1. **List<T>**   
   Използвайте List<T>, когато очаквате да **добавяте нови елементи в края**, но не и средата и да достъпвате елементите по индекс.

- **Добавянето на елемент** е с константна сложност **(много бързо)**

- **Търсенето на елемент (Contains)**  е с линейна сложност (**бавно**)

- **Изтриване на елемент** е с линейна сложност(**бавно**)

- **Достъпване на елемент по индекс** е с константна сложност **(много бързо)**

1. **LinkedList<T>**

Използвайте свързан списък (LinkedList<T>), когато трябва да **добавяте и изтривате елементи от двата края** на списъка. В противен случай ползвайте List<T>.

- **Добавянето на елемент** е с константна сложност **(много бързо, но не колкото при List<T>)**

- **Търсенето на елемент (Contains)**  е с линейна сложност (**бавно**)

- **Изтриване на елемент** е с линейна сложност(**бавно**)

- **Достъпване на елемент по индекс** е с линейна сложност(**бавно**)

1. **Array Т[]**

Масивите трябва да се ползват само когато трябва да обработим **фиксиран брой елементи**, до които е необходим достъп по индекс.

- **Добавянето на елемент** е с линейна сложност (**бавно**)

- **Търсенето на елемент (Contains)**  е с линейна сложност (**бавно**)

- **Изтриване на елемент** е с линейна сложност(**бавно**)

- **Достъпване на елемент по индекс** е с константна сложност **(много бързо)**

1. **Stack<T>**

В стекът можем да добавяме елементи най-отгоре и да извличаме последния добавен елемент, но не и предходните. **Стек използваме, когато елементите ни трябват в обратен ред.**

- **Добавянето на елемент** е с константна сложност **(много бързо)**

- **Търсенето на елемент (Contains)**  **не се поддържа!**

- **Изтриване на елемент** е с константна сложност **(много бързо)**

- **Достъпване на елемент по индекс** **не се поддържа!**

1. **Queue<Т>**

В опашката можем да добавяме елементи само най-отзад и да извличаме елементи само най-отпред. **Опашка използваме, когато очакваме да получим елементите в реда, в който сме ги поставили.**

- **Добавянето на елемент** е с константна сложност **(много бързо)**

- **Търсенето на елемент (Contains)**  **не се поддържа!**

- **Изтриване на елемент** е с константна сложност **(много бързо)**

- **Достъпване на елемент по индекс** **не се поддържа!**

1. **Dictionary<К, V>**

В общия случай Dictionary<K, V> работи малко по-бързо от SortedDictionary<K, V> и е за предпочитане.

- **Добавянето на елемент** е с константна сложност **(много бързо)**

- **Търсенето на елемент (Contains)**  е с константна сложност **(много бързо)**

- **Изтриване на елемент** е с константна сложност **(много бързо)**

- **Достъпване на елемент по индекс** **не се поддържа!**

1. **SortedDictionary<K, V>**

Използвайте SortedDictionary<K, V>, когато искате бързо да добавяте елементи, бързо да търсите по ключ и елементите ще ви трябват след това сортирани по ключ.

- **Добавянето на елемент** е с логаритмична сложност **(бързо)**

- **Търсенето на елемент (Contains)**  е с логаритмична сложност **(бързо)**

- **Изтриване на елемент** е логаритмична сложност **(бързо)**

- **Достъпване на елемент по индекс** **не се поддържа!**

1. **HashSet<Т>**

Ползвайте HashSet<Т>, когато трябва бързо да добавяте елементи към множество и да проверявате дали даден елемент е от множеството. **Поддържа най-бързото търсене!**

- **Добавянето на елемент** е с константна сложност **(много бързо)**

- **Търсенето на елемент (Contains)**  е с константна сложност **(много бързо)**

- **Изтриване на елемент** е с константна сложност **(много бързо)**

- **Достъпване на елемент по индекс** **не се поддържа!**

1. **SortedSet<Т>**

Използвайте SortedSet<Т>, когато трябва бързо да добавяте елементи към множество и да проверявате дали даден елемент е от множеството и освен това елементите ще ви трябват сортирани в нарастващ ред.

- **Добавянето на елемент** е с логаритмична сложност **(бързо)**

- **Търсенето на елемент (Contains)**  е с логаритмична сложност **(бързо)**

- **Изтриване на елемент** е логаритмична сложност **(бързо)**

- **Достъпване на елемент по индекс** **не се поддържа!**

**Съвет:**

**Тръгвайте винаги от алгоритъма към структурите от данни, а не обратното!**

* **При достъпване на елемент по индекс използваме List<T> или Array Т[].**
* **При търсене на елемент (Contains) използваме Dictionary<К, V> или HashSet<Т>.**
* **При добавяне и изтриване на елемент използваме HashSet<Т>, Dictionary<К, V>, Stack<T>, Queue<Т>**
* **При добавяне на елемент (без изтриване) използваме List<T> или LinkedList<T>**
* **При нужда от сортирани резултати използваме SortedDictionary<K, V> или SortedSet<Т>**

**Сложност на алгоритъм**

|  |  |  |
| --- | --- | --- |
| **Сложност** | **Означение** | **Описание** |
| **константна** | O(1) | За извършване на дадена операция са необходими константен брой стъпки (например 1, 5, 10 или друго число) и този брой не зависи от обема на входните данни. |
| **логаритмична** | O(log(N)) | За извършване на дадена операция върху N елемента са необходими брой стъпки от порядъка на log(N), където основата на логаритъма е най-често 2. Например алгоритъм със сложност O(log(N)) за N = 1 000 000 ще направи около 20 стъпки (с точност до константа). Тъй като основата на логаритъма няма съществено значение за порядъка на броя операции, тя обикновено се изпуска. |
| **линейна** | O(N) | За извършване на дадена операция върху N елемента са необходими приблизително толкова стъпки, колкото са елементите. Например за 1 000 елемента са нужни около 1 000 стъпки. Линейната сложност означава, че броят елементи и броят операции са линейно зависими, например броят стъпки за N елемента е около N/2 или 3\*N.  **(единичен цикъл – бърз при много елементи)** |
| **n\*log(n)** | O(n\*log(n)) | За извършване на дадена операция върху N елемента са необходими приблизително N\*log(N) стъпки. Например при  1 000 елемента са нужни около 10 000 стъпки. |
| **квадратична** | O(n2) | За извършване на дадена операция са необходими от порядъка на N2 на брой стъпки, където N характеризира обема на входните данни. Например за дадена операция върху 100 елемента са необходими 10 000 стъпки. Реално квадратична сложност имаме, когато броят стъпки е в квадратна зависимост спрямо обема на входните данни, например за N елемента стъпките могат да са от порядъка на 3\*N2/2.  **(два вложени цикъла – бърз до 10 000 елемента)** |
| **кубична** | O(n3) | За извършване на дадена операция са необходими от порядъка на N3 стъпки, където N характеризира обема на входните данни. Например при 100 елемента се изпълняват около  1 000 000 стъпки.  **(три вложени цикъла – бърз до 1000 елемента)** |
| **експоненциална** | O(2n), O(N!), O(nk), … | За извършване на дадена операция или изчисление са необходими брой стъпки, който е в експоненциална зависимост спрямо размера на входните данни. Например при N=10 експоненциалната функция 2N има стойност 1024, при N=20 има стойност 1 048 576, а при N=100 функцията има стойност, която е число с около 30 цифри. Експоненциалната функция N! расте още по-бързо: за N=5 има стойност 120, за N=10 има стойност 3 628 800, а за N=20 – 2 432 902 008 176 640 000. |