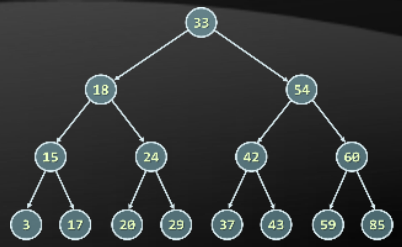
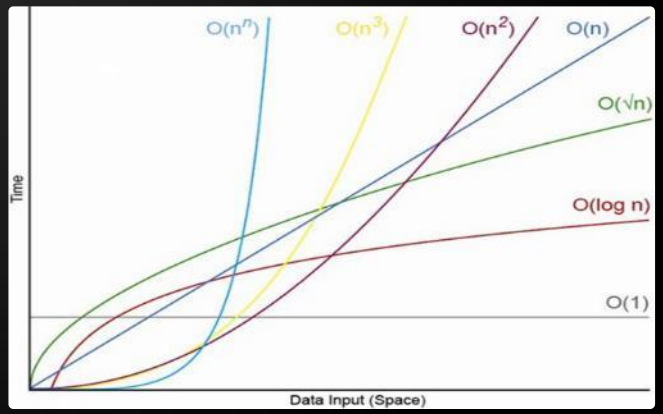
Telerik Academy



Data Structures, Algorithms and Complexity

Overview of Data Structures and Basic Algorithms. Computational Complexity. Asymptotic Notation



Data structures and algorithms

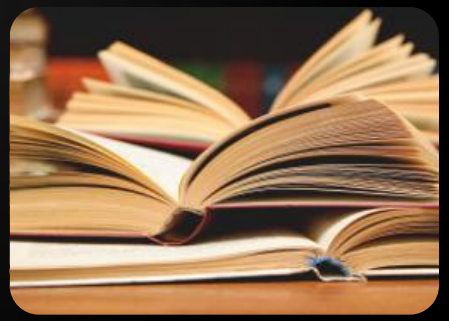
Telerik Software Academy

http://academy.telerik.com

Telerik Academy

Table of Contents

1. Data Structures Overview
   - Linear Structures, Trees, Hash Tables, Others
2. Algorithms Overview
   - Sorting and Searching, Combinatorics, Dynamic Programming, Graphs, Others
3. Algorithms Complexity
   - Time and Memory Complexity
   - Mean, Average and Worst Case
   - Asymptotic Notation O(g)



Telerik Academy

Data Structures

Introduction

What is a Data Structure?

> "In computer science, a data structure is a particular way of storing and organizing data in a computer so that it can be used efficiently."
>
> *-- Wikipedia*

- Examples of data structures:
  - `Person` structure (first name + last name + age)
  - Array of integers – `int[]`
  - List of strings – `List<string>`
  - Queue of people – `Queue<Person>`

Why Are Data Structures So Important?

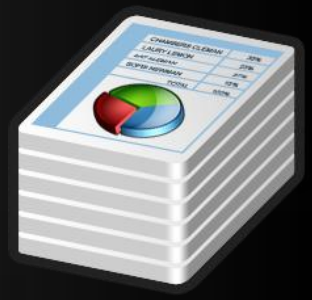
- Data structures and algorithms are the foundation of computer programming
- Algorithmic thinking, problem solving and data structures are vital for software engineers
  - All .NET developers should know when to use `T[]`, `LinkedList<T>`, `List<T>`, `Stack<T>`, `Queue<T>`, `Dictionary<K,T>`, `HashSet<T>`, `SortedDictionary<K,T>` and `SortedSet<T>`
- Computational complexity is important for algorithm design and efficient programming

Telerik Academy

Primitive Types and Collections

- Primitive data types
  - Numbers: `int`, `float`, `decimal`, …
  - Text data: `char`, `string`, …

- Simple structures
  - A group of fields stored together
  - E.g. `DateTime`, `Point`, `Rectangle`, …



- Collections
  - A set of elements (of the same type)
  - E.g. array, list, stack, tree, hash-table, …

Abstract Data Types (ADT)

- An Abstract Data Type (ADT) is
  - A data type together with the operations, whose properties are specified independently of any particular implementation
  - ADT are set of definitions of operations
    - Like the interfaces in C#
- ADT can have multiple different implementations

