

**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.2**

***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;

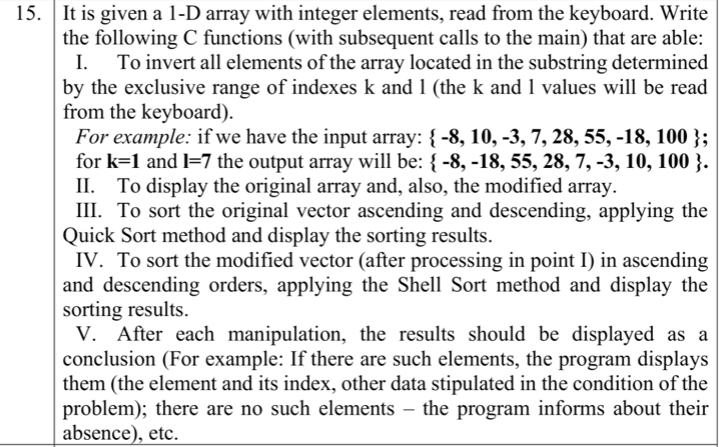
B. with the use of the method of passing parameters of functions by address/pointers (the formal parameter will be a pointer to the value of the corresponding object).

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 Counting Merge & Merge 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

// invert all elements of the array located in the substring determined by the exclusive range of indexes k and l (which will be read from the keyboard)

// display the original and the modified vectors

// sort the original vector ascending and descending, applying the quickSort method and display the sorting results

// sort the modified vector in ascending and descending orders, applying the shellSort method and display the sorting results

void invertWithinRange(int arr[], int k, int l) {

if ((l-k)%2==0) { // if there are an even number of elements to invert

int j = l-1;

for (int i = k; i <= ((l-k+1)/2); i++) {

int temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

j--;

}

} else { // if there are an odd number of elements to invert

int j = l-1;

for (int i = k; i < ((l-k+1)/2); i++) {

int temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

j--;

}

}

}

// swap elements

void swap(int \*a, int \*b) {

int t = \*a;

\*a = \*b;

\*b = t;

}

// find the partition position

int partition(int array[], int low, int high) {

// select the rightmost element as pivot

int pivot = array[high];

// pointer for greater element

int i = (low - 1);

// traverse each element of the array

// compare them with the pivot

for (int j = low; j < high; j++) {

if (array[j] <= pivot) {

// if element smaller than pivot is found

// swap it with the greater element pointed by i

i++;

// swap element at i with element at j

swap(&array[i], &array[j]);

}

}

// swap the pivot element with the greater element at i

swap(&array[i + 1], &array[high]);

// return the partition point

return (i + 1);

}

void quickSort(int array[], int low, int high) {

if (low < high) {

// find the pivot element such that

// elements smaller than pivot are on left of pivot

// elements greater than pivot are on right of pivot

int pi = partition(array, low, high);

// recursive call on the left of pivot

quickSort(array, low, pi - 1);

// recursive call on the right of pivot

quickSort(array, pi + 1, high);

}

}

// shell sort

void shellSort(int array[], int n) {

// Rearrange elements at each n/2, n/4, n/8, ... intervals

for (int interval = n / 2; interval > 0; interval /= 2) {

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

int temp = array[i];

int j;

for (j = i; j >= interval && array[j - interval] > temp; j -= interval) {

array[j] = array[j - interval];

}

array[j] = temp;

}

}

}

int main() {

int n; // length of array

printf("Length of array: 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]);

}

// input k and l

int k, l;

printf("Input k: ");

scanf("%d", &k);

printf("Input l: ");

scanf("%d", &l);

// display the original array

printf("Original array:\n[");

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

if (i == n-1) {

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

} else {

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

}

}

int copyArr[n];

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

copyArr[i] = arr[i];

}

invertWithinRange(copyArr, k, l);

// display the modified array

printf("Modified array:\n[");

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

if (i == n-1) {

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

} else {

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

}

}

// quick sort the original array

quickSort(arr, 0, n-1);

// display the sorted array in ascending order

printf("Sorted original array with quick sort ascendingly:\n[");

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

if (i == n-1) {

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

} else {

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

}

}

// display the sorted array in descending order

printf("Sorted original array with quick sort descendingly:\n[");

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

if (i == 0) {

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

} else {

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

}

}

// shell sort the modified array

shellSort(copyArr, n);

// display the sorted array in ascending order

printf("Sorted modified array with shell sort ascendingly:\n[");

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

if (i == n-1) {

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

} else {

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

}

}

// display the sorted array in descending order

printf("Sorted modified array with shell sort descendingly:\n[");

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

if (i == 0) {

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

} else {

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

}

}

return 0;

}

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

#include <stdio.h>

#include <stdbool.h>

void invertWithinRange(int \*arr, int k, int l) {

if ((l-k)%2==0) {

int j = l-1;

for (int i = k; i <= ((l-k+1)/2); i++) {

int temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

j--;

}

} else {

int j = l-1;

for (int i = k; i < ((l-k+1)/2); i++) {

int temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

j--;

}

}

}

void swap(int \*a, int \*b) {

int t = \*a;

\*a = \*b;

\*b = t;

}

int partition(int \*array, int low, int high) {

int pivot = array[high];

int i = (low - 1);

for (int j = low; j < high; j++) {

if (array[j] <= pivot) {

i++;

swap(&array[i], &array[j]);

}

}

swap(&array[i + 1], &array[high]);

return (i + 1);

}

void quickSort(int \*array, int low, int high) {

if (low < high) {

int pi = partition(array, low, high);

quickSort(array, low, pi - 1);

quickSort(array, pi + 1, high);

}

}

void shellSort(int \*array, int n) {

for (int interval = n / 2; interval > 0; interval /= 2) {

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

int temp = array[i];

int j;

for (j = i; j >= interval && array[j - interval] > temp; j -= interval) {

array[j] = array[j - interval];

}

array[j] = temp;

}

}

}

int main() {

int n;

printf("Length of array: n = ");

scanf("%d", &n);

int arr[n];

printf("Array:\n");

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

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

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

}

int k, l;

printf("Input k: ");

scanf("%d", &k);

printf("Input l: ");

scanf("%d", &l);

printf("Original array:\n[");

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

if (i == n-1) {

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

} else {

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

}

}

int copyArr[n];

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

copyArr[i] = arr[i];

}

invertWithinRange(copyArr, k, l);

printf("Modified array:\n[");

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

if (i == n-1) {

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

} else {

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

}

}

quickSort(arr, 0, n-1);

printf("Sorted original array with quick sort ascendingly:\n[");

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

if (i == n-1) {

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

} else {

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

}

}

printf("Sorted original array with quick sort descendingly:\n[");

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

if (i == 0) {

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

} else {

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

}

}

shellSort(copyArr, n);

printf("Sorted modified array with shell sort ascendingly:\n[");

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

if (i == n-1) {

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

} else {

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

}

}

printf("Sorted modified array with shell sort descendingly:\n[");

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

if (i == 0) {

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

} else {

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

}

}

return 0;

}

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

#include <stdio.h>

#include <stdbool.h>

// nr 15

// invert all elements of the array located in the substring determined by the exclusive range of indexes k and l (which will be read from the keyboard)

