Algorithms and data structures

# Data structures

## .NET types

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Data structure** | **Addition** | **Search** | **Deletion** | **Access by index** | **Usage** |
| Array (T[]) | O(N) | O(N) | O(N) | O(1) | Use when fixed number of elements should be processed by index |
| Linked list (LinkedList<T>) | O(1) | O(N) | O(N) | O(N) | Use when elements should be added and processed by index |
| Dynamic array (List<T>) | O(1) | O(N) | O(N) | O(1) | Use when elements should be added at the both sides of the list. Otherwise use resizable array list (List<T>) |
| Stack (Stack<T>) | O(1) | - | O(1) | - | Use to implement LIFO (last-in-first-out) behavior. List<T> could also work well |
| Queue (Queue<T>) | O(1) | - | O(1) | - | Use to implement FIFO (first-in-first-out) behavior. LinkedList<T> could also work well |
| Dictionary<K, T> | O(1) | O(1) | O(1) | - | Use when key-value pairs should be added fast and searched fast by key. Elements in a hash table have no particular order |
| SortedDictionary<K, T> | O(log(N)) | O(log(N)) | O(log(N)) | - | Use when key-value pairs should be added fast, searched fast by key and enumerated sorted by key |
| HashSet<T> | O(1) | O(1) | O(1) | - | Use to keep a group of unique values, to add and check belonging to the set fast  Elements are in no particular order |
| SortedSet<T> | O(log(N)) | O(log(N)) | O(log(N)) | - | Use to keep a group of ordered unique values |

## General types

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Data Structure** | **Time** | | | | | | | | **Space** |
|  | **Average** | | | | **Worst** | | | | **Worst** |
|  | **Access** | **Search** | **Insertion** | **Deletion** | **Access** | **Search** | **Insertion** | **Deletion** |  |
| [Array](http://en.wikipedia.org/wiki/Array_data_structure) | O(1) | O(n) | O(n) | O(n) | O(1) | O(n) | O(n) | O(n) | O(n) |
| [Stack](http://en.wikipedia.org/wiki/Stack_(abstract_data_type)) | O(n) | O(n) | O(1) | O(1) | O(n) | O(n) | O(1) | O(1) | O(n) |
| [Singly-Linked List](http://en.wikipedia.org/wiki/Singly_linked_list#Singly_linked_lists) | O(n) | O(n) | O(1) | O(1) | O(n) | O(n) | O(1) | O(1) | O(n) |
| [Doubly-Linked List](http://en.wikipedia.org/wiki/Doubly_linked_list) | O(n) | O(n) | O(1) | O(1) | O(n) | O(n) | O(1) | O(1) | O(n) |
| [Skip List](http://en.wikipedia.org/wiki/Skip_list) | O(log(n)) | O(log(n)) | O(log(n)) | O(log(n)) | O(n) | O(n) | O(n) | O(n) | O(n log(n)) |
| [Hash Table](http://en.wikipedia.org/wiki/Hash_table) | - | O(1) | O(1) | O(1) | - | O(n) | O(n) | O(n) | O(n) |
| [Binary Search Tree](http://en.wikipedia.org/wiki/Binary_search_tree) | O(log(n)) | O(log(n)) | O(log(n)) | O(log(n)) | O(n) | O(n) | O(n) | O(n) | O(n) |
| [Cartesian Tree](https://en.wikipedia.org/wiki/Cartesian_tree) | - | O(log(n)) | O(log(n)) | O(log(n)) | - | O(n) | O(n) | O(n) | O(n) |
| [B-Tree](http://en.wikipedia.org/wiki/B_tree) | O(log(n)) | O(log(n)) | O(log(n)) | O(log(n)) | O(log(n)) | O(log(n)) | O(log(n)) | O(log(n)) | O(n) |
| [Red-Black Tree](http://en.wikipedia.org/wiki/Red-black_tree) | O(log(n)) | O(log(n)) | O(log(n)) | O(log(n)) | O(log(n)) | O(log(n)) | O(log(n)) | O(log(n)) | O(n) |
| [Splay Tree](https://en.wikipedia.org/wiki/Splay_tree) | - | O(log(n)) | O(log(n)) | O(log(n)) | - | O(log(n)) | O(log(n)) | O(log(n)) | O(n) |
| [AVL Tree](http://en.wikipedia.org/wiki/AVL_tree) | O(log(n)) | O(log(n)) | O(log(n)) | O(log(n)) | O(log(n)) | O(log(n)) | O(log(n)) | O(log(n)) | O(n) |