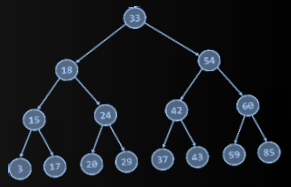
  - Different implementations can have different efficiency, inner logic and resource needs

Telerik Academy

Basic Data Structures

- Linear structures

  - Lists: fixed size and variable size
  - Stacks: LIFO (Last In First Out) structure
  - Queues: FIFO (First In First Out) structure
- Trees and tree-like structures



  - Binary, ordered search trees, balanced, etc.
- Dictionaries (maps)

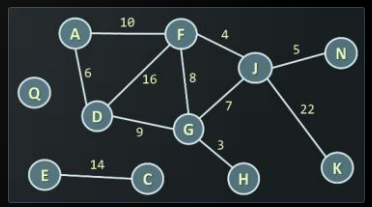
  - Contain pairs (key, value)
  - Hash tables: use hash functions to search/insert

Basic Data Structures (2)

- Sets and bags
  - Set – collection of unique elements
  - Bag – collection of non-unique elements

- Ordered sets, bags and dictionaries
- Priority queues / heaps
- Special tree structures
  - Suffix tree, interval tree, index tree, trie

- Graphs
  - Directed / undirected, weighted / un-weighted, connected/ non-connected, …

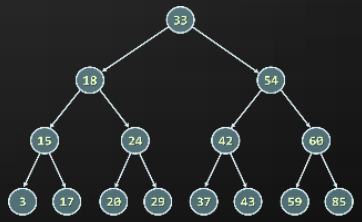


Telerik Academy

Algorithms

Introduction

```
DFS(node)
{
  for each child c of node
    DFS(c);
  print the current node;
}
```




```
DFS(node)
{
  stack ← node
  visited[node] = true
  while stack not empty
    v ← stack
    print v
    for each child c of v
      if not visited[c]
        stack ← c
        visited[c] = true
}
```

What is an Algorithm?

> "In mathematics and computer science, an algorithm is a step-by-step procedure for calculations. An algorithm is an effective method expressed as a finite list of well-defined instructions for calculating a function."
>
> *-- Wikipedia*

- The term "algorithm" comes from the
  - Derived from Muḥammad Al-Khwārizmī, a Persian mathematician and astronomer
    - An algorithm for solving quadratic equations

Telerik Academy

Algorithms in Computer Science

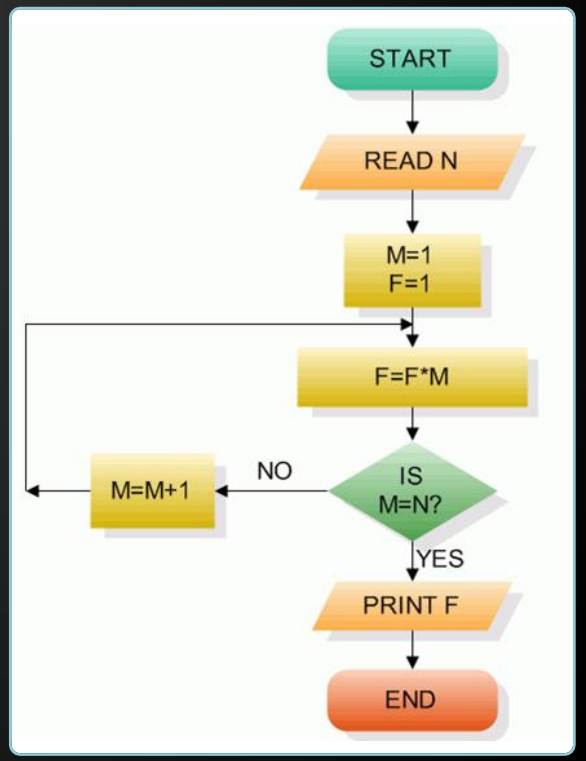
- Algorithms are fundamental in programming
  - Imperative (traditional) programming means to describe in formal steps how to do something
  - Algorithm == sequence of operations (steps)
    - Can include branches (conditional blocks) and repeated logic (loops)
- Algorithmic thinking (mathematical thinking, logical thinking, engineering thinking)
  - Ability to decompose the problems into formal sequences of steps (algorithms)

