****

**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**

**Țîganescu Anastasia**

**Group: FAF-231**

**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 – 2023**

**Purpose of laboratory work:**

Solving problems with 1-D arrays. It aims to strengthen programming skills

by working with 1-D arrays, array manipulation, algoritmic thinking and

problem-solving.

**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:

* 1. with the use of the method of transmitting the parametric functions by value;
  2. 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).
  3. 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

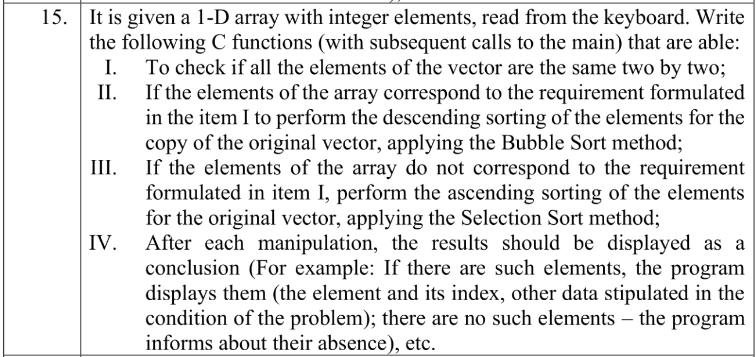
existing problem. Formulate and present in writing the modified condition; to

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:**



**Code of the program (having relevant comments in it):**

1. Version with the use of the method of transmitting the function parameters by value:

#include <stdio.h>

//The function which checks whether the elements are

//the same two by two

*int* checkpairs(*int* *a*, *int* *b*, *int* *c*, *int* *d*){

if (*a*==*c* && *b*==*d*){

return 1;

}

else {

return 0;

}

}

//The function which sorts the elements descendingly

//if the elements ARE the same two by two (bubble sort)

*void* bubblesort(*int* *array*[], *int* *n*){

*int* i, j, t;

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

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

if (*array*[j]<*array*[j+1]){

t = *array*[j];

*array*[j]=*array*[j+1];

*array*[j+1]=t;

}

}

}

}

//The function which sorts the elements ascendingly

//if they are NOT the same two by two (selection sort)

*void* selectionsort(*int* *array*[], *int* *n*){

*int* i, j, min, minind;

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

min = *array*[i];

minind=i;

for(j=i;j<*n*;j++){

if (*array*[j]<min){

min = *array*[j];

minind = j;

}

}

*array*[minind]=*array*[i];

*array*[i]=min;

}

}

*int* main(){

*int* A[100], n, i, copy[100];

//Reading the number of elements for the vector

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

//Reading the actual vector elements and copying them

//in another vector

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

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

copy[i]=A[i];

}

//This block of code traverses the array and calls the checkpairs function

//which compares each pair of consecutive elements with the next pairs

*int* status = 1;

*int* k=0;

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

for (k=i+2; k<n;k+=2){

status = checkpairs(A[i], A[i+1],A[k], A[k+1]);

if (status == 0){

break;

}

}

if (status == 0){

break;

}

}

//The conclusion (whether the elements are equal two by two or not)

//is displayed on the screen

if(status == 1){

printf("All elements are equal two by two.\n");

}

else {

printf("Not all elements are equal two by two.\n");

}

//If the elements of the vectors are the same two by two (status=1), I had to sort a copy

//of the original vector descendingly, using bubble sort.

//If the opposite is true (status = 0), I had to sort the actual vector

//ascendingly using selection sort

if (status==1){

bubblesort(copy, n);

printf("Descending sorting using bubble sort:\n");

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

printf("%d ", copy[i]); //the sorted copy of the vector is printed

}

}

else {

selectionsort(A, n);

printf("Ascending sorting using selection sort:\n");

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

printf("%d ", A[i]); //the sorted vector is printed

}

}

return 0;

}

2. Version with the use of the method of passing the function parameters by pointers.

#include <stdio.h>

//The function which checks whether the elements are

//the same two by two

*int* checkpairspointer(*int* \**array*, *int* *n*){

*int* i,k, status;