// divide by 2 all even elements that were inverted, and multiply by 3 all multiples of 3 the elements that were inverted

// display the original and the modified vectors

// sort the original vector ascending and descending, applying the counting sort method and display the sorting results

// sort the modified vector in ascending and descending orders, applying the merge sort method and display the sorting results

void invertWithinRange(int arr[], int k, int l) {

if ((l-k)%2==0) { // if there are an even number of elements to invert

int j = l-1;

for (int i = k; i <= ((l-k+1)/2); i++) {

int temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

j--;

}

} else { // if there are an odd number of elements to invert

int j = l-1;

for (int i = k; i < ((l-k+1)/2); i++) {

int temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

j--;

}

}

for (int i = k; i < l; i++) {

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

arr[i] = arr[i] / 2;

} else if (arr[i] % 3 == 0) {

arr[i] = arr[i] \* 3;

}

}

}

void countingSort(int array[], int size) {

int output[10];

// Find the largest element of the array

int max = array[0];

for (int i = 1; i < size; i++) {

if (array[i] > max)

max = array[i];

}

// The size of count must be at least (max+1) but

// we cannot declare it as int count(max+1) in C as

// it does not support dynamic memory allocation.

// So, its size is provided statically.

int count[10];

// Initialize count array with all zeros.

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

count[i] = 0;

}

// Store the count of each element

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

count[array[i]]++;

}

// Store the cummulative count of each array

for (int i = 1; i <= max; i++) {

count[i] += count[i - 1];

}

// Find the index of each element of the original array in count array, and

// place the elements in output array

for (int i = size - 1; i >= 0; i--) {

output[count[array[i]] - 1] = array[i];

count[array[i]]--;

}

// Copy the sorted elements into original array

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

array[i] = output[i];

}

}

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

int i, j, k;

int n1 = m - l + 1;

int n2 = r - m;

// Create temp arrays

int L[n1], R[n2];

// Copy data to temp arrays

// L[] and R[]

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

// into arr[l..r]

// Initial index of first subarray

i = 0;

// Initial index of second subarray

j = 0;

// Initial index of merged subarray

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) {

// Same as (l+r)/2, but avoids

// overflow for large l and 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);

}

}

int main() {

int n; // length of array

printf("Length of array: 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]);

}

// input k and l

int k, l;

printf("Input k: ");

scanf("%d", &k);

printf("Input l: ");

scanf("%d", &l);

// display the original array

printf("Original array:\n[");

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

if (i == n-1) {

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

} else {

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

}

}

int copyArr[n];

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

copyArr[i] = arr[i];

}

invertWithinRange(copyArr, k, l);

// display the modified array

printf("Modified array:\n[");

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

if (i == n-1) {

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

} else {

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

}

}

// counting sort the original array

countingSort(arr, n);

// display the sorted array in ascending order

printf("Sorted original array with counting sort ascendingly:\n[");

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

if (i == n-1) {

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

} else {

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

}

}

// display the sorted array in descending order

printf("Sorted original array with counting sort descendingly:\n[");

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

if (i == 0) {

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

} else {

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

}

}

// merge sort the modified array

mergeSort(copyArr, 0, n-1);

// display the sorted array in ascending order

printf("Sorted modified array with merge sort ascendingly:\n[");

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

if (i == n-1) {

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

} else {

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

}

}

// display the sorted array in descending order

printf("Sorted modified array with merge sort descendingly:\n[");

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

if (i == 0) {

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

} else {

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

}

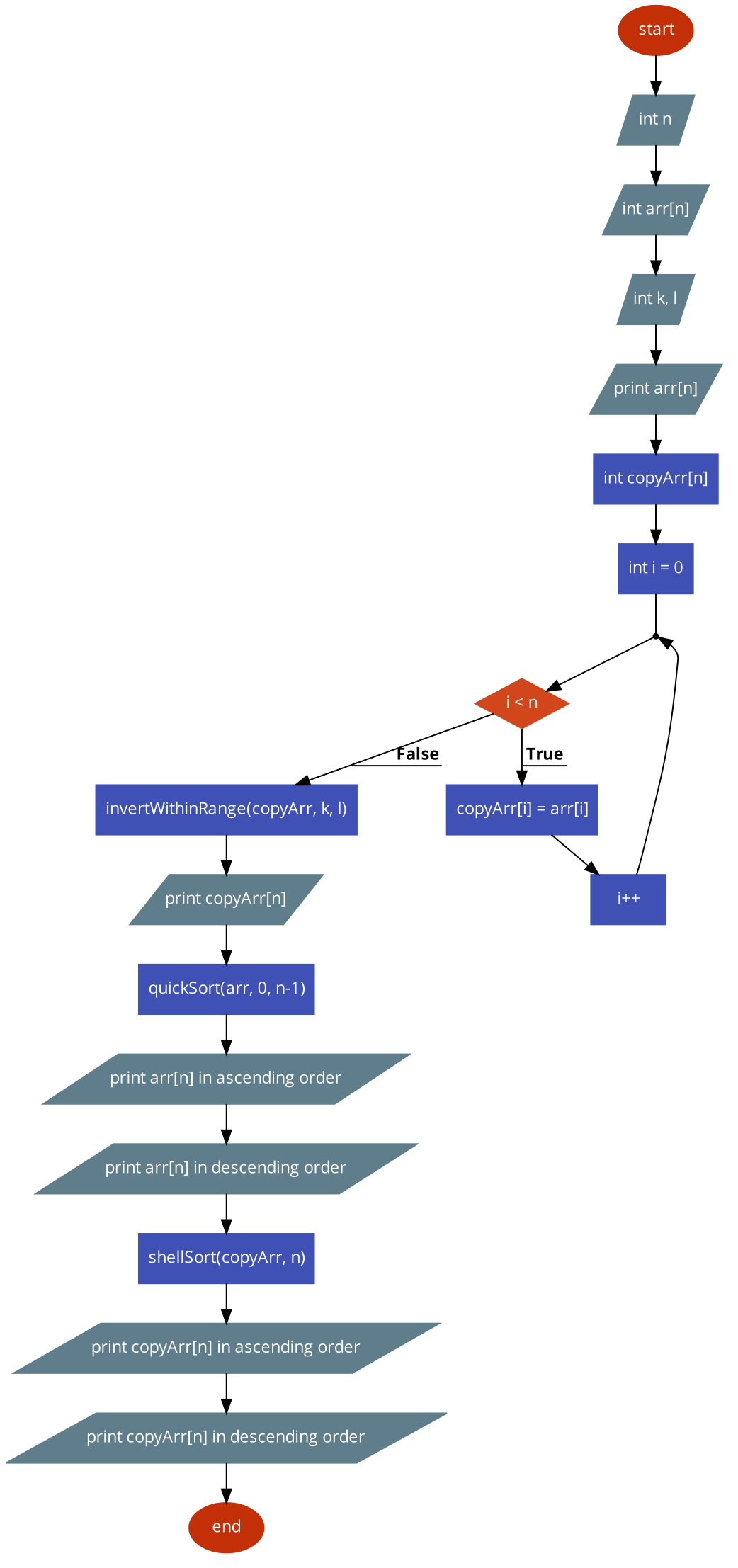
}

return 0;