Pseudocode and Flowcharts

- Algorithms can be expressed in pseudocode, through flowcharts or program code

```
BFS(node)
{
  queue ← node
  while queue not empty
    v ← queue
    print v
    for each child c of v
      queue ← c
}
```

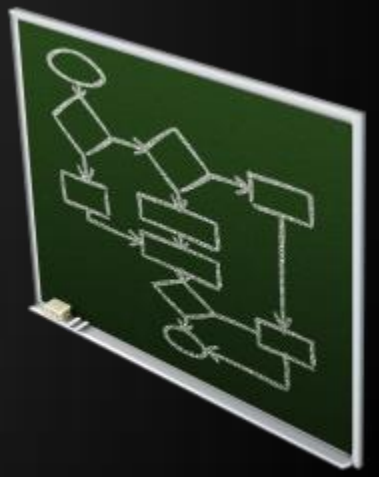
Example of pseudocode



Example of flowchart

Algorithms in Programming

- Sorting and searching
- Dynamic programming
- Graph algorithms
  - DFS and BFS traversals
- Combinatorial algorithms
  - Recursive algorithms
- Other algorithms
  - Greedy algorithms, computational geometry, randomized algorithms, genetic algorithms



Telerik Academy

Algorithm Complexity

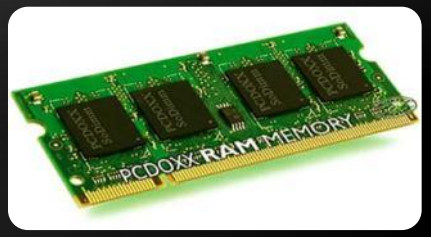
Asymtotic Notation

Algorithm Analysis

- Why we should analyze algorithms?
  - Predict the resources the algorithm requires
    - Computational time (CPU consumption)
    - Memory space (RAM consumption)
    - Communication bandwidth consumption
  - The running time of an algorithm is:
    - The total number of primitive operations executed (machine independent steps)
    - Also known as algorithm complexity

Algorithmic Complexity

- What to measure?
  - CPU Time
  - Memory
  - Number of steps
  - Number of particular operations
    - Number of disk operations
    - Number of network packets
  - Asymptotic complexity

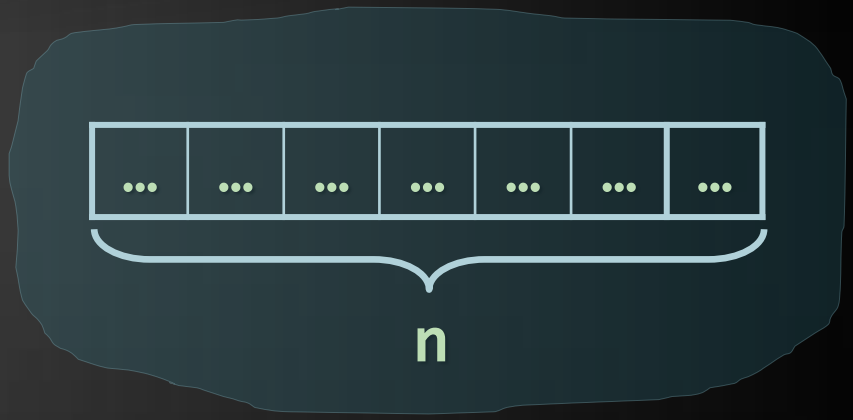


Time Complexity

- Worst-case
  - An upper bound on the running time for any input of given size
- Average-case
  - Assume all inputs of a given size are equally likely
- Best-case
  - The lower bound on the running time (the optimal case)

Time Complexity – Example

- Sequential search in a list of size n
  - Worst-case:
    - n comparisons
  - Best-case:
    - 1 comparison
  - Average-case:
    - n/2 comparisons
- The algorithm runs in linear time
  - Linear number of operations



Algorithms Complexity

- Algorithm complexity is a rough estimation of the number of steps performed by given computation depending on the size of the input data
  - Measured through asymptotic notation
    - $O(g)$ where g is a function of the input data size
  - Examples:
    - Linear complexity $O(n)$ – all elements are processed once (or constant number of times)
    - Quadratic complexity $O(n^2)$ – each of the elements is processed n times

