# Introduction

## 0.1.2. A Philosophy of Data Structures

Our method for evaluating the efficiency of an algorithm or computer program is called [**asymptotic analysis**](https://traky.cs.hut.fi/Books/CSE-A1141/html/Glossary.html#term-asymptotic-analysis). Asymptotic analysis also gives a way to define the inherent difficulty of a problem.

A method for estimating the efficiency of an algorithm or computer program by identifying its [**growth rate**](https://traky.cs.hut.fi/Books/CSE-A1141/html/Glossary.html#term-growth-rate). Asymptotic analysis also gives a way to define the inherent difficulty of a [**problem**](https://traky.cs.hut.fi/Books/CSE-A1141/html/Glossary.html#term-problem). We frequently use the term [**algorithm analysis**](https://traky.cs.hut.fi/Books/CSE-A1141/html/Glossary.html#term-algorithm-analysis) to mean the same thing.

In the most general sense, a [**data structure**](https://traky.cs.hut.fi/Books/CSE-A1141/html/Glossary.html#term-data-structure) is any data representation and its associated operations. Even an integer or floating point number stored on the computer can be viewed as a simple data structure. More commonly, people use the term "data structure" to mean an organization or structuring for a collection of data items. A sorted list of integers stored in an array is an example of such a structuring. These ideas are explored further in a discussion of [**Abstract Data Types**](https://traky.cs.hut.fi/Books/CSE-A1141/html/ADT.html#adt).

Data structure: The implementation for an [**ADT**](https://traky.cs.hut.fi/Books/CSE-A1141/html/Glossary.html#term-adt).

Given sufficient space to store a collection of [**data items**](https://traky.cs.hut.fi/Books/CSE-A1141/html/Glossary.html#term-data-item), it is always possible to search for specified items within the collection, print or otherwise process the data items in any desired order, or modify the value of any particular data item. searching for a given record in a [**hash table**](https://traky.cs.hut.fi/Books/CSE-A1141/html/Glossary.html#term-hash-table) is much faster than searching for it in an unsorted array.

A solution is said to be [**efficient**](https://traky.cs.hut.fi/Books/CSE-A1141/html/Glossary.html#term-efficient) if it solves the problem within the required [**resource constraints**](https://traky.cs.hut.fi/Books/CSE-A1141/html/Glossary.html#term-resource-constraints). Examples of resource constraints include the total space available to store the data—possibly divided into separate main memory and disk space constraints—and the time allowed to perform each subtask. A solution is sometimes said to be efficient if it requires fewer resources than known alternatives, regardless of whether it meets any particular requirements. The [**cost**](https://traky.cs.hut.fi/Books/CSE-A1141/html/Glossary.html#term-cost) of a solution is the amount of resources that the solution consumes. Most often, cost is measured in terms of one key resource such as time, with the implied assumption that the solution meets the other resource constraints.

Examples of resource constraints include the total space available to store the data—possibly divided into separate main memory and disk space constraints—and the time allowed to perform each subtask.

## 0.1.3. Selecting a Data Structure

Only by first analyzing the problem to determine the performance goals that must be achieved can there be any hope of selecting the right data structure for the job. Poor program designers ignore this analysis step and apply a data structure that they are familiar with but which is inappropriate to the problem. Conversely, there is no sense in adopting a complex representation to "improve" a program that can meet its performance goals when implemented using a simpler design.

Problem: A task to be performed. It is best thought of as a [**function**](https://traky.cs.hut.fi/Books/CSE-A1141/html/Glossary.html#term-function) or a mapping of inputs to outputs.

When selecting a data structure to solve a problem, you should follow these steps.

1. Analyze your problem to determine the [***basic operations***](https://traky.cs.hut.fi/Books/CSE-A1141/html/Glossary.html#term-basic-operations) that must be supported. Examples of basic operations include inserting a data item into the data structure, deleting a data item from the data structure, and finding a specified data item.
2. Quantify the resource constraints for each operation.
3. Select the data structure that best meets these requirements.

This three-step approach to selecting a data structure operationalizes a data-centered view of the design process. The first concern is for the data and the operations to be performed on them, the next concern is the representation for those data, and the final concern is the implementation of that representation.

