Combinatorial Summation Primes: a Discussion of Different Methods to Solve the Problem

Kamil Książek
Faculty of Applied Mathematics
Silesian University of Technology
Gliwice, Poland
Email: kamiksi862@student.polsl.pl

Zbigniew Marszałek
Institute of Mathematics
Silesian University of Technology
Gliwice, Poland
Email: Zbigniew.Marszalek@polsl.pl

***Abstract*—This paper illustrates combinatorial summation numbers by means of computer operations. The aim of the research is to give a definition of all different primes which sum is equal 100, where the smallest combination has only two elements and the biggest has nine elements. This paper presents some ways to deal with this problem. There are analyzed two methods. One of them uses nested loops (non-recursive method), second way is connected with stacks (recursive method).**

***Index Terms*—prime, summation, stack, loop, analysis, recursion**

I. Introduction

The computational complexity of algorithms, one of the most important problem of computer science, is clearly one of the most important aspects of computer science. Although computers repeatedly increased its computing power, still micro machines are working on acceleration of existing algorithms. The problem remains an open question whether the higher performance algorithms are recursive, or may not recursive versions are faster.

There are reported many approaches to increase computational efficiency. Gabryel presented devoted methods to implement in data base systems, where implemented method serve as efficient tool for data management in high performance SQL environments [1] and human supporting systems for medical purposes by Wozniak et al. [20] and daily routine assistance by Damasevicius et al. [12]. Grycuk et al. presented devoted architectures to process visual data by clustering based on inverse frequency [2], and improved approach for multi-layer SQL architectures [3]. Czerwinski presented Hadoop implementation of data filtering [4], where information streaming was processed by developed system. Nowicki et al. developed intelligent processing of information in data mining with more efficient categorization approach [14]. Similarly to architectures algorithms with dedicated structures and commands have a great impact on development in computer science [10]. Carlsson et al. presented devoted sublinear sorting methods [5], Rauh proposed median based method [11]. Cole discussed efficiency in parallelization of processing [6], while Gubias extended this approach on various structures of input information [7], and Wozniak et al. proposed devoted versions of sorting methods developed for large data sets by the use of merge sort [19] and quick sort model [18]. Benchmark tests on improved versions of merge sort methods were presented by Marszalek et al. [17]. Practical approaches to implement sorting methods are presented by Huang and Langston [8], with extension for devoted main memory usage by Huang and Langston [9]. High efficiency request service models for SQL systems were presented by Wozniak et al. [13]. Similarly advances in man-machine interactions management can significantly improve efficiency and quality of service as presented by Polap et al. [15], [16].

This paper presents benchmark tests on improvements in computer methods by introduction of non recursive algorithm in comparison to its regular version. In the following sections we present different versions of combinatorial algorithm for determining the sum of the set of numbers using primes. Performance tests show the effectiveness of the proposed algorithm operating on the stake.

II. Nested loops 'for' - non-recursive method

In the research on efficiency of non recursive approach we have implemented two versions of algorithm used for combinatorial summation of primes that comply condition that a sum is equal 100.

A. Description

A prime is a number which has only two divisors: 1 and itself. The initial step is isolation primes from first 100 numbers. You can use a algorithm which works like the Sieve of Eratosthenes. By hand, we should create a table with numbers from 0 to 100, where 0 and 1 are not prime and not composite. We are starting our search from next number - 2, which is prime so every multiplies of 2 are composite. Now we can cross off these numbers from the table. Next number which is left on the table is 3. We should remove every multiplies of 3 and repeat the process until you find the last prime number less than 100. These are all steps of the implemented algorithm. To apply it you should create a boolean table (the last element will be '100'). A loop 'for' assigns *true* to all elements. Second loop ’for’ explore numbers from 2 to 100. If a constituent has a *true* value all its multiplies will be marked *false*. Every primes (with 0 and 1) are designated *true* now. The next step is to transfer all primes to separate array - there it will make necessary calculations.

After preparing the base for computations the program is finding all combinations which meet the conditions (the sum is equal 100). Nested loops ’for’ (their amount depend on elements of the combinations) are searching every components of array. Conditional statements choose only these different from each other, whose sum is equal 100. Effective procedure requires eight steps - *n-elements* combinations need *n nested loops ’for’*. If a combination meets the conditions, a method *’Write’* from *StreamWriter* is saving a result to the files. Every stage is stored in separate places. This method is non-recursive. The Block Diagram of the general part of this program is showed on Fig. 11. An example of nested loops ’for’ (4-elements combinations) is presented in Fig. 12.