Asymptotic Notation: Definition

- Asymptotic upper bound
  - O-notation (Big O notation)
- For given function g(n), we denote by O(g(n)) the set of functions that are different than g(n) by a constant

> O(g(n)) = {f(n): there exist positive constants c and n_0 such that f(n) <= c*g(n) for all n >= n_0}

- Examples:
  - $3 * n^2 + n/2 + 12 \in O(n^2)$
  - $4*n*\log_2(3*n+1) + 2*n-1 \in O(n * \log n)$

Typical Complexities

Complexity	Notation	Description
constant	O(1)	Constant number of operations, not depending on the input data size, e.g. n = 1 000 000 → 1-2 operations
logarithmic	O(log n)	Number of operations proportional of $\log_2(n)$ where n is the size of the input data, e.g. n = 1 000 000 000 → 30 operations
linear	O(n)	Number of operations proportional to the input data size, e.g. n = 10 000 → 5 000 operations

Typical Complexities (2)

Complexity	Notation	Description
quadratic	$O(n^2)$	Number of operations proportional to the square of the size of the input data, e.g. n = 500 → 250 000 operations
cubic	$O(n^3)$	Number of operations proportional to the cube of the size of the input data, e.g. n = 200 → 8 000 000 operations
exponential	$O(2^n)$, $O(k^n)$, $O(n!)$	Exponential number of operations, fast growing, e.g. n = 20 → 1 048 576 operations

Telerik Academy

Time Complexity and Speed

Complexity	10	20	50	100	1 000	10 000	100 000
O(1)	< 1 s	< 1 s	< 1 s	< 1 s	< 1 s	< 1 s	< 1 s
O(log(n))	< 1 s	< 1 s	< 1 s	< 1 s	< 1 s	< 1 s	< 1 s
O(n)	< 1 s	< 1 s	< 1 s	< 1 s	< 1 s	< 1 s	< 1 s
O(n*log(n))	< 1 s	< 1 s	< 1 s	< 1 s	< 1 s	< 1 s	< 1 s
$O(n^2)$	< 1 s	< 1 s	< 1 s	< 1 s	< 1 s	2 s	3-4 min
$O(n^3)$	< 1 s	< 1 s	< 1 s	< 1 s	20 s	5 hours	231 days
$O(2^n)$	< 1 s	< 1 s	260 days	hangs	hangs	hangs	hangs
O(n!)	< 1 s	hangs	hangs	hangs	hangs	hangs	hangs
$O(n^n)$	3-4 min	hangs	hangs	hangs	hangs	hangs	hangs

Time and Memory Complexity

- Complexity can be expressed as formula on multiple variables, e.g.
  - Algorithm filling a matrix of size `n * m` with the natural numbers 1, 2, … will run in `O(n*m)`
  - A traversal of graph with `n` vertices and `m` edges will run in `O(n + m)`
- Memory consumption should also be considered, for example:
  - Running time `O(n)` & memory requirement $O(n^2)$
  - `n = 50 000` → `OutOfMemoryException`

The Hidden Constant

- Sometimes a linear algorithm could be slower than quadratic algorithm
  - The hidden constant could be significant
- Example:
  - Algorithm A makes: `100*n` steps → $O(n)$
  - Algorithm B makes: `n*n/2` steps → $O(n^2)$
  - For `n < 200` the algorithm B is faster
- Real-world example:
  - Insertion sort is faster than quicksort for `n < 9`