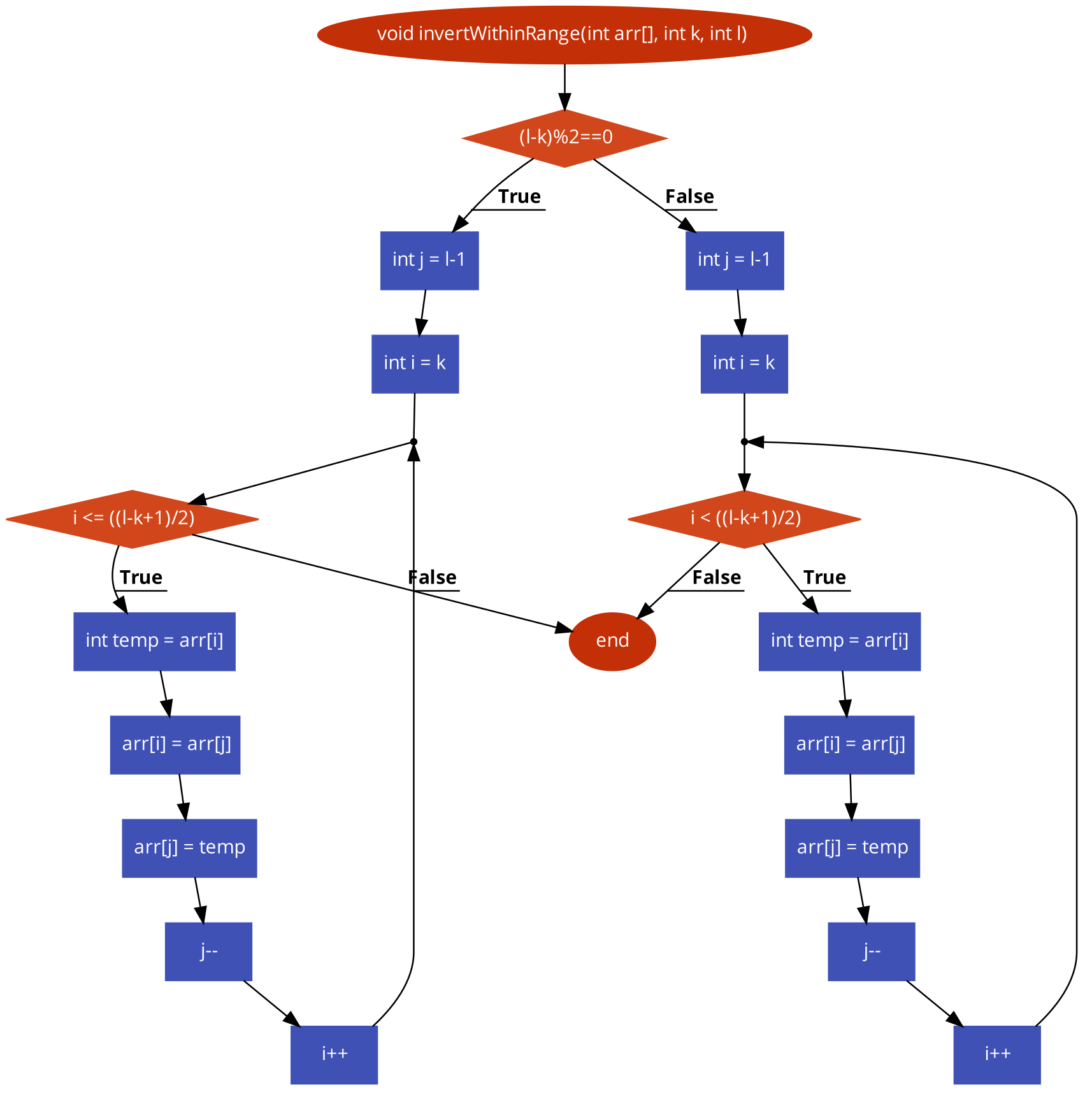
}

In this version I added an algorithm to divide by 2 all even elements that were inverted in the first condition, and multiply by 3 all multiples of 3 the elements that were inverted

