4. Write pseudocode for an algorithm for finding real roots of equation $ax^2 + bx + c = 0$ for arbitrary real coefficients a, b, and c. (You may assume the availability of the square root function $sqrt(x)$.)

6. Describe the algorithm used by your favorite ATM machine in dispensing cash. (You may give your description in either English or pseudocode, whichever you find more convenient.)

1. For each of the following algorithms, indicate (i) a natural size metric for its inputs, (ii) its basic operation, and (iii) whether the basic operation count can be different for inputs of the same size:

   a. computing the sum of n numbers

   b. computing $n!$

   c. finding the largest element in a list of n numbers

   d. Euclid's algorithm

   e. sieve of Eratosthenes

   f. pen-and-pencil algorithm for multiplying two n-digit decimal integers

2. a. Consider the definition-based algorithm for adding two $n \times n$ matrices. What is its basic operation? How many times is it performed as a function of the matrix order n? As a function of the total number of elements in the input matrices?

   b. Answer the same questions for the definition-based algorithm for matrix multiplication.

3. Consider a variation of sequential search that scans a list to return the number of occurrences of a given search key in the list. Will its efficiency differ from the efficiency of classic sequential search?

4. a. *Glove selection* There are 22 gloves in a drawer: 5 pairs of red gloves, 4 pairs of yellow, and 2 pairs of green. You select the gloves in the dark and can check them only after a selection has been made. What is the smallest number of gloves you need to select to have at least one matching pair in the best case? in the worst case?

5. a.▷ Prove formula (2.1) for the number of bits in the binary representation of a positive integer.

$$b = \lfloor \log_2 n \rfloor + 1.$$

b.▷ Prove the alternative formula for the number of bits in the binary representation of a positive integer n:

$$b = \lceil \log_2 (n+1) \rceil.$$

c. What would be the analogous formulas for the number of decimal digits?

1. Algorithm Complexity Analysis

We refer the reader to the lecture notes for explanations of classical order of growth functions, the count step method and the time measuring method to apply in the following tasks.

```
int SecondFunction(const int n)
{
 int sum=0;
 for (int i=1; i<n; i*=2)
    { for (j=n; j>0 ; j/=2)
        { for (k=j; k<n ; k+=2)
            { sum+=2*j; }
        }
    }

 return sum;
}
```

Setup.

1. Create a basic C++ project in Visual Studio 2022. Create a project by pointing to New on the File menu, and then clicking Project. In the Visual C++ project type panel, click Windows Desktop, and then click Windows Console Application. Type a name for the project. Then, click OK to create the project. Finally, create a new main.cpp file.

2. For the C++ function detailed above, perform the tasks ”Counting Steps Method” and ”Time Measurement Method” details below.

Counting Steps Method. Apply the Counting Steps Method to measure experimentally the order of growth (as a function of n) of the running algorithms:

3. Implement the provided function. By looking at the code, what is intuitively the order of growth of this function?

4. Modify the original function to count elementary operations.

5. Run the code for n = 1, 10, 100, 1000, 10000. Report the total number of steps counted in function of n.

6. Plot the curve of the number of steps taken by the provided function (in function of n). Identify the tightest upper bound well-known classical functions. What is the worst-case time complexity in big-O notation (with respect to the size of the input) obtained from the Counting Steps Method?

7. State the order of growth (as a function of N) of the running times of the function.

Time Measurement Method. Apply the Time Measurement Method to measure experimentally the time consumed (as a function of n) of the running algorithm:

8. Modify the original function to measure the execution running time of the overall function.

9. Run the code for n = 1, 10, 100, 1000, 10000. Report the total time measured in function of n.

10. Plot the curve of the running time of the provided function (in function of n). Identify the tightest upper bound well-known classical functions.

11. What is the worst-case time complexity in big-O notation (with respect to the size of the input) obtained from the Time Measurement Method?

Observations. Perform an analysis of the obtained experimental results.

12. For the function, compare the resulting complexity obtained by the Counting Steps and the Time Measurement Method.

13. Explain the differences or similarities observed between both strategies.