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

for (k=i+2; k<*n*;k+=2){

if (\*(*array*+i)==\*(*array*+k) && \*(*array*+i+1)==\*(*array*+k+1)){

status = 1;}

else{

status = 0;

return status;

}

}

}

return status;

}

//The function which sorts the elements descendingly

//if the elements ARE the same two by two (bubble sort)

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

*int* i, j, t;

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

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

if (\*(*array*+j)<\*(*array*+j+1)){

t = \*(*array*+j);

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

\*(*array*+j+1)=t;

}

}

}

}

//The function which sorts the elements ascendingly

//if they are NOT the same two by two (selection sort)

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

*int* i, j, min, minind;

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

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

minind=i;

for(j=i;j<*n*;j++){

if (\*(*array*+j)<min){

min = \*(*array*+j);

minind = j;

}

}

\*(*array*+minind)=\*(*array*+i);

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

}

}

*int* main(){

*int* A[100], n, i, copy[100];

//Reading the number of elements for the vector

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

//Reading the actual vector elements and copying them

//in another vector

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

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

copy[i]=A[i];

}

//This block of code calls the checkpairspointer function

//which compares each pair of consecutive elements with the next pairs

*int* status = 1;

status = checkpairspointer(A, n);

//The conclusion (whether the elements are equal two by two or not)

//is displayed on the screen

if(status == 1){

printf("All elements are equal two by two.\n");

}

else {

printf("Not all elements are equal two by two.\n");

}

//If the elements of the vectors are the same two by two (status=1), I had to sort a copy

//of the original vector descendingly, using bubble sort.

//If the opposite is true (status = 0), I had to sort the actual vector

//ascendingly using selection sort

if (status==1){

bubblesort(copy, n);

printf("Descending sorting using bubble sort:\n");

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

printf("%d ", copy[i]);//the sorted copy of the vector is printed

}

}

else {

selectionsort(A, n);

printf("Ascending sorting using selection sort:\n");

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

printf("%d ", A[i]); //the sorted vector is printed

}

}

return 0;

}

3. Modified version

The condition for my new version was to check whether all the sums of pairs of consecutive elements are equal. It the condition was met, then I had to sort a copy of the original vector descendingly, using insertion sort. If the condition was not met, then I had to sort the original vector ascendingly, using merge sort.

#include <stdio.h>

//The function which checks whether all the sums of pairs of consecutive

//elements are equal

*int* checksums(*int* \**array*, *int* *n*){

*int* status, i;

for(i=0;i<*n*-3;i+=2){

if(\*(*array*+i)+\*(*array*+i+1)==\*(*array*+i+2)+\*(*array*+i+3)){

status = 1;

}

else{

status = 0;

return status;

}

}

return status;

}

//The function which sorts the elements descendingly

//if the sums ARE the same (insertion sort)

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

*int* i,j, t;

for(i=1;i<*n*;i++){

t = \*(*array*+i);

for(j=i-1;j>=0;j--){

if(t > \*(*array*+j)){

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

\*(*array*+j) = t;

}

}

}

}

//The function which sorts the elements ascendingly

//if the sums are NOT the same (merge sort)

*void* mergeascend(*int* \**array*, *int* *a*, *int* *m*, *int* *b*){

*int* i,j,k, n1, n2;

n1 = *m*-*a*+1;

n2=*b*-*m*;

*int* L[n1], R[n2];

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

L[i]=*array*[*a*+i];

}

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

R[j]=*array*[*m*+1+j];

}

i=0; j=0; k=*a*;

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

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

*array*[k]=L[i];

i++;

}

else{

*array*[k]=R[j];

j++;

}

k++;

}

while(i<n1){

*array*[k]=L[i];

k++;

i++;

}

while(j<n2){

*array*[k]=R[j];

k++;

j++;

}

}

//The function which divides the original array into smaller subarrays which would be merged

*void* dividemerge(*int* \**array*,*int* *a*, *int* *b*){

*int* m;

if(*a*<*b*){

m=(*a*+*b*)/2;

dividemerge(*array*, *a*,m);

dividemerge(*array*, m+1, *b*);

mergeascend(*array*, *a*,m ,*b*);