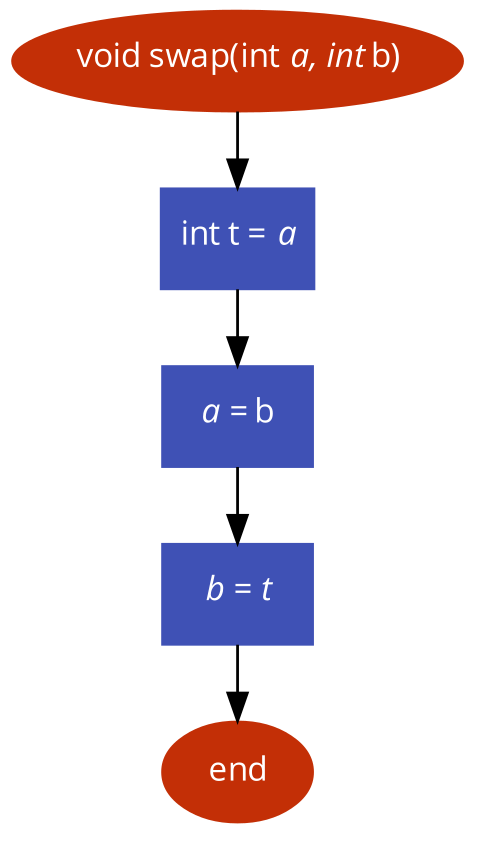
**Flowchart for original version:**



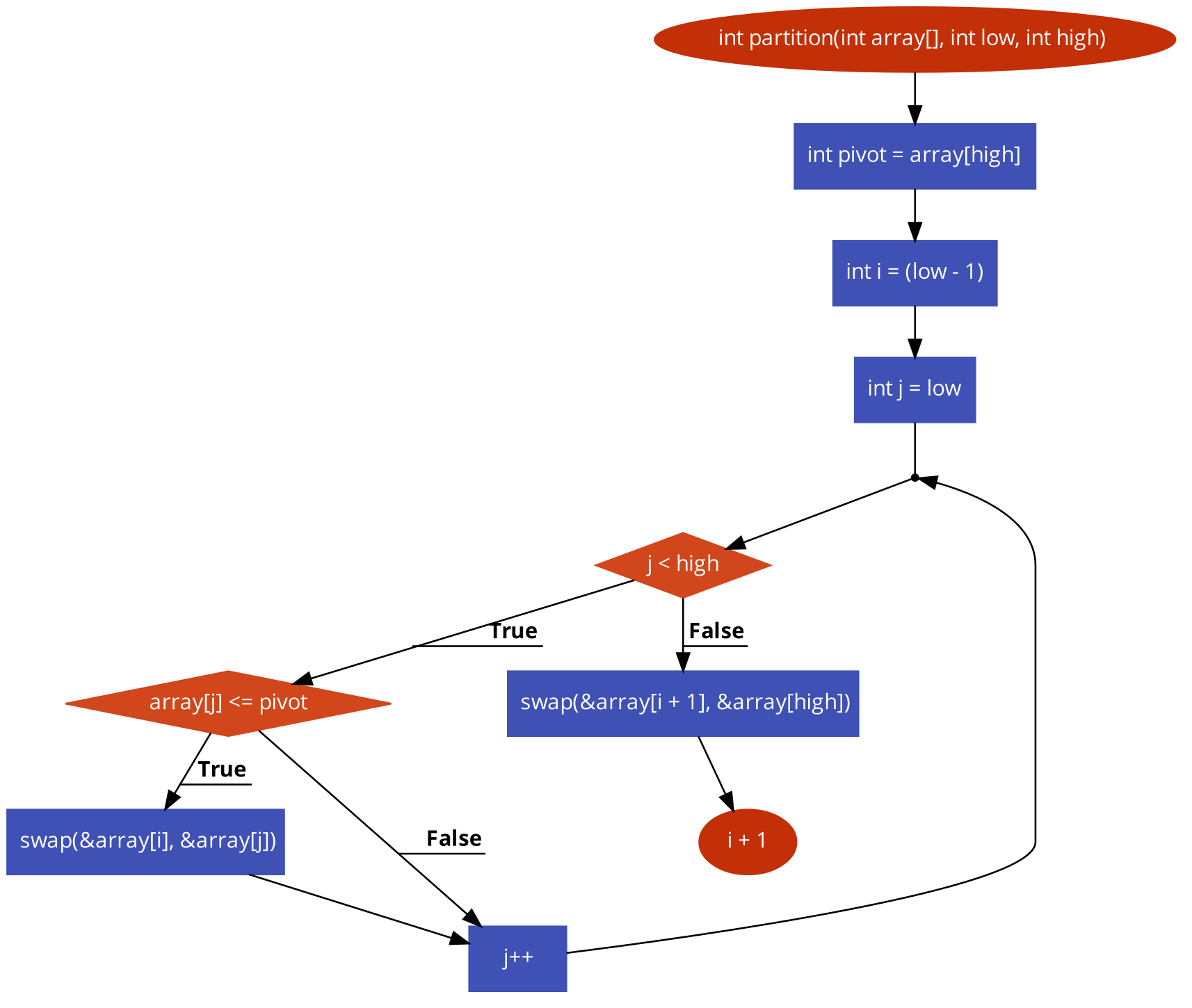
**Fig 2.1** – main function



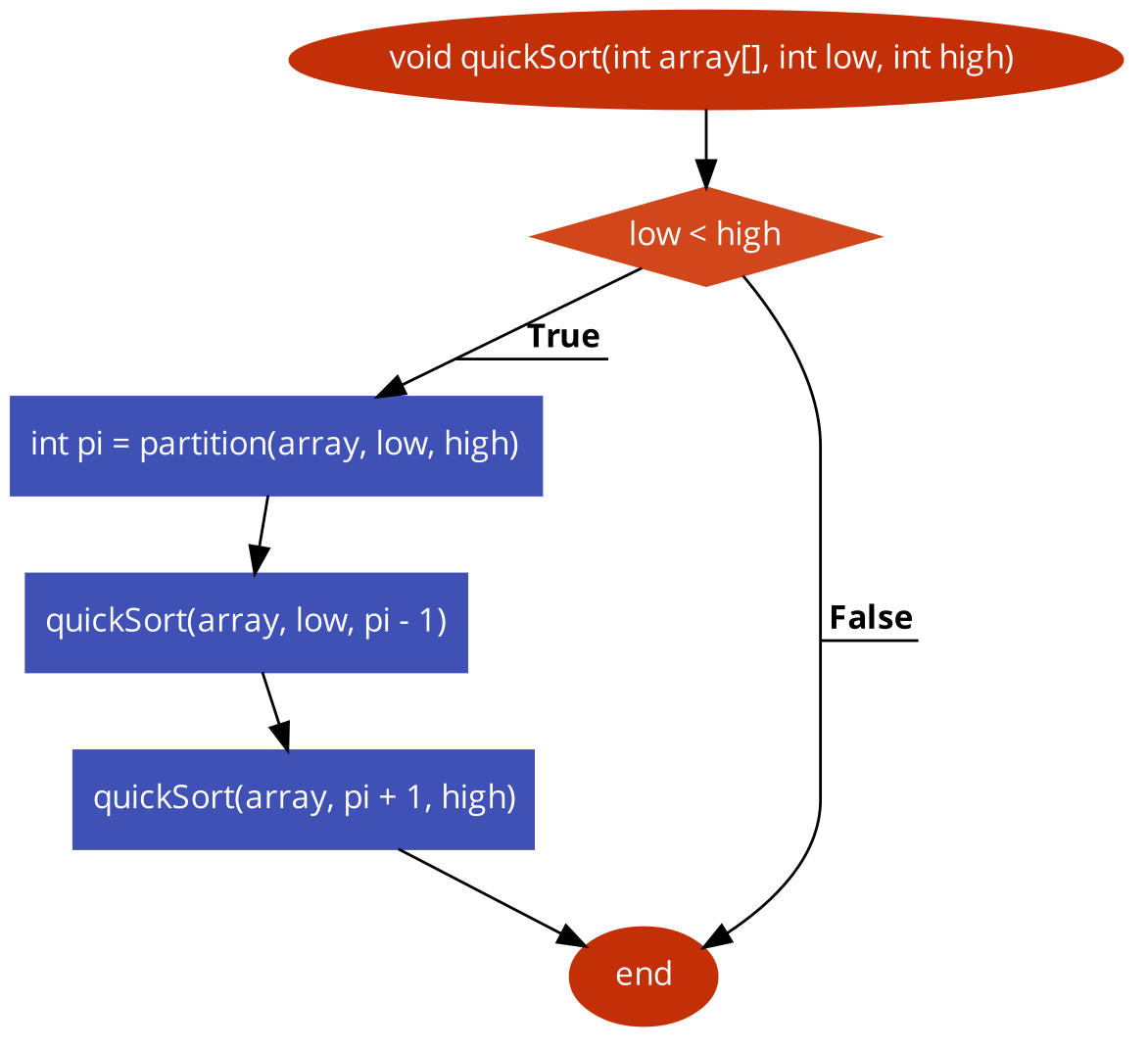
**Fig 2.2** – invertWithinRange function



