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 “pseudo Append” |
| Lot of Prepend calls | Very slow (need to reallocate memory) | Fastest (constant time) | Relatively Fast  \* -- linear time change |
| Lot of Insertion | Very slow (need to reallocate memory) | Fastest (constant time) | Very 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 |
| Data size < 85000 bytes | Lowest | Largest | Good |
| Data size > 85000 bytes | this can lead to heap fragmentation, a mild form of memory leak | Allowable choice, but takes a lot of RAM for “links”.  Best choice in case of containing high-weight objects, but count of objects is relatively low | this can lead to heap fragmentation, a mild form of memory leak |
| Sequential access speed: Foreach() | Fast | Relatively Slow | Relatively Fast |
| Sequential access speed: For () | Fast | No Access by index | Fast  ( but slower than Array’s ) |
| Random access speed | Fast | No Access by index | Fast |
| Count; Length speed | Fast | Fast | Fast |
| Contains() speed | Fast | Pretty fast | Pretty fast |
| Max possible elems count | 2GB; But can be larger on x64 with GC.AllowVeryLargeObjects  4 billion elements; | ?  Better to use with lower elements count, but great in use for *high-weight objects* | 2GB; But can be larger on x64 with GC.AllowVeryLargeObjects  4 billion elements; |
| Data is NOT fragmented in RAM | + | Fragmented | + |

|  |  |  |  |
| --- | --- | --- | --- |
|  | Int[] | LinkedList<int> | List<int> |
| Lot of Append calls | [Memory relocate]  Range: 0....560000  Time:00:00:01.8000000 | Range: 0....560000  Time:00:00:00.0319849 | Range: 0....560000  Time:00:00:00.0063821  [fastest because this is pseudo append] |
| Lot of Prepend calls | Very slow (need to reallocate memory) | Range: 0....560000  Time:00:00:00.0700003 | Range: 0....560000  Time:00:00:00.7000000 \* |
| Lot of Insertion | Very slow (need to reallocate memory) | Range: 0....560000  Time:00:00:00.0305724 | Range: 0....560000  Time:00:00:39.1222610 |
| Lot of removal | Very slow (need to reallocate memory) | ??????????????????????  ??????????????????????  ?????????????????????? | ??????????????????????  ??????????????????????  ?????????????????????? |
| Data size < 85000 bytes | int[560 000]  Weight: 2 284 Kb | Elems: 560 000  Weight: 13 230 Kb | Elems: 560 000  Weight: 4 193 Kb |
| Data size > 85000 bytes | int[56 000 000] Weight: 218 852 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 |
| Sequential access speed: Foreach() | Items: 5600  Called times: 5600  Time: 00:00:00.1083056  Items: 56000  Called times: 56000  Time: 00:00:07.9632761 | Items: 5600  Called times: 5600  Time: 00:00:00.2017669  Items: 56000  Called times: 56000  Time: 00:00:19.8083982 | Items: 5600  Called times: 5600  Time: 00:00:00.1449199  Items: 56000  Called times: 56000  Time: 00:00:14.5889001 |
| Sequential access speed: For () | Items: 5600  Called times: 5600  Time: 00:00:00.0821685  Items: 56000  Called times: 56000  Time: 00:00:08.2556414 | No Access by index | Items: 5600  Called times: 5600  Time: 00:00:00.1089242  Items: 56000  Called times: 56000  Time: 00:00:10.8495136 |
| Random access speed | Items: 56000000  Called times: 56000000  Time: 00:00:06.2688381 | No Access by index | Items: 56000000  Called times: 56000000  Time: 00:00:06.2979606 |
| 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>