## Wintellect data structures

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  | Add | Find | Remove | Index |
| Bag<T> | A bag (multi-set) based on hash-table  Unordered collection (with duplicates)  **T should provide Equals() and GetHashCode()** | O(1) | O(1) | O(1) | - |
| OrderedBag<T> | A bag (multi-set) based on balanced search tree  **T should implement IComparable<T>** | O(log(N)) | O(log(N)) | O(log(N)) | - |
| Set<T> | A set based on hash-table (no duplicates) Like .NET’s HashSet<T> | O(1) | O(1) | O(1) | - |
| OrderedSet<T> | A set based on balanced search tree (red-black)  Like .NET’s SortedSet<T>  Provides fast .Range(from, to) operation | O(log(N)) | O(log(N)) | O(log(N)) | - |
| MultiDictionary<TKey,TValue> | A dictionary (map) implemented by hash-table  Allows duplicates (configurable) Like Dictionary<TKey,List<TValue>> | O(1) | O(1) | O(1) | - |
| OrderedDictionary<TKey,TValue>  OrderedMultiDictionary<TKey,TValue> | A dictionary based on balanced search tree  **Provides fast .Range(from, to) operation** | O(log(N)) | O(log(N)) | O(log(N)) | - |
| Deque<T> | Double-ended queue (deque) |  |  |  |  |
| BigList<T> | Editable sequence of indexed items  Like List<T> but provides  **Fast Insert / Delete operations (at any position)**  **Fast Copy / Concat / Sub-range operations**  Implemented by the data structure "Rope"  Special kind of balanced binary tree: <http://en.wikipedia.org/wiki/Rope_(data_structure)> |  |  |  |  |

Oeis.org

**Пермутация:**  Всяка възможна подредба на дадено множество М [а,б,в,г,д,е]  е пермутация: Например: [в,д,е,а,б,г] е пермуатация на М. (една от всичките възможни) . Ротация се нарича когато елементите алтернативно се изместват наляво или на дясно (което няма значение защото всяка ротация може да бъде достигната с ляво и дясно въртене) . Например ротацията [е,а,б,в,г,д] е постигната с една стъпка въртене на дясно или 5 стъпки наляво. Ротацията винаги е симетрична на броя на елементите на множеството, т.е ротация с 6 стъпки привежда множеството във същата конфигурация. Броят на пермутациите е  Pn=n! (пермутацията е код, наредбата е значима)

**Вариация:** Вариацията е същата като пермутацията но се взима подмножество с точно определена дължина. Например: [в,е] е вариация на М. Всички двойки [в,е] са 2-ри клас вариация от ен елемента на М. Видовете класове са ясни: двоики, тройки , четворки и така натам. Формулата е  Vkn=n!/(n-k)!; k ≤ n. Т.е всяка ка-орка в М е вариция (к<n). В тази формула се работи само с един к-клас който ни интересува. (вариацията е подкод, подподредбата е значима)

**Комбинация:**  Във същият дух: Всеки клас к от **ненаредени** елементи в М е комбинация. Т.е {а,б,д} е комбинация от 3ти клас, която е тъждествена с {а,д,б} и другите там възможности, за да изменим комбинация трябва да вложим нов елемент {a,b, e} и някой да напусне. Формулата е следната: C(n,k) = Vkn/Pk.Броят на комбинаците значително спада. (комбинацията не е код, подредбата не е значима) (Пример: спорт тото) Комбинацията е к-та вариация разделена на к-та пермутация, защото пермутациите елеминират подредбите.