Polynomial Algorithms

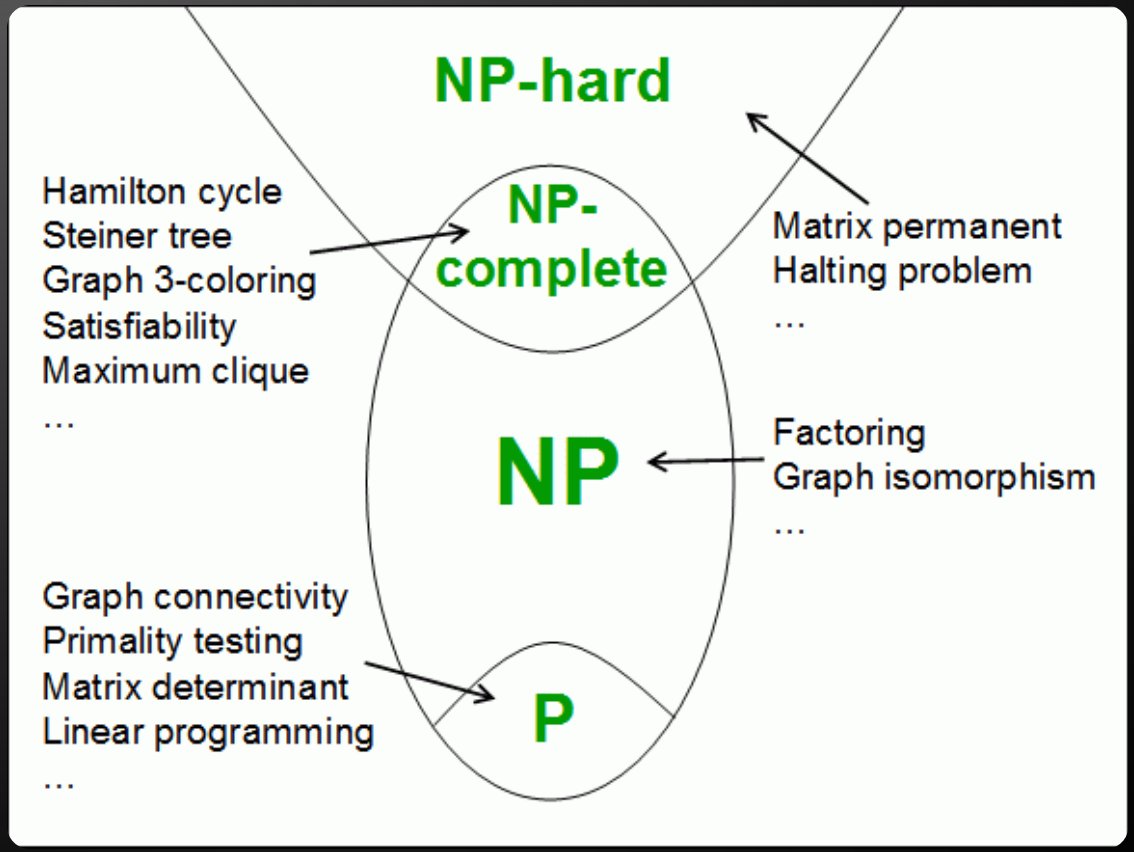
- A polynomial-time algorithm is one whose worst-case time complexity is bounded above by a polynomial function of its input size

$$W(n) \in O(p(n))$$

- Examples:
  - Polynomial-time: log n, 2n, $3n^3 + 4n$, 2 * n log n
  - Non polynomial-time : 2^n, 3^n, n^k, n!
- Non-polynomial algorithms hang for large input data sets

Computational Classes

- Computational complexity theory divides the computational problems into several classes:



Telerik Academy

Analyzing Complexity of Algorithms

Examples

Complexity Examples

```
int FindMaxElement(int[] array)
{
    int max = array[0];
    for (int i=0; i<array.length; i++)
    {
        if (array[i] > max)
        {
            max = array[i];
        }
    }
    return max;
}
```

- Runs in O(n) where n is the size of the array
- The number of elementary steps is ~ n

Complexity Examples (2)

```
long FindInversions(int[] array)
{
    long inversions = 0;
    for (int i=0; i<array.Length; i++)
        for (int j = i+1; j<array.Length; i++)
            if (array[i] > array[j])
                inversions++;
    return inversions;
}
```

- Runs in $O(n^2)$ where n is the size of the array
- The number of elementary steps is ~ n*(n+1) / 2

Complexity Examples (3)

```
decimal Sum3(int n)
{
    decimal sum = 0;
    for (int a=0; a<n; a++)
        for (int b=0; b<n; b++)
            for (int c=0; c<n; c++)
                sum += a*b*c;
    return sum;
}
```