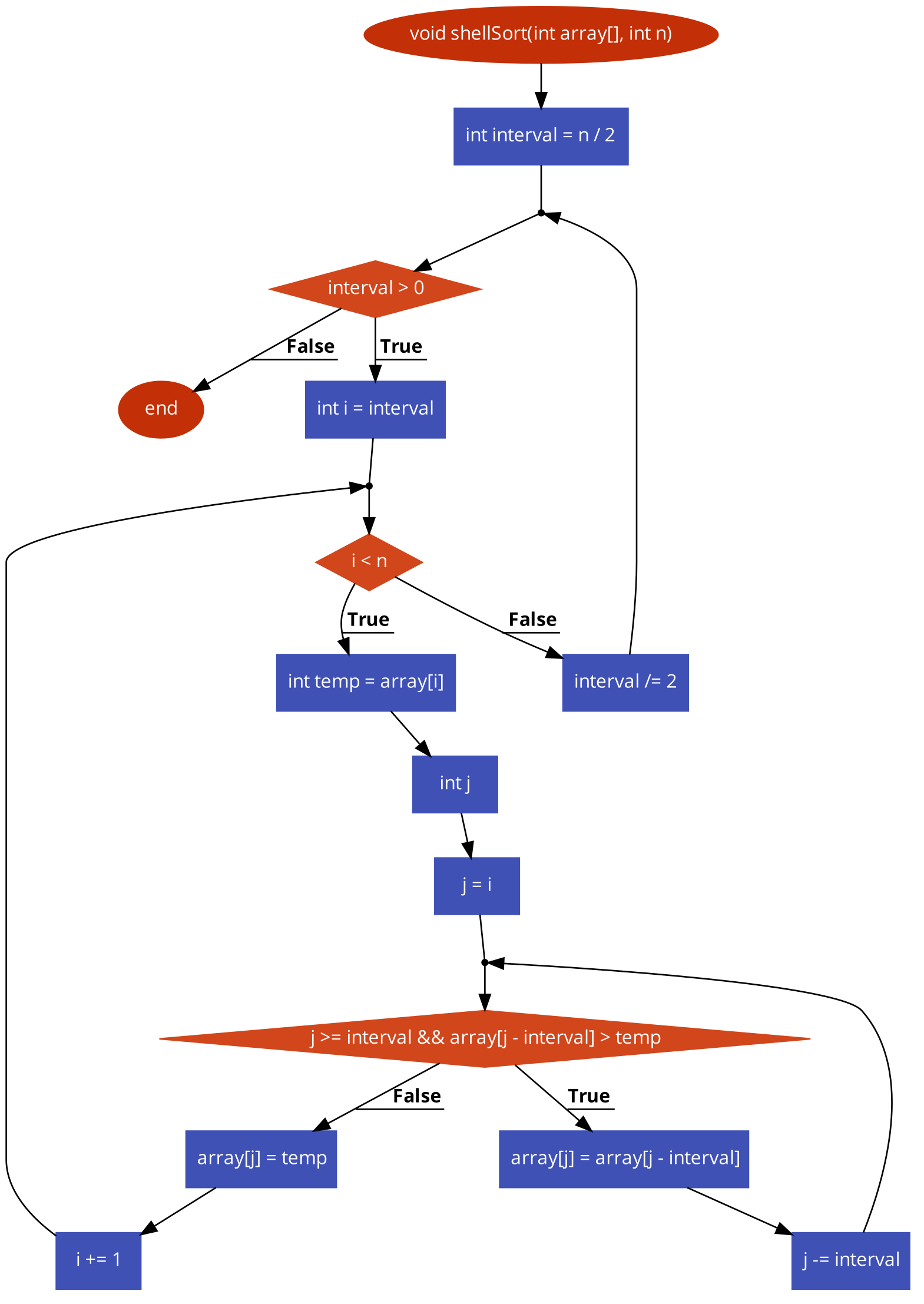
**Fig 2.3** – swap function



**Fig 2.7** – partition function

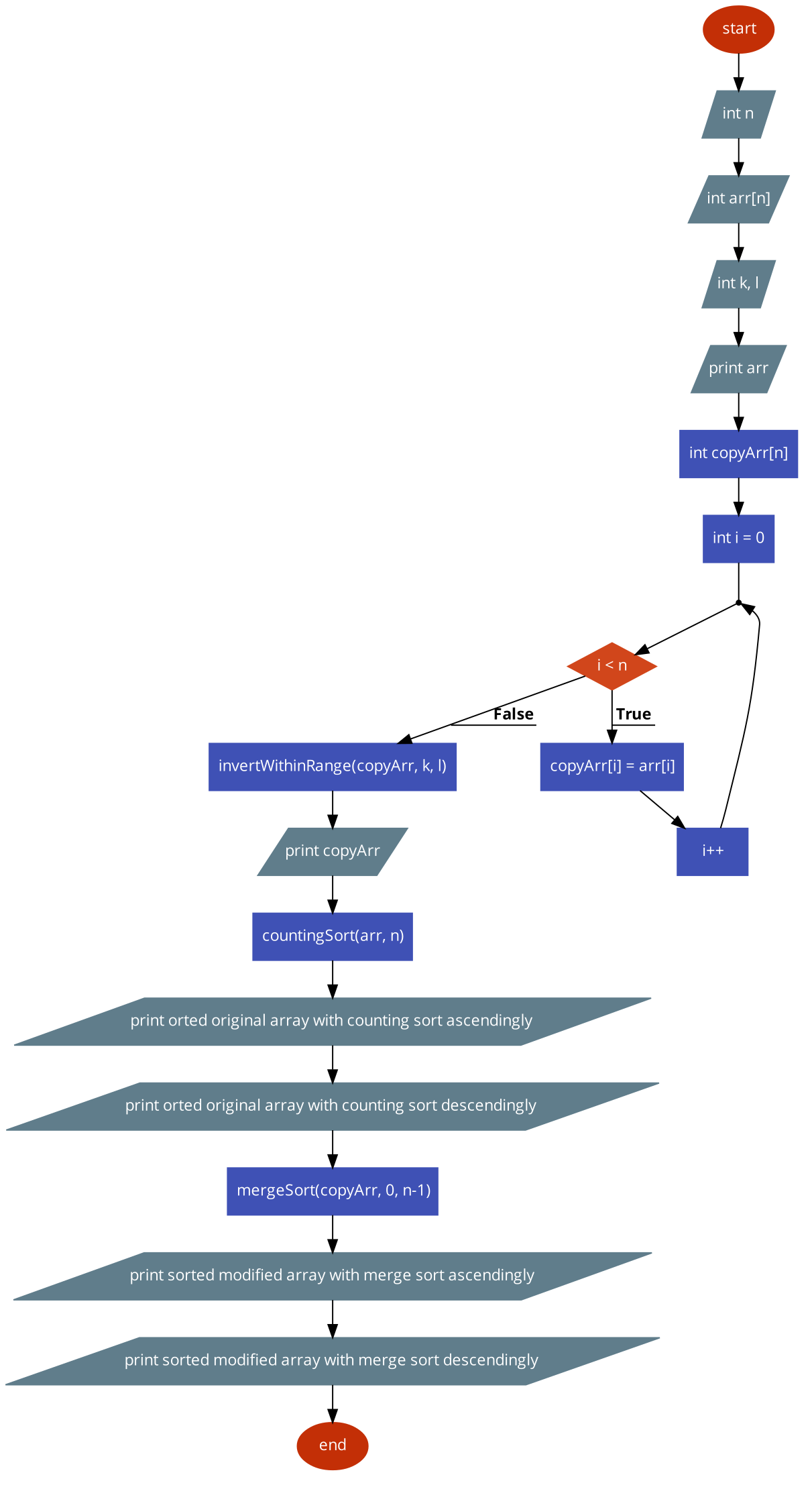