Resource constraints on certain key operations, such as search, inserting data records, and deleting data records, normally drive the data structure selection process.

1. Are all data items inserted into the data structure at the beginning, or are insertions interspersed with other operations? Static applications (where the data are loaded at the beginning and never change) typically get by with simpler data structures to get an efficient implementation, while dynamic applications often require something more complicated.
2. Can data items be deleted? If so, this will probably make the implementation more complicated.
3. Are all data items processed in some well-defined order, or is search for specific data items allowed? "Random access" search generally requires more complex data structures.

Each data structure has associated costs and benefits. In practice, it is hardly ever true that one data structure is better than another for use in all situations. If one data structure or algorithm is superior to another in all respects, the inferior one will usually have long been forgotten.

A data structure requires a certain amount of space for each data item it stores, a certain amount of time to perform a single basic operation, and a certain amount of programming effort. Each problem has constraints on available space and time.

Example 0.1

When considering the choice of data structure to use in the database system that manages customer accounts, we see that a data structure that has little concern for the cost of deletion, but is highly efficient for search and moderately efficient for insertion, should meet the resource constraints imposed by this problem. Records are accessible by unique account number (sometimes called an [**exact-match query**](https://traky.cs.hut.fi/Books/CSE-A1141/html/Glossary.html#term-exact-match-query)). One data structure that meets these requirements is the [**hash table**](https://traky.cs.hut.fi/Books/CSE-A1141/html/HashIntro.html#hashintro). Hash tables allow for extremely fast exact-match search. A record can be modified quickly when the modification does not affect its space requirements. Hash tables also support efficient insertion of new records. While deletions can also be supported efficiently, too many deletions lead to some degradation in performance for the remaining operations. However, the hash table can be reorganized periodically to restore the system to peak efficiency. Such reorganization can occur offline so as not to affect ATM transactions.

当考虑在管理客户帐户的数据库系统中使用数据结构的选择时，我们看到一个数据结构对于删除的成本几乎不关心，但是对于搜索和中等效率的插入来说是高效的，应该满足资源这个问题施加的约束。记录可通过唯一的帐号（有时称为完全匹配查询）访问。满足这些要求的一个数据结构是哈希表。哈希表允许极快的精确匹配搜索。当修改不影响其空间要求时，可以快速修改记录。哈希表还支持高效插入新记录。虽然还可以有效地支持删除，但是删除太多会导致其余操作的性能下降。然而，可以定期重组散列表，以将系统恢复到峰值效率。这种重组可以脱机发生，以免影响ATM交易。

**Example 0.1.2**

A company is developing a database system containing information about cities and towns in the United States. There are many thousands of cities and towns, and the database program should allow users to find information about a particular place by name (another example of an exact-match query). Users should also be able to find all places that match a particular value or range of values for attributes such as location or population size. This is known as a [***range query***](https://traky.cs.hut.fi/Books/CSE-A1141/html/Glossary.html#term-range-query).

A reasonable database system must answer queries quickly enough to satisfy the patience of a typical user. For an exact-match query, a few seconds is satisfactory. If the database is meant to support range queries that can return many cities that match the query specification, the user might tolerate the entire operation to take longer, perhaps on the order of a minute. To meet this requirement, it will be necessary to support operations that process range queries efficiently by processing all cities in the range as a batch, rather than as a series of operations on individual cities.

合理的数据库系统必须足够快地回答查询以满足典型用户的耐心。 对于精确匹配查询，几秒钟是令人满意的。 如果数据库旨在支持可以返回与查询规范匹配的许多城市的范围查询，则用户可能会容忍整个操作花费更长时间，也许在一分钟之内。 为了满足这一要求，有必要通过以一批的方式处理范围内的所有城市，而不是在各个城市的一系列操作来有效地支持处理范围查询的操作。

The [**B+-tree**](https://traky.cs.hut.fi/Books/CSE-A1141/html/BTree.html#btree) supports large databases, insertion and deletion of data records, and range queries. However, a simple [**linear index**](https://traky.cs.hut.fi/Books/CSE-A1141/html/LinearIndexing.html#linearindexing) would be more appropriate if the database is created once, and then never changed, such as an atlas distributed on a CD or accessed from a website.