}

}

*int* main(){

*int* A[100], n, i, copy[100];

//Reading the number of elements for the vector

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

//Reading the actual vector elements and copying them

//in another vector

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

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

copy[i]=A[i];

}

//Calling the checksums function

*int* status = 1;

status = checksums(A, n);

//The conclusion (whether the sums are equal or not)

//is displayed on the screen

if(status == 1){

printf("All sums are equal.\n ");

}

else {

printf("Not all sums are equal.\n ");

}

//If the sums are equal(status=1), I had to sort a copy

//of the original vector descendingly, using insertion sort.

//If the opposite is true (status = 0), I had to sort the actual vector

//ascendingly using merge sort

if(status==1){

insertiondescend(copy, n);

printf("Descending sorting using insertion sort:\n");

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

printf("%d ", copy[i]);//the sorted copy of the vector is printed

}

} else{

dividemerge(A,0,n-1);

printf("Ascending sorting using merge sort:\n");

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

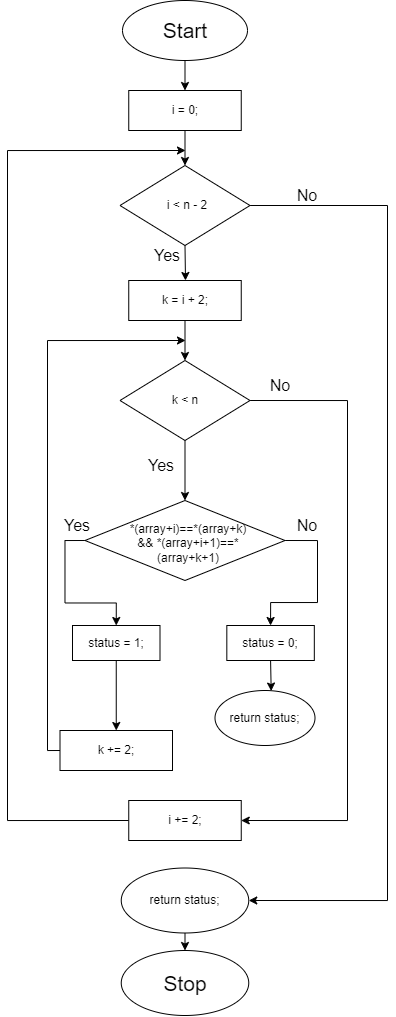
printf("%d ", A[i]); //the sorted original vector is printed

}}

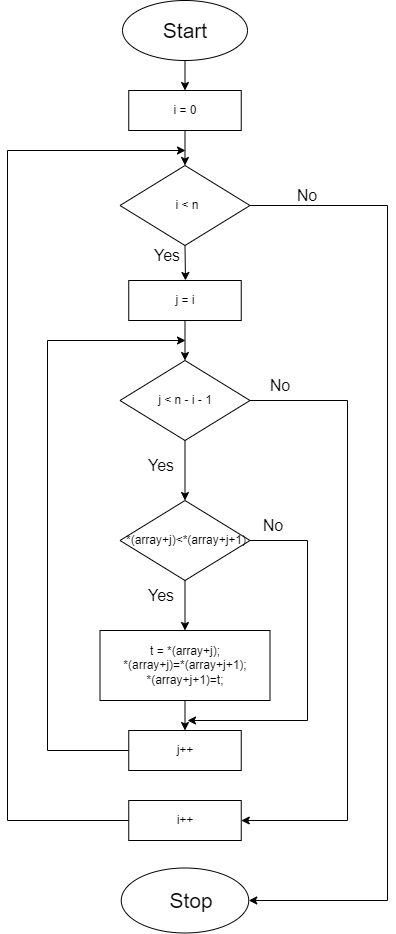
return 0;