B. *Information about research and benchmark tests*

Presented methods had been written in C++ CLR Microsoft Visual Studio 2013. We would like to describe two points of view in numbers summation. After the presentation of algorithms we are going to compare these methods. Researches had been done with *Stopwatch* from *System::Diagnostics* class. The measurements were performed at two time units: seconds and CPU ticks. CPU clock cycle allow to estimate how fast a processor carries out essential operation. Data based on 100 tests for all programs. The results are the arithmetic average. During the research has been used Intel Xeon CPU E3-1241 v3 3.50GHz.

C. *Analysis of efficiency*

This solution is not effective enough (compare to Fig. 1 - Fig. 4). The most important disadvantage is the length of calculation. Searching 2-7-elements combinations ends with success. There exist no 8-elements sequence and one 9-elements sequence that fills the conditions. A huge calculation causes slowing of computation. We should notice that the program permutes found combinations. It would be enough to write only one sequence. We have to eliminate duplications in files, ie. by our calculations based on the finding of repeated sequences without removing repetitions. You can see that non-recursive method is inefficient, but still the results are acceptable.

Analyzing Table I we can see that searching time connected with 1 to 6-elements combination is relatively low but in case for 7 and 9 elements is several times larger. There exist only one 9-elements combination which meets task’s condition so we have stopped tests after finding a first sequence with correct solution. If you tried apply this idea in similar way for instance to find different primes which sum equals more than 100, you would get results even slower.

III. Stacks - recursive method

Let us now discuss possible improvement by application of recursive approach, where instead of repetition of cycles

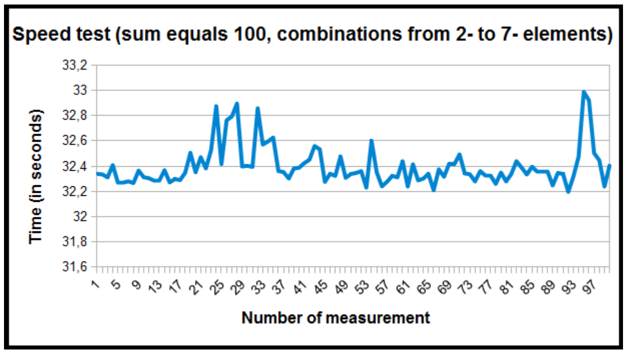

Figure 1. Chart of time of summation (Nested loops, sum equals 100, part 1)

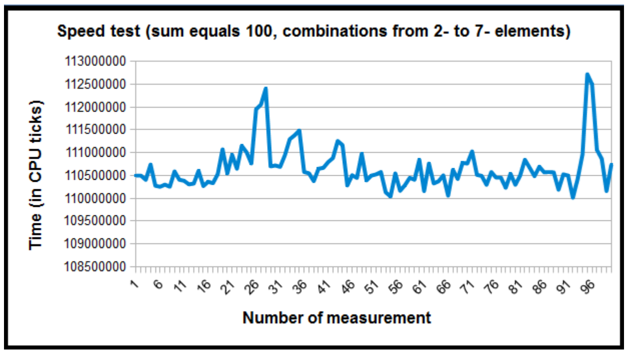

Figure 2. Chart of time of summation (Nested loops, sum equals 100, CPU clock cycles, part 1)

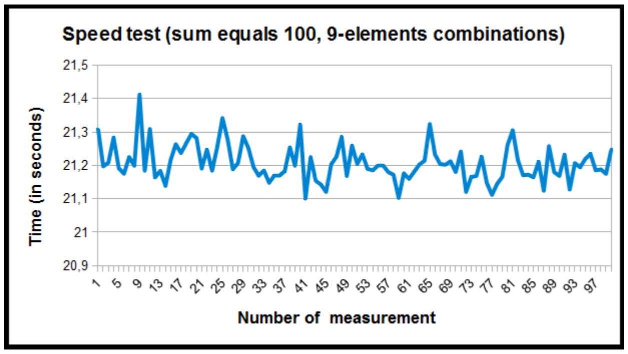

Figure 3. Chart of time of summation (Nested loops, sum equals 100, part 2)

we have introduced computational technique that enables self recalling in procedures for more efficient processing.