**Fig 2.8** – quick sort function

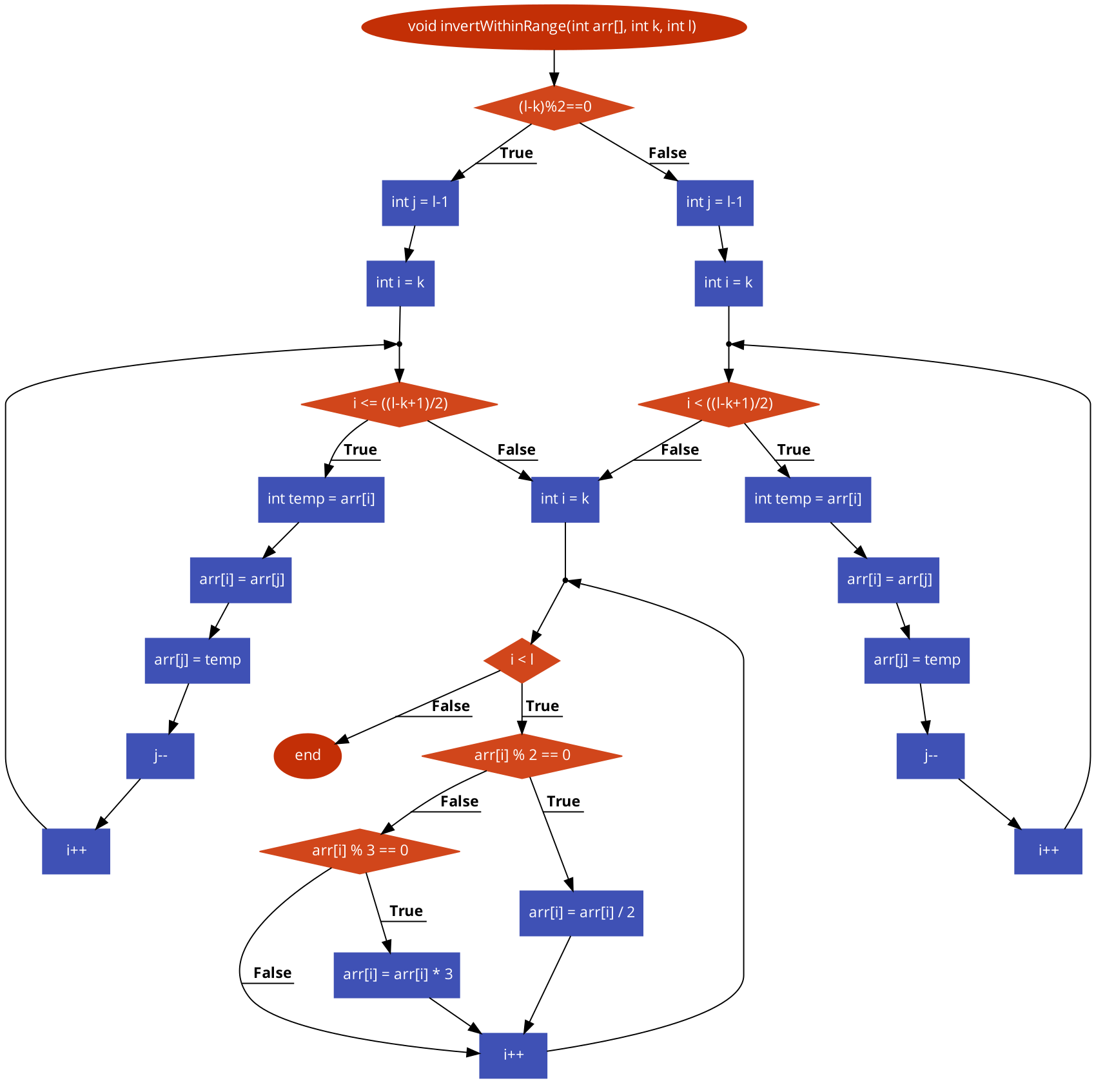


**Fig 2.9** – shell sort function

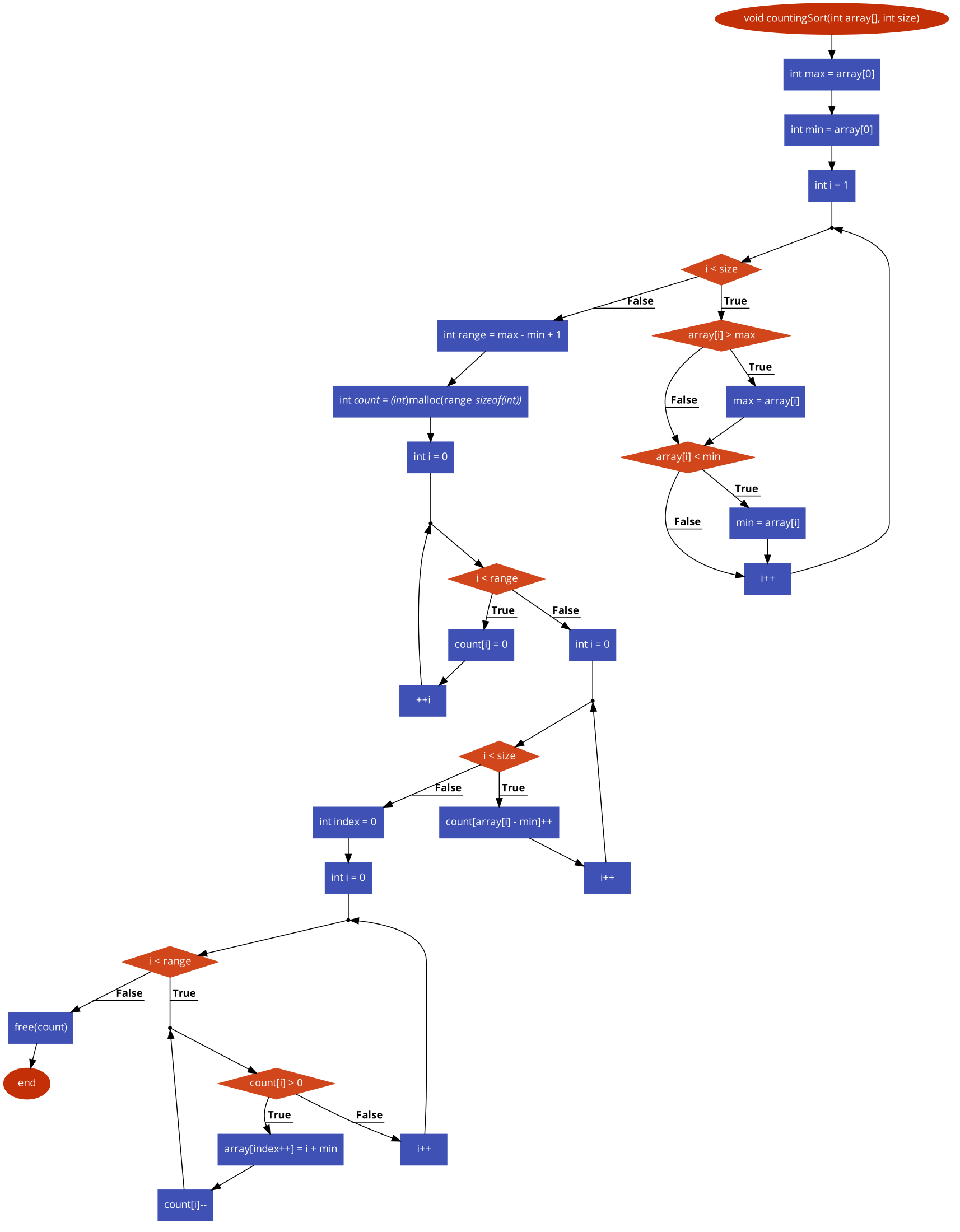
**Flowchart for modified