}

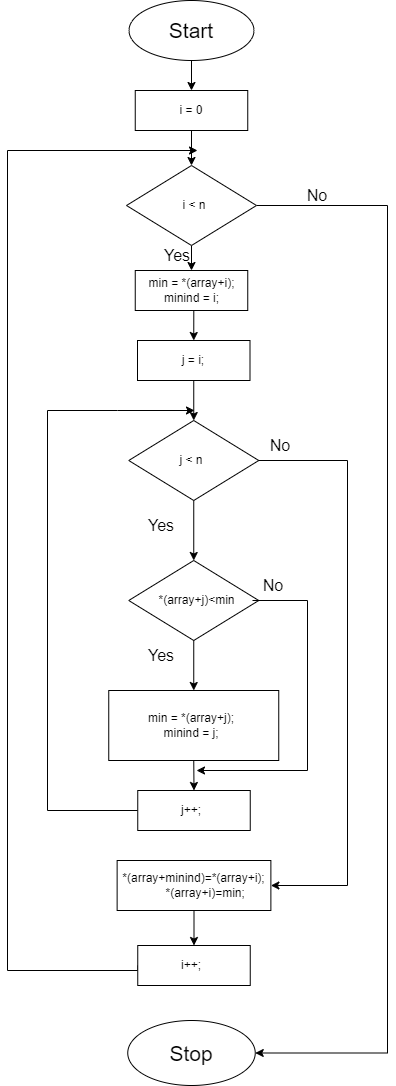
**Block diagrams (for the second version):**



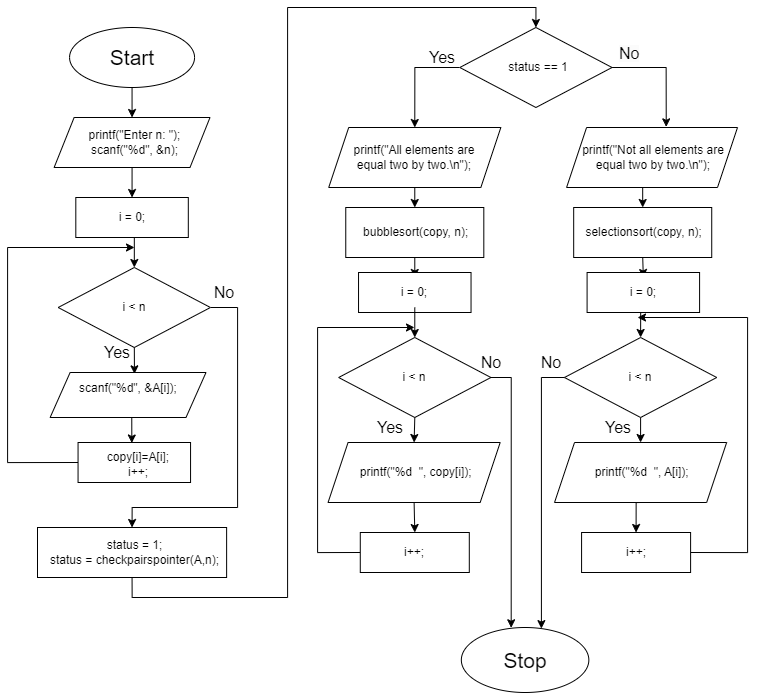
**Figure 1.1 -** *Function “checkpairspointer()”*