A. *Description*

There exists other method to solve our problem. We can use stacks (Last In First Out - LIFO) from namespace *System::Collections*. As we will see, this solution is more efficient and suitable. It is recursive method. The last algorithm was constructed only for one case (sum primes equals 100). In that situation you can easily change model.

Table I
ANALYSIS OF EFFICIENCY - NESTED LOOPS

number of elements	2	3	4	5	6	7	9
average (in seconds)	0.000038197	0.00095737	0.00338778	0.0552465	1.1553366	31.1835792	21.2075880
minimum (in seconds)	0,000019900	0.00005150	0.00229490	0.0427253	1.1486146	30.9947676	21.1009756
average (in CPU ticks)	130.9	327	11576	148634	3947798	106554599	72466562
minimum (in CPU ticks)	68	158	7842	145993	3924829	105909462	72102266
Combined Minimum Time (in seconds)					53.2894494		
Combined Average Time (in seconds)					53.6052720		
Combined Minimum Time (in CPU ticks)					182090618		
Combined Average Time (in CPU ticks)					183129627		

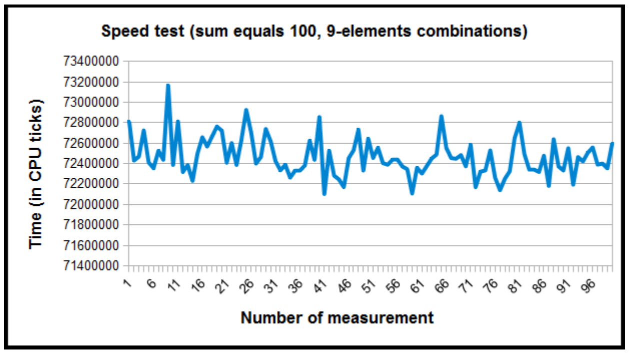

Figure 4. Chart of time of summation (Nested loops, sum equals 100, CPU clock cycles, part 2)

The first step is also finding all primes between 0 and 100. We can solve the problem similarly as before. The next stage is different. We should create two stacks: one "temporary" stack (a place, where we could store numbers) and nested stack which will collect sums equal 100. The construction is shown on Fig. 5. There could be used recursion connected with the amount of numbers. If the sum in "temporary" stack is more than 100, the algorithm will end the operation. A loop ’for’ gives to the stack primes (for instance "i") and runs recursion to "i-1". If the sum on the stack is equal 100, this result is added to nested stack (a collection of every solutions of our problem). The loop removes number "i" from the stack. The main part of the program is presented on Fig. 6. Optimal results are printed to the file.

1) Analysis of efficiency: This method is very efficient. Results are printed in one file and there is no repetition of our combinations. You can get solutions faster than for previous approach as it is presented in Tab. II.

You can easy modify this algorithm. Last program was constructed special for primes which sum is 100. In that option you can write other sum - the algorithm is universal. It is presented in Fig. 5 and Fig. 6. Using recursive searching with stacks is most favorable. We were doing also a research with other sum. The program have found primes which sum is equal 200. Results were very good: it was faster than finding last sum in the algorithm "nested loops".

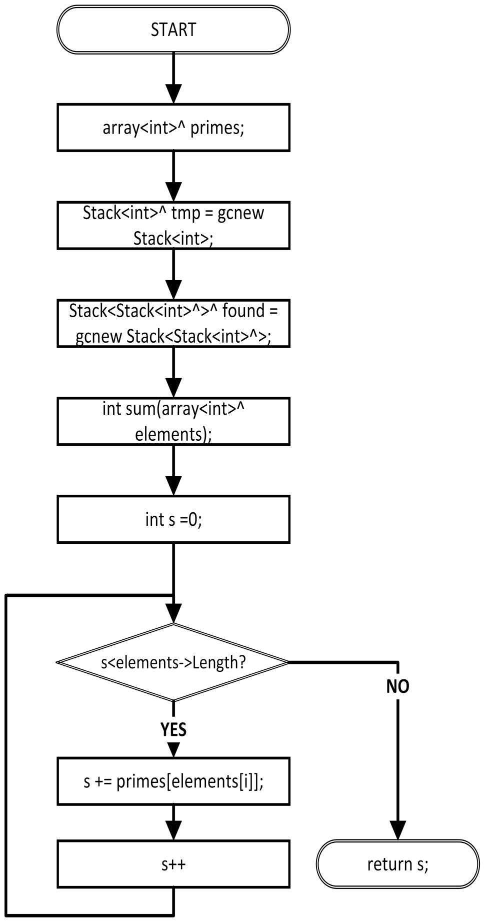
START
array<int>^ primes;
Stack<int>^ tmp = gcnew Stack<int>;
Stack<Stack<int>^>^ found = gcnew Stack<Stack<int>^>;
int sum(array<int>^ elements);
int s =0;
s<elements->Length?
YES
NO
s += primes[elements[i]];
s++
return s;