- Runs in cubic time $O(n^3)$
- The number of elementary steps is $\sim n^3$

Complexity Examples (4)

```
long SumMN(int n, int m)
{
    long sum = 0;
    for (int x=0; x<n; x++)
        for (int y=0; y<m; y++)
            sum += x*y;
    return sum;
}
```

- Runs in quadratic time O(n*m)
- The number of elementary steps is ~ n*m

Complexity Examples (5)

```
long SumMN(int n, int m)
{
    long sum = 0;
    for (int x=0; x<n; x++)
        for (int y=0; y<m; y++)
            if (x==y)
                for (int i=0; i<n; i++)
                    sum += i*x*y;
    return sum;
}
```

- Runs in quadratic time O(n*m)
- The number of elementary steps is ~ n*m + min(m,n)*n

Complexity Examples (6)

```
decimal Calculation(int n)
{
    decimal result = 0;
    for (int i = 0; i < (1<<n); i++)
        result += i;
    return result;
}
```

- Runs in exponential time $O(2^n)$
- The number of elementary steps is ~ 2^n

Complexity Examples (7)

```
decimal Factorial(int n)
{
    if (n==0)
        return 1;
    else
        return n * Factorial(n-1);
}
```

- Runs in linear time O(n)
- The number of elementary steps is ~ n

Complexity Examples (8)

```
decimal Fibonacci(int n)
{
    if (n == 0)
        return 1;
    else if (n == 1)
        return 1;
    else
        return Fibonacci(n-1) + Fibonacci(n-2);
}
```

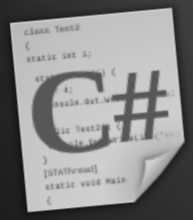
- Runs in exponential time $O(2^n)$
- The number of elementary steps is ~ `Fib(n+1)` where `Fib(k)` is the k-th Fibonacci's number

Summary

- Data structures organize data for efficient use
  - ADT describe a set of operations
  - Collections hold a group of elements
- Algorithms are sequences of steps for performing or calculating something
- Algorithm complexity is rough estimation of the number of steps performed by given computation
  - Complexity can be logarithmic, linear, n log n, square, cubic, exponential, etc.
  - Allows to estimating the speed of given code before its execution

Telerik Academy

Data Structures, Algorithms and Complexity



Questions?

```
DFS(node)
{
  stack ← node
  visited[node] = true
  while stack not empty
    v ← stack
    print v
    for each child c of v
      if not visited[c]
        stack ← c
        visited[c] = true
}
```

```
DFS(node)
{
  for each child c of node
    DFS(c);
  print the current node;
}
```

http://academy.telerik.com

Exercises

1. What is the expected running time of the following C# code? Explain why. Assume the array's size is n.

```
long Compute(int[] arr)
{
    long count = 0;
    for (int i=0; i<arr.Length; i++)
    {
        int start = 0, end = arr.Length-1;
        while (start < end)
            if (arr[start] < arr[end])
                { start++; count++; }
            else
                end--;
    }
    return count;
}
```

Exercises (2)

2. What is the expected running time of the following C# code? Explain why.

```
long CalcCount(int[,] matrix)
{
    long count = 0;
    for (int row=0; row<matrix.GetLength(0); row++)
        if (matrix[row, 0] % 2 == 0)
            for (int col=0; col<matrix.GetLength(1); col++)
                if (matrix[row,col] > 0)
                    count++;
    return count;
}
```

Assume the input matrix has size of n * m.

Exercises (3)

3. * What is the expected running time of the following C# code? Explain why.

```
long CalcSum(int[,] matrix, int row)
{
    long sum = 0;
    for (int col = 0; col < matrix.GetLength(0); col++)
        sum += matrix[row, col];
    if (row + 1 < matrix.GetLength(1))
        sum += CalcSum(matrix, row + 1);
    return sum;
}

Console.WriteLine(CalcSum(matrix, 0));
```

Assume the input matrix has size of n * m.

Telerik Academy

Free Trainings @ Telerik Academy

- C# Programming @ Telerik Academy
  - csharpfundamentals.telerik.com

- Telerik Software Academy
  - academy.telerik.com

- Telerik Academy @ Facebook
  - facebook.com/TelerikAcademy

- Telerik Software Academy Forums
  - forums.academy.telerik.com