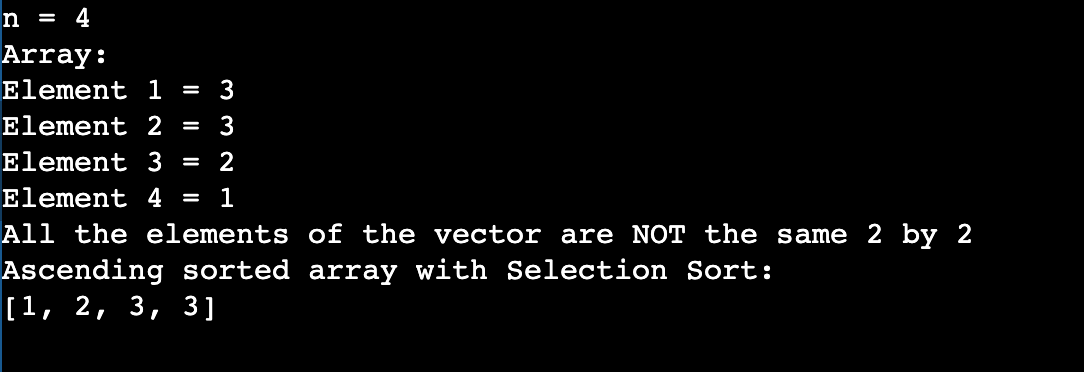


**Fig 2.10** – insertion sort function

**Output first version:**



**Fig 1 – all elements of the vector are same 2 by 2**



**Fig 2 – all elements of the vector are not same 2 by 2**

**Output modified version:**

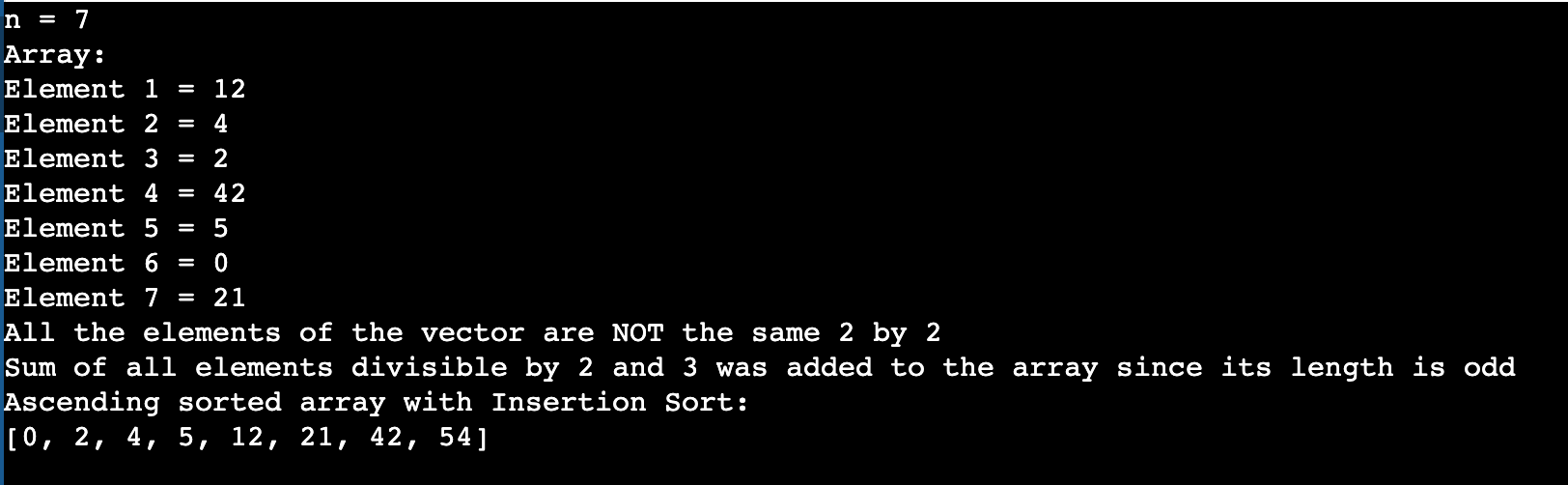


Fig 3 – element 54 was added at the end of the array, then sorted together with all other elements, since the only divisors of 2 and 3 from array are 12 and 42, then 54 = 12+42

**Conclusion:**

During the laboratory work, I applied the knowledge acquired in courses and seminars, employing various sorting methods to arrange a vector. Starting with simpler ones like Bubble Sort and Insertion Sort, I progressed to more complex ones like Merge Sort. I learned about finding the median of a vector and its calculation, and developed an algorithm to identify local extremes in a vector. Overall, considering the first work spans 27 pages, I consider this a good starting point.

As a result of the project, I gained skills in working with custom functions in the C language and structures. I incorporated pointers within structures and implemented multiple sorting methods for both digits and characters. I discovered the pros and cons of pointers, particularly in the realm of memory allocation. Moreover, I enhanced my abilities in creating flowcharts.