Figure 5. Block Diagram of the recursive program ’Stacks’

IV. COMPARISON AND DISCUSSION

There is a big difference between presented methods. The recursive program with stacks is several times faster than nested loops (non-recursive). We could see in Tab. I that combined minimum time in case of nested loops is 53.2894494 seconds but complete time in case of recursion is 0.0025118 seconds. In addition clarity of results is much better. Moreover, time necessary to finding every primes which sum equals 200 is also smaller than previous sum in nested loops: 0.272592 seconds.

The algorithm in second method is universal but in first case

Table II
ANALYSIS OF EFFICIENCY - STACKS

algorithm	arithmetic mean (in seconds)	arithmetic mean (CPU ticks)	minimum (in seconds)	minimum (in CPU ticks)
100 - recursion	0.00282584	9656.09	0.0025118	8583
200 - recursion	0.28448827	97212.83	0.272592	93415

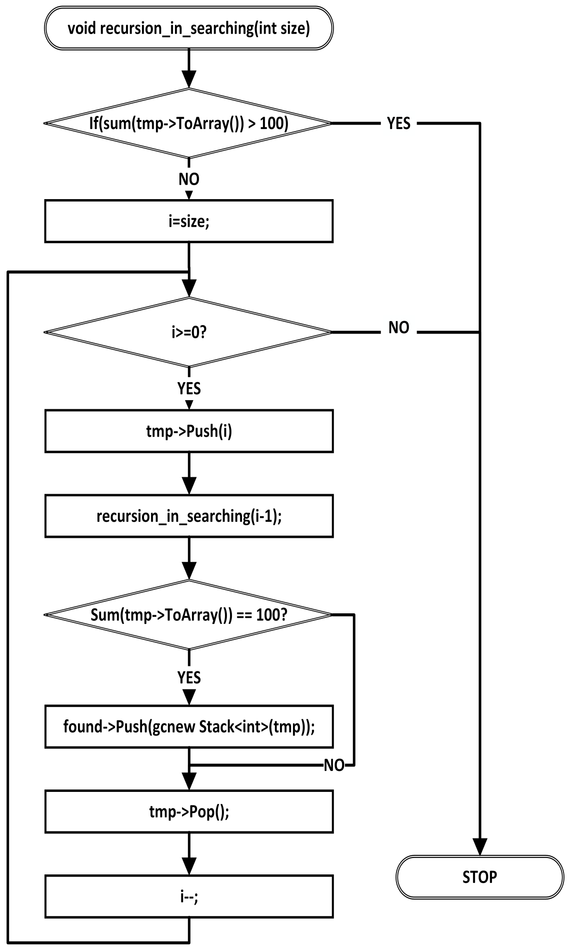
void recursion_in_searching(int size)
If(sum(tmp->ToArray()) > 100)
YES
NO
i=size;
i>=0?
NO
YES
tmp->Push(i)
recursion_in_searching(i-1);
Sum(tmp->ToArray()) == 100?
YES
found->Push(gcnew Stack<int>(tmp));
NO
tmp->Pop();
i--;
STOP

Figure 6. Block Diagram of the recursion in searching a specific sum.

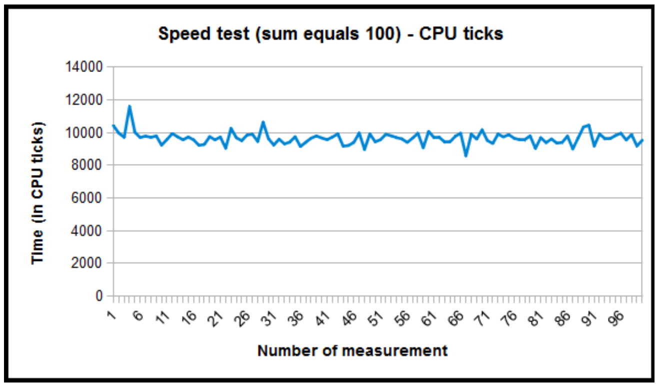

Figure 8. Chart of time of summation (Stack, sum equals 100, CPU clock cycles)

