Bad choice Best choice Good choice

Important thing – most useful for most of tasks

|  |  |  |  |
| --- | --- | --- | --- |
|  | Array | LinkedList<T> | List<T> |
| Lot of Append calls | Very slow (need to reallocate memory) | Relatively Fast (constant time) | Even if (amortized constant time, linear in worst case) this is fastest append. Because of it’s “pseudoAppend” |
| Lot of Prepend calls | Very slow (need to reallocate memory) | Fastest (constant time) | Relatively Fast (linear time) |
| Lot of Insertion | Very slow (need to reallocate memory) | Fastest (constant time) | Slow |
| Lot of removal | Very slow (need to reallocate memory) | Remove(item) - linear time; Remove(node) - constant time | Remove(item) - linear time; RemoveAt(index) - linear time |
| Ram memory usage (<85000 bytes) | Lowest | Largest | Good |
| Ram memory usage (>85000 bytes) | this can lead to heap fragmentation, a mild form of memory leak | Best choice | this can lead to heap fragmentation, a mild form of memory leak |
| Random access speed | Best | Very slow. But sequential data accessing is great (forward/backward) | Best |
| Access by index | + | - | + |
| Data is not fragmented in RAM | + | - | + |
| Count; Length speed | Fastest | 3 times slower | 3 times slower |
| Contains() speed | Fastest | Pretty fast | Pretty fast |
| Max possible elems count | 2GB; But can be larger on x64 with GC.AllowVeryLargeObjects  4 billion elements; | ? | 2GB; But can be larger on x64 with GC.AllowVeryLargeObjects  4 billion elements; |

|  |  |  |  |
| --- | --- | --- | --- |
|  | Int[] | LinkedList<int> | List<int> |
| Lot of Append calls | Range: 0....560000  Time:00:00:01.8000000 | Range: 0....560000  Time:00:00:00.0319849 | Range: 0....560000  Time:00:00:00.0063821 |
| Lot of Prepend calls |  | Range: 0....560000  Time:00:00:00.0700003 | Range: 0....560000  Time:00:00:00.7000000 |
| Lot of Insertion |  | Range: 0....560000  Time:00:00:00.0305724 | Range: 0....560000  Time:00:00:39.1222610 |
| Lot of removal |  |  |  |
| Ram memory usage (<85000 bytes) | int[560 000]  Weight: 2284 Kb | Elems: 560 000  Weight: 4193 Kb | Elems: 560 000  Weight: 4193 Kb |
| Ram memory usage (>85000 bytes) | int[56 000 000] Weight: 218852 Kb  But this can lead to heap fragmentation, a mild form of memory leak | Elems: 56 000 000  Weight: 1 312 604 Kb | Elems: 56 000 000  Weight: 262 246 Kb  But this can lead to heap fragmentation, a mild form of memory leak |
| Random access speed | Best | Very slow. But sequential data accessing is great (forward/backward) | Best |
| Count; Length speed | Items: 560000  Called times: 560000  Time: 00:00:00.0021742  Items: 560000  Called times: 560000000  Time: 00:00:01.5768008 | Items: 560000  Called times: 560000  Time: 00:00:00.0017365  Items: 560000  Called times: 560000000  Time: 00:00:01.6869734 | Items: 560000  Called times: 560000  Time: 00:00:00.0022802  Items: 560000  Called times: 560000000  Time: 00:00:01.5626380 |
| Contains() speed | Items: 560000  Called times: 560000  Time: 00:00:04.5355194 | Items: 560000  Called times: 560000  Time: 00:00:15.0761847 | Items: 560000  Called times: 560000  Time: 00:00:14.4939457 |

<http://stackoverflow.com/questions/169973/when-should-i-use-a-list-vs-a-linkedlist/29263914#29263914>