**Figure 1.2 -** *Function “bubblesort()”*

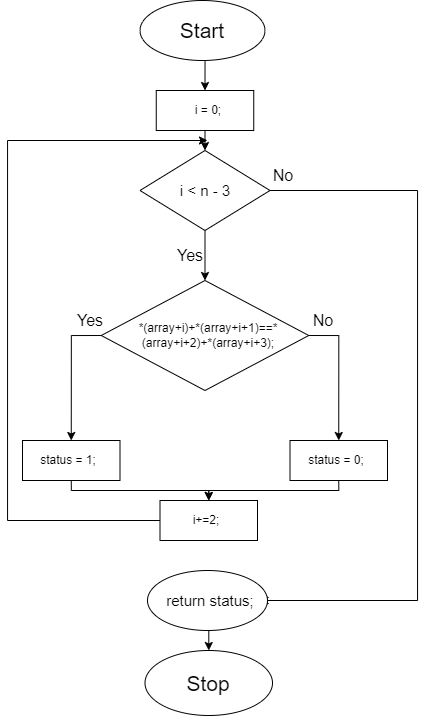


**Figure 1.3 -** *Function “selectionsort()”*

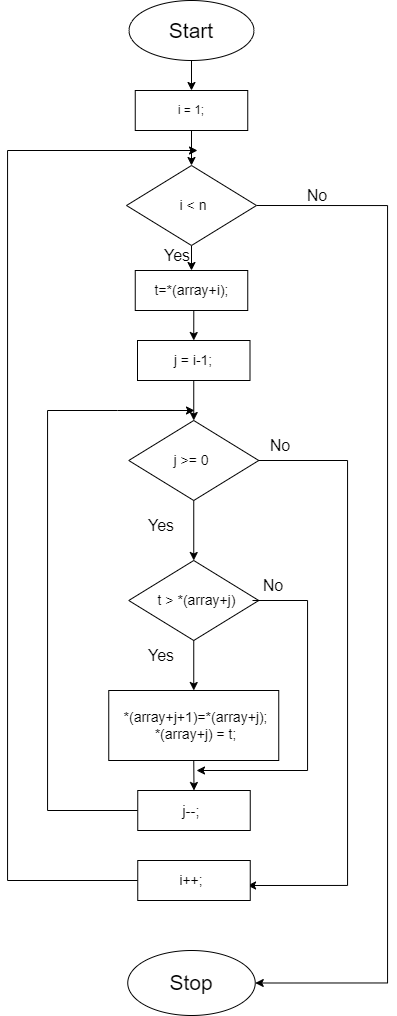


**Figure 1.4 -** *Function “main()”*

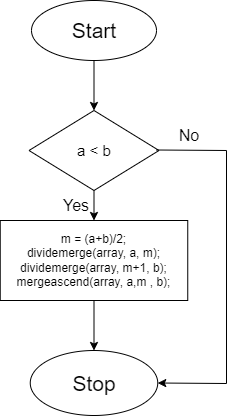
**Block diagrams (for the modified version):**



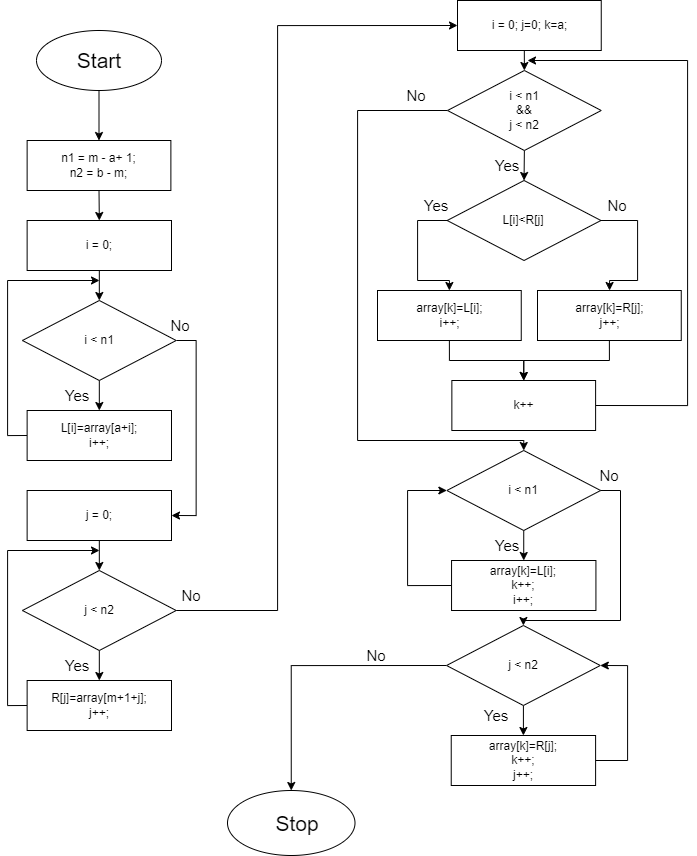
**Figure 2.1 -** *Function “checksums()”*