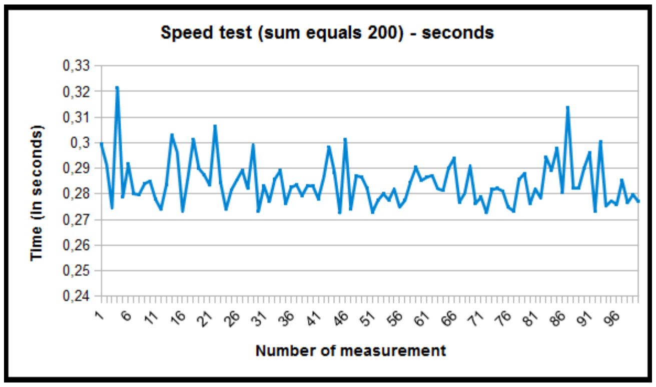

Figure 9. Chart of time of summation (Stack, sum equals 200)

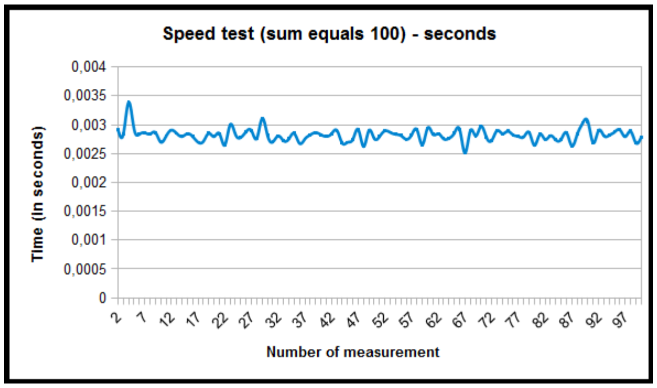

Figure 7. Chart of time of summation (Stack, sum equals 100)

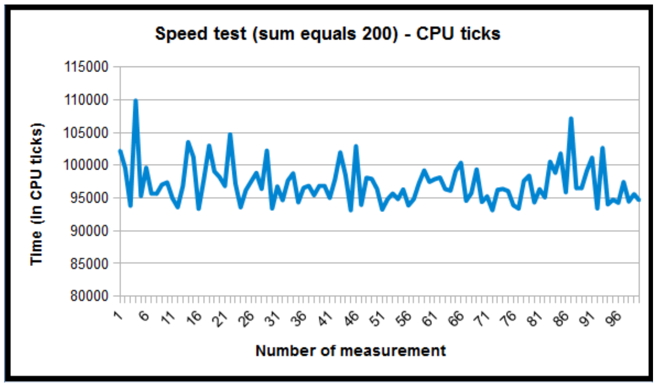

Figure 10. Chart of time of summation (Stack, sum equals 200, CPU clock cycles)

it could add loops which will explore more than 9-elements combinations. Structure of the method in 'nested loops' is not programmer friendly, since it is easy to get confused while programming this approach, however the general idea can be more clear for beginners. Recursive approach is not as easily understandable for beginners and may cause some problems in understanding, however it's structure is programmer friendly and easily computed in implemented approach.

V. Conclusions

We have compared two method implemented to determine summation primes. The recursive algorithm connected with stacks is several times faster and it is useful in other cases. The speed of operation is much better and versatility of this method allow to apply it in every sums. Recursive algorithm is very efficient in combinatorial summation of numbers.

In the next research we plan to improve proposed approach by introduction of parallelization to the code, what can help on more efficient processing and decrease computation time. Moreover it could be interesting to implement similar approach based on computational intelligence, to compare efficiency and discuss possible advantages.

References

[1] M. Gabryel, "The bag-of-features algorithm for practical applications using the mysql database," *Lecture Notes in Artificial Intelligence - ICAISC'2016*, vol. 9693, pp. 635–646, 2016, DOI: 10.1007/978-3-319-39384-1_56.

[2] R. Grycuk, M. Gabryel, M. Korytkowski, and R. Scherer, "Content-based image indexing by data clustering and inverse document frequency," *Communications in Computer and Information Science - BDAS'2014*, vol. 424, pp. 374–383, 2014, DOI: 10.1007/978-3-319-06932-6.

