# Data Structures/Tradeoffs



It is important to fully understand the problem you need to solve before choosing a data structure because each structure is optimized for a particular job. Hash tables, for example, favor fast lookup times over memory usage while arrays are compact and inflexible. Other structures, such as stacks, are optimized to enforce rigid rules on how data is added, removed and accessed throughout the program execution. A good understanding of data structures is fundamental because it gives us the tools for thinking about a program's behavior in a structured way.

```
[TODO:]
Use asymptotic behaviour to decide, most importantly seeing how the
structure will be used: an infrequent operation does not need to be
fast if it means everything else will be much faster
```

```
[TODO:]
Can use a table like this one to compare the asymptotic behaviour of every
structure for every operation on it.
```

Sequences (aka lists):

| | Array | Dynamic Array | Array Deque | Singly Linked List | Double Linked List |
|---|---|---|---|---|---|
| Push (Front) | - | O(n) | O(1) | O(1) | O(1) |
| Pop (Front) | - | O(n) | O(1) | O(1) | O(1) |
| Push (Back) | - | O(1) | O(1) | O(n), maybe O(1)* | O(1) |
| Pop (Back) | - | O(1) | O(1) | O(n) | O(1) |
| Insert before (given iterator) | - | O(n) | O(n) | O(n) | O(1) |
| Delete (given iterator) | | O(n) | O(n) | O(n) | O(1) |
| Insert after (given iterator) | | O(n) | O(n) | O(1) | O(1) |
| Delete after (given iterator) | - | O(n) | O(n) | O(1) | O(1) |
| Get nth element (random access) | O(1) | O(1) | O(1) | O(n) | O(n) |
| Good for implementing stacks | no | yes (back is top) | yes | yes (front is top) | yes |
| Good for implementing queues | no | no | yes | maybe* | yes |
| C++ STL | `std::array` | `std::vector` | `std::deque` | `std::forward_list` | `std::list` |
| Java Collections | `java.util.Array` | `java.util.ArrayList` | `java.util.ArrayDeque` | - | `java.util.LinkedList` |

```
* singly-linked lists can push to the back in O(1) with the modification that you keep a pointer to the last node
```

Associative containers (sets, associative arrays):

| | Sorted Array | Sorted Linked List | Self-balancing Binary Search Tree | Hash Table |
|---|---|---|---|---|
| Find key | O(log n) | O(n) | O(log n) | O(1) average O(n) worst |
| Insert element | O(n) | O(n) | O(log n) | O(1) average O(n) worst |
| Erase key | O(n) | O(n) | O(log n) | O(1) average O(n) worst |
| Erase element (given iterator) | O(n) | O(1) | O(1) | O(1) |
| Can traverse in sorted order? | yes | yes | yes | no |
| Needs | comparison function | comparison function | comparison function | hash function |
| C++ STL | - | - | `std::set`<br>`std::map`<br>`std::multiset`<br>`std::multimap` | `__gnu_cxx::hash_set`<br>`__gnu_cxx::hash_map`<br>`__gnu_cxx::hash_multiset`<br>`__gnu_cxx::hash_multimap` |
| Java Collections | - | - | `java.util.TreeSet`<br>`java.util.TreeMap` | `java.util.HashSet`<br>`java.util.HashMap` |

- Please correct any errors

Various Types of Trees

| | Binary Search | AVL Tree | Binary Heap (min) | Binomial Queue (min) |
|---|---|---|---|---|
| Insert element | O(log n) | O(log n) | O(log n) | O(1) (on average) |
| Erase element | O(log n) | O(log n) | unavailable | unavailable |
| Delete min element | O(log n) | O(log n) | O(log n) | O(log n) |
| Find min element | O(log n) | O(log n) | O(1) | O(log n) (can be O(1) if ptr to smallest) |
| Increase key | unavailable | unavailable | O(log n) | O(log n) |
| Decrease key | unavailable | unavailable | O(log n) | O(log n) |
| Find | O(log n) | O(log n) | unavailable | unavailable |
| Delete element | O(log n) | O(log n) | unavailable | unavailable |
| Create | O(1) | O(1) | O(1) | O(1) |
| find kth smallest | O(log n) | O(log n) | O((k-1)*log n) | O(k*log n) |

Hash table:

| | Hash table (hash map) |
|---|---|
| Set Value | Ω(1), O(n) |
| Get Value | Ω(1), O(n) |
| Remove | Ω(1), O(n) |

```
[TODO:]
Can also add a table that specifies the best structure for some specific need
e.g. For queues, double linked. For stacks, single linked. For sets, hash tables. etc...
```

```
[TODO:]
Could also contain table with space complexity information (there is a significative cost
in using hashtables or lists implemented via arrays, for example).
```



Retrieved from "https://en.wikibooks.org/w/index.php?title=Data_Structures/Tradeoffs&oldid=3090331"

**This page was last edited on 14 June 2016, at 19:22.**

Text is available under the Creative Commons Attribution-ShareAlike License.; additional terms may apply. By using this site, you agree to the Terms of Use and Privacy Policy.