version:**



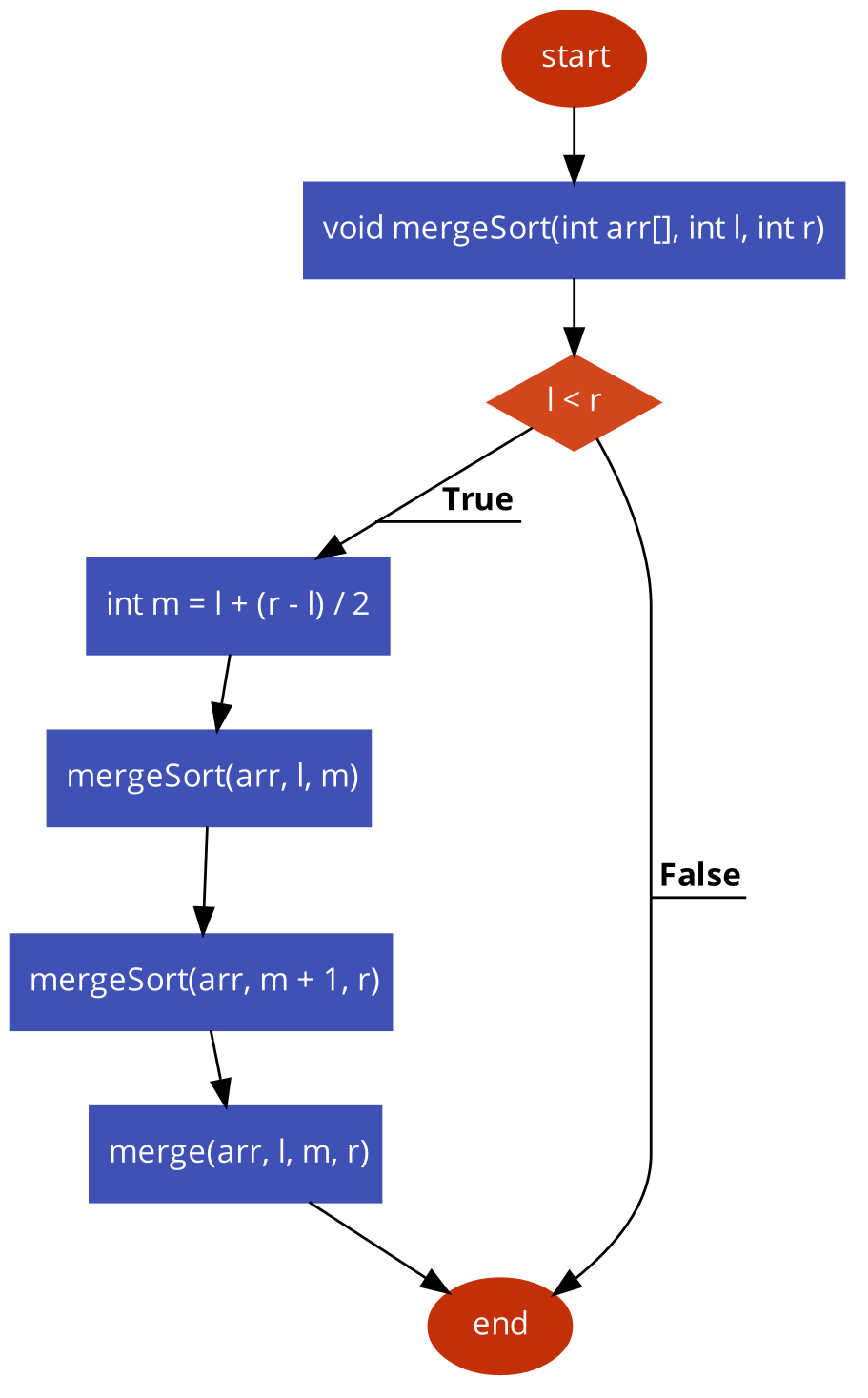
**Fig 2.10** – main function



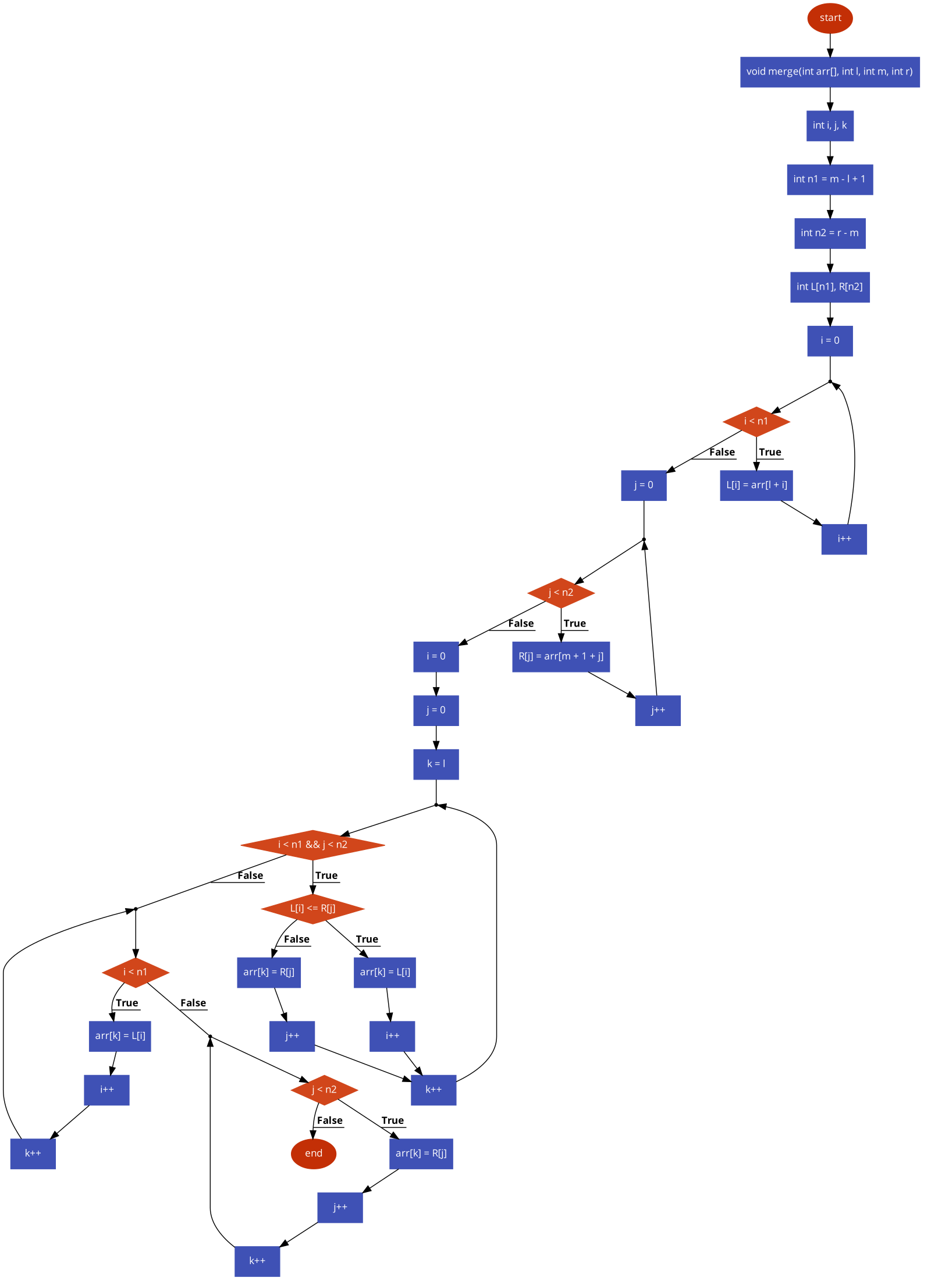
**Fig 2.11** – invertWithinRange function