[3] R. Grycuk, M. Gabryel, R. Scherer, and S. Voloshynovskiy, "Multi-layer architecture for storing visual data based on WCF and microsoft SQL server database," *Lecture Notes in Artificial Intelligence - ICAISC'2015*, vol. 9119, pp. 715–726, 2015, DOI: 10.1007/978-3-319-19324-3.

[4] D. Czerwinski, "Digital filter implementation in hadoop data mining system," *Communications in Computer and Information Sciences - CN'2015*, vol. 522, pp. 410–420, 2015, DOI: 10.1007/978-3-319-19419-6_39.

[5] S. Carlsson, C. Levcopoulos, and O. Petersson, "Sublinear merging and natural merge sort," *Lecture Notes on Computer Science - SIGAL'1990*, vol. 450, pp. 251–260, 1990, DOI: 10.1007/3-540-52921-7_74.

[6] R. Cole, "Parallel merge sort," *SIAM Journal on Computing*, vol. 17, no. 4, pp. 770–785, 1988, DOI: 10.1137/0217049.

[7] L. Gubias, "Sorting unsorted and partially sorted lists using the natural merge sort," *Software Practice and Experience*, vol. 11, no. 12, pp. 1339–1340, 2006, DOI: 10.1002/spe.4380111211.

[8] B. Huang and M. Langston, "Practical in-place merging," *Communications of ACM*, vol. 31, no. 3, pp. 348–352, 1998, DOI: 10.1002/spe.4380111211.

[9] B. Huang and M. Langston, "Merging sorted runs using main memory," *Acta Informatica*, vol. 27, no. 3, pp. 195–215, 1989, DOI: 10.1007/BF00572988.

[10] M. Lutz, L. Wegner, and J. Teuhola, "The external heap sort," *IEEE Transactions on Software Engineering*, vol. 15, no. 7, pp. 917–925, 1989, DOI: 0098-5589/89/0700-0917.

[11] A. Rauh and G. Arce, "A fast weighted median algorithm based on quick select," in *Proceedings of the IEEE 17th International Conference on Image Processing*. 26-29 September, 2010, Hong Kong: IEEE, 2010, pp. 105–108.

[12] R. Damaševičius, M. Vasiljevas, J. Salkevicius, and M. Woźniak, "Human activity recognition in aal environments using random projections," *Computational and Mathematical Methods in Medicine*, vol. 2016, pp. 4 073 584:1–4 073 584:17, 2016, DOI: 10.1155/2016/4073584.

[13] M. Woźniak, M. Gabryel, R. K. Nowicki, and B. Nowak, "An application of firefly algorithm to position traffic in nosql database systems," *Advances in Intelligent Systems and Computing - KICSS'2014*, vol. 416, pp. 259–272, 2016, DOI: 10.1007/978-3-319-27478-2_18.

[14] R. K. Nowicki, B. Nowak, and M. Woźniak, "Application of rough sets in k nearest neighbours algorithm for classification of incomplete samples," *Advances in Intelligent Systems and Computing - KICSS'2014*, vol. 416, pp. 243–257, 2016, DOI: 10.1007/978-3-319-27478-2_17.

[15] D. Połap, M. Woźniak, C. Napoli, and E. Tramontana, "Is swarm intelligence able to create mazes?" *International Journal of Electronics and Telecommunications*, vol. 61, no. 4, pp. 305–310, 2015, DOI: 10.1515/eletel-2015-0039.

[16] D. Połap, M. Woźniak, C. Napoli, and E. Tramontana, "Real-time cloud-based game management system via cuckoo search algorithm," *International Journal of Electronics and Telecommunications*, vol. 61, no. 4, pp. 333–338, 2015, DOI: 10.1515/eletel-2015-0043.

[17] Z. Marszałek, G. Woźniak, M. Borowik, R. Wazirali, C. Napoli, G. Pappalardo, and E. Tramontana, "Benchmark tests on improved merge for big data processing," in *Asia-Pacific Conference on Computer Aided System Engineering – APCASE'2015*. 14-16 July, Quito, Ecuador: IEEE, 2015, pp. 96–101, DOI: 10.1109/APCASE.2015.24.

[18] M. Woźniak, Z. Marszałek, M. Gabryel, and R. K. Nowicki, "Preprocessing large data sets by the use of quick sort algorithm," *Advances in Intelligent Systems and Computing: Knowledge, Information and Creativity Support Systems: Recent Trends, Advances and Solutions - KICSS'2013*, vol. 364, pp. 111–121, 2015, DOI: 10.1007/978-3-319-19090-7_9.