## 0.2 Abstract Data Types[¶](https://traky.cs.hut.fi/Books/CSE-A1141/html/ADT.html#abstract-data-types)

A [**type**](https://traky.cs.hut.fi/Books/CSE-A1141/html/Glossary.html#term-type) is a collection of values. For example, the Boolean type consists of the values true and false. The integers also form a type. An integer is a [**simple type**](https://traky.cs.hut.fi/Books/CSE-A1141/html/Glossary.html#term-simple-type) because its values contain no subparts. A bank account record will typically contain several pieces of information such as name, address, account number, and account balance. Such a record is an example of an [**aggregate type**](https://traky.cs.hut.fi/Books/CSE-A1141/html/Glossary.html#term-aggregate-type) or [**composite type**](https://traky.cs.hut.fi/Books/CSE-A1141/html/Glossary.html#term-composite-type). A [**data item**](https://traky.cs.hut.fi/Books/CSE-A1141/html/Glossary.html#term-data-item) is a piece of information or a record whose value is drawn from a type. A data item is said to be a [**member**](https://traky.cs.hut.fi/Books/CSE-A1141/html/Glossary.html#term-member) of a type.

A [**data type**](https://traky.cs.hut.fi/Books/CSE-A1141/html/Glossary.html#term-data-type) is a type together with a collection of operations to manipulate the type. For example, an integer variable is a member of the integer data type. Addition is an example of an operation on the integer data type.

A distinction should be made between the logical concept of a data type and its physical implementation in a computer program. For example, there are two traditional implementations for the list data type: the linked list and the array-based list. The list data type can therefore be implemented using a linked list or an array. But we don't need to know how the list is implemented when we wish to use a list to help in a more complex design. For example, a list might be used to help implement a [**graph data structure**](https://traky.cs.hut.fi/Books/CSE-A1141/html/GraphImpl.html#graphimpl).

An [**abstract data type**](https://traky.cs.hut.fi/Books/CSE-A1141/html/Glossary.html#term-abstract-data-type) (ADT) is the realization of a data type as a software component. An ADT is defined in terms of a type and a set of operations on that type. The behavior of each operation is determined by its inputs and outputs. An ADT does not specify how the data type is implemented. These implementation details are hidden from the user of the ADT and protected from outside access, a concept referred to as [**encapsulation**](https://traky.cs.hut.fi/Books/CSE-A1141/html/Glossary.html#term-encapsulation).

**encapsulation**

In programming, the concept of hiding implementation details from the user of an ADT, and protecting [***data members***](https://traky.cs.hut.fi/Books/CSE-A1141/html/Glossary.html#term-data-member) of an object from outside access.

A [**data structure**](https://traky.cs.hut.fi/Books/CSE-A1141/html/Glossary.html#term-data-structure) is the implementation for an ADT. In an object-oriented language, an ADT and its implementation together make up a [**class**](https://traky.cs.hut.fi/Books/CSE-A1141/html/Glossary.html#term-class). Each operation associated with the ADT is implemented by a [**member function**](https://traky.cs.hut.fi/Books/CSE-A1141/html/Glossary.html#term-member-function) or [**method**](https://traky.cs.hut.fi/Books/CSE-A1141/html/Glossary.html#term-method). The variables that define the space required by a data item are referred to as [**data members**](https://traky.cs.hut.fi/Books/CSE-A1141/html/Glossary.html#term-data-member). An [**object**](https://traky.cs.hut.fi/Books/CSE-A1141/html/Glossary.html#term-object) is an instance of a class, that is, something that is created and takes up storage during the execution of a computer program.

The term [**data structure**](https://traky.cs.hut.fi/Books/CSE-A1141/html/Glossary.html#term-data-structure) often refers to data stored in a computer's main memory. The related term [**file structure**](https://traky.cs.hut.fi/Books/CSE-A1141/html/Glossary.html#term-file-structure) often refers to the organization of data on peripheral storage, such as a disk drive or CD.

Example 0.2.1

The mathematical concept of an integer, along with operations that manipulate integers, form a data type. The int variable type is a physical representation of the abstract integer. The int variable type, along with the operations that act on an int variable, form an ADT.