**Figure 2.2 -** *Function “insertiondescend()”*

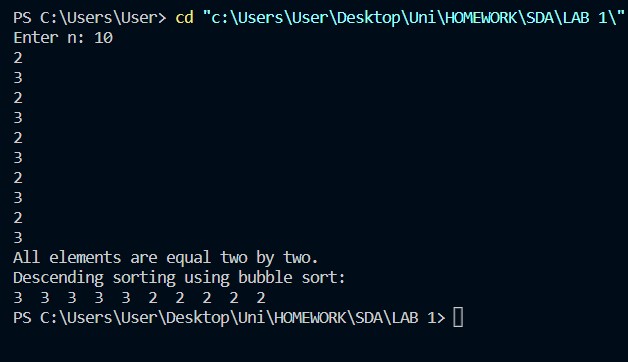


**Figure 2.3 -** *Function “dividemerge()”*

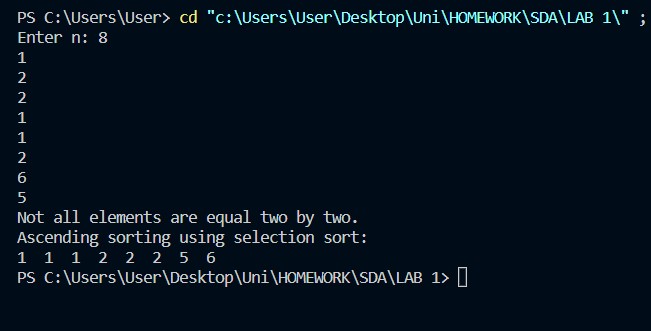


**Figure 2.2 -** *Function “mergeascend()”*

**Output (second version):**

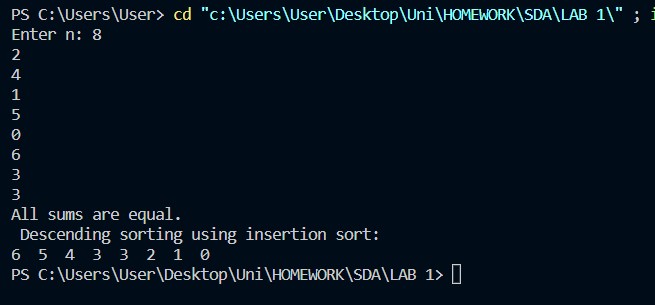


**Figure 3.1 -** *Output when all elements are equal two by two.*

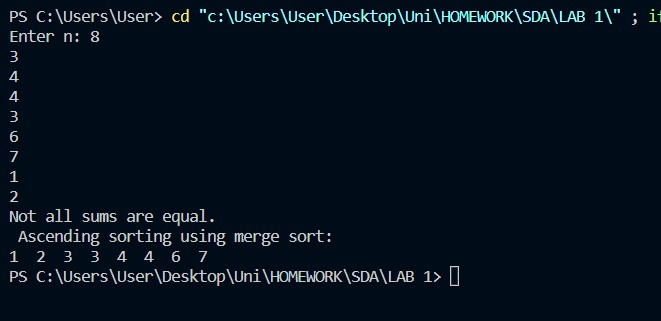


**Figure 3.2 -** *Output when not all elements are equal two by two.*

**Output (modified version):**



**Figure 3.3 -** *Output when all sums are equal.*



**Figure 3.4 -** *Output when not all sums are equal.*

**Conclusion:**

In this laboratory work, I dealt with vectors and sorting algorithms. I had to check if elements were the same two by two, then use descending bubble sort or ascending selection sort accordingly (in two ways). My modified version consisted on checking the sum of all consecutive pairs and then use insertion sort or merge sort based on the condition.

I managed to put to use the knowledge I gained during the lectures and the seminars and I was also provided with insights regarding efficiency of various sorting algorithms.

I took notice of how straightforward and easy to implement Selection and Bubble Sort are but, in spite of this, they are limited as the input size increases. Insertion sort outperformed the previous sorting arrays, however, the Merge sort proved to be the most efficient to use, being a divide-and-conquer method.

So, I learned the importance of choosing sorting algorithms based on diverse scenarios and also learned to implement them in reverse (descendingly).