[19] M. Woźniak, Z. Marszałek, M. Gabryel, and R. K. Nowicki, "Modified merge sort algorithm for large scale data sets," *Lecture Notes in Artificial Intelligence - ICAISC'2013*, vol. 7895, pp. 612–622, 2013, DOI: 10.1007/978-3-642-38610-7_56.

[20] M. Woźniak, D. Połap, R. K. Nowicki, C. Napoli, G. Pappalardo, and E. Tramontana, "Novel approach toward medical signals classifier," in *IEEE IJCNN 2015 - 2015 IEEE International Joint Conference on Neural Networks, Proceedings*. 12-17 July, Killarney, Ireland: IEEE, 2015, pp. 1924–1930, DOI: 10.1109/IJCNN.2015.7280556.

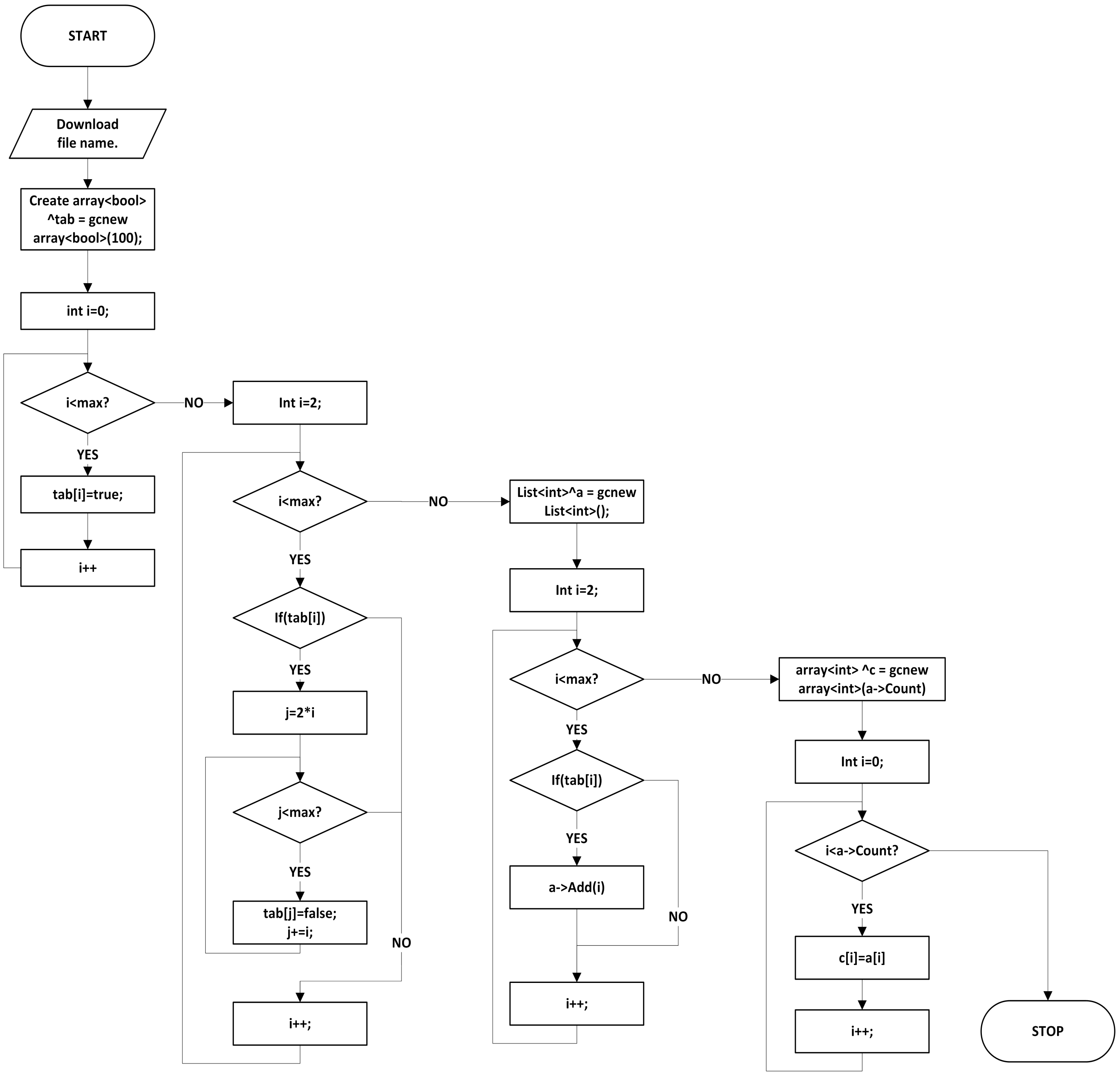

Figure 11. Block Diagram of the Program 'Nested loops'