**Fig 2.12** – countingSort function

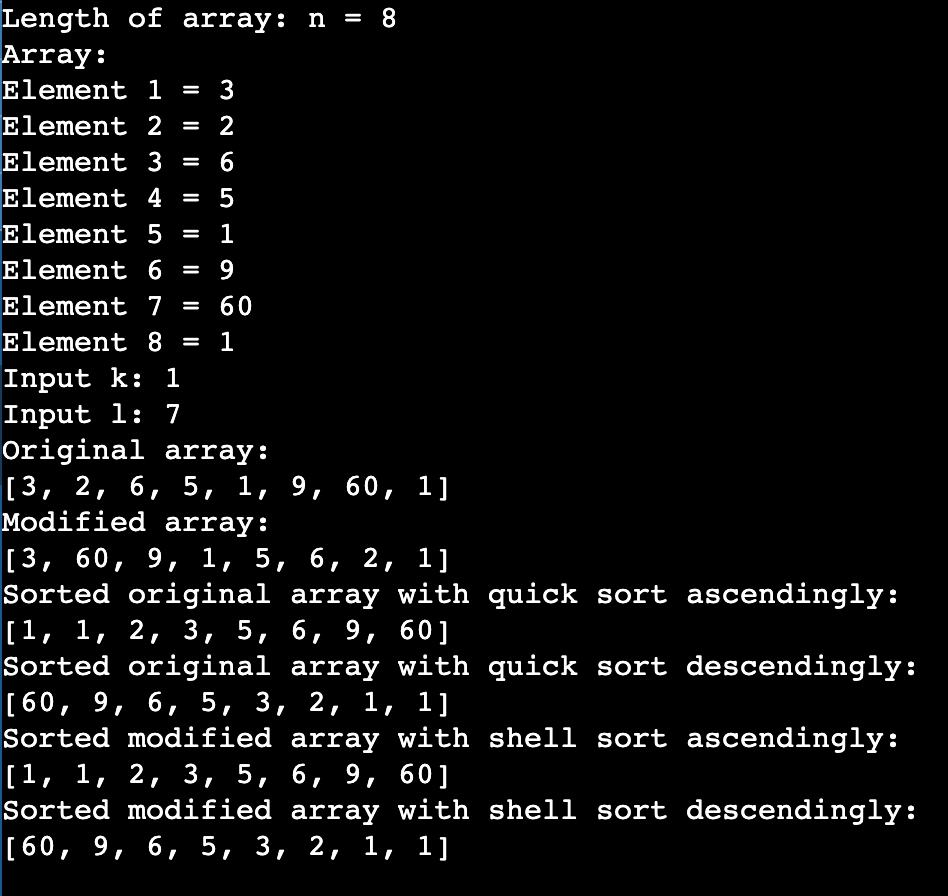


**Fig 2.13** – mergeSort function



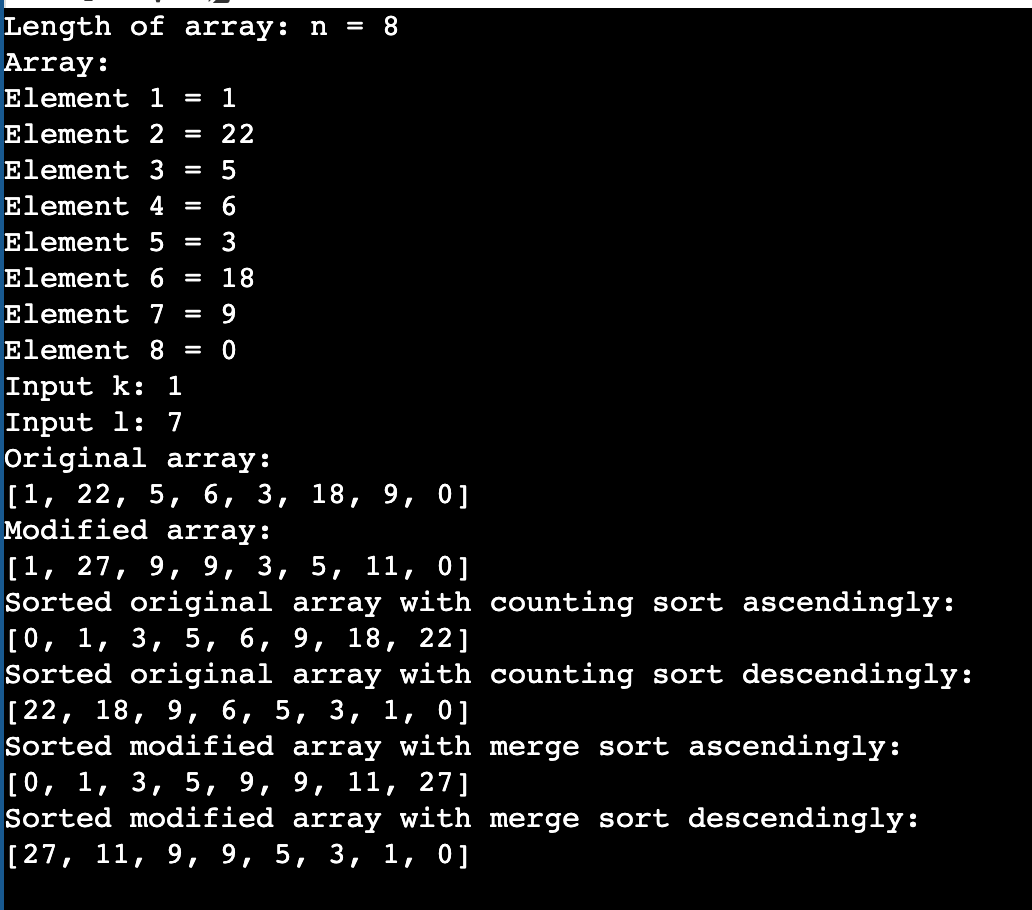
**Fig 2.14** – merge function

**Output first version:**



**Fig 1**

**Output modified version:**



**Fig 2**

**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 Quick Sort and Shell 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 work spans over 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.