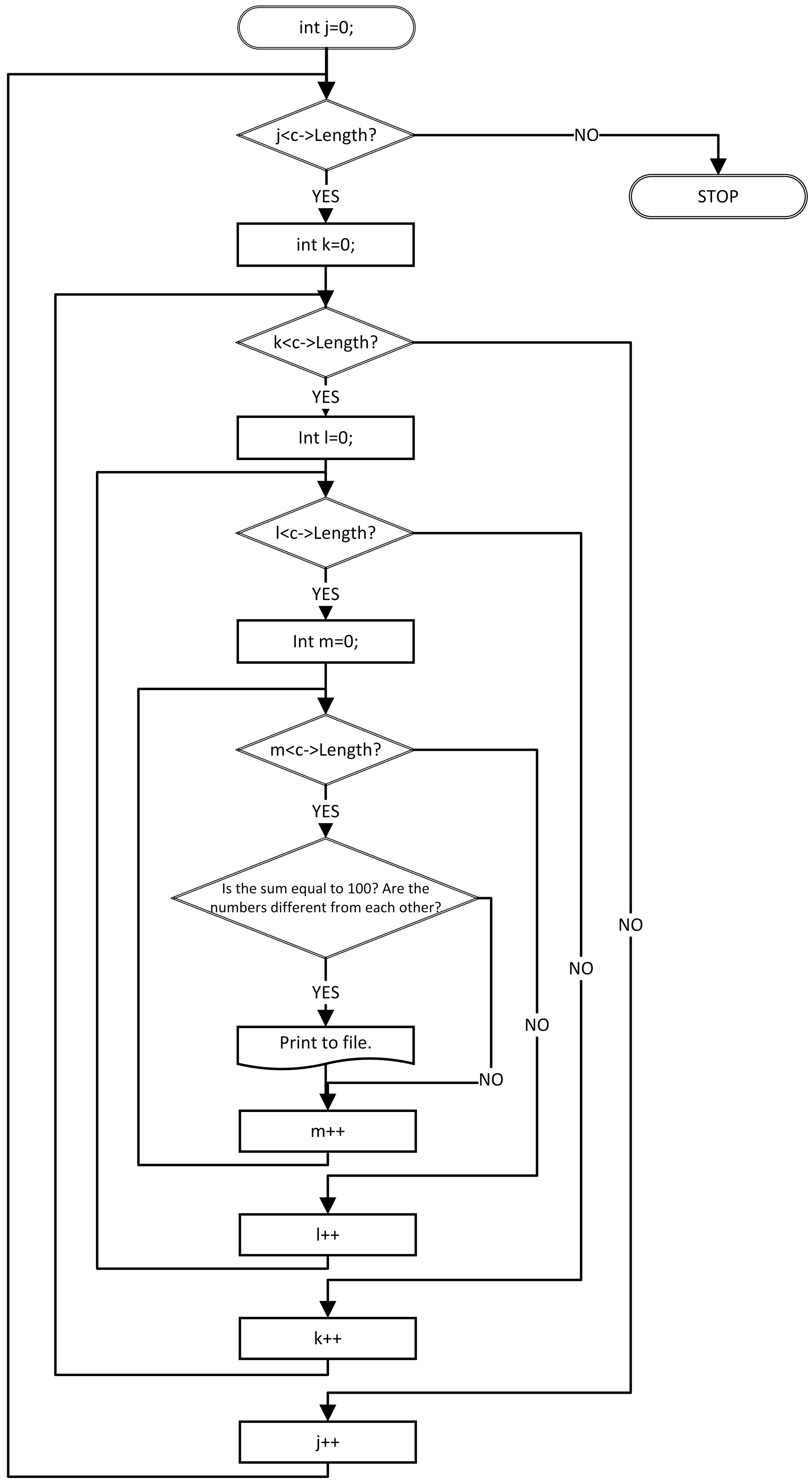

Figure 12. Example of loops for 4-elements combinations.