| Algoritm | insertion | deletion | element access |
|---|---|---|---|
| Linked list | O(1) | O(1) | O(n) |
| Arrays | O(n) | O(n) | O(1) |
| Dynamic Arrays | O(1) at the end | O(1) at the end | O(1) |

| Heap | bubble-up | bubble-down | heapify |
|---|---|---|---|
| | O(log n) | O(log n) | Θ (n) |

| Algorithm | Runtime |
|---|---|
| Quick Select | Average Case Θ(n) Worst Case: Θ(n^2) |
| Rrandomized Quick Select | Expected: Θ(n) Worst Case: Θ(n^2) |
| Medians of Five Quick Select | Worst Case Θ(n) |

| Sort | Running time | Analysis |
|---|---|---|
| Selection Sort | Θ(n^2) | worst-case |
| Insertion Sort | Θ(n^2) | worst-case |
| Bubble Sort | Θ(n^2) | worst-case |
| Merge Sort | Θ(n log n) | worst-case |
| Heap Sort | Θ(n log n) | worst-case |
| Quick Sort | Θ(n log n) | average-case (worst case Θ(n^2)) |
| Randomized Quick Sort | Θ(n log n) | expected (worst case Θ(n^2)) |
| Medians of Five Quick Sort | Θ(n log n) | worst-case |
| Heap Sort | O(n log n) | average-case |

| Trees | Search | Insert | Delete | Note |
|---|---|---|---|---|
| Binary Search Tree | Θ(h) | Θ(h) | Θ(h) | h(height of the tree): Worst-case Θ(n), Best-case Θ(log n), Avaerage-case Θ(log n) |
| AVL Tree | Θ(h) | Θ(h) | Θ(h) | h: Θ(log n) |
| 2-3 Tree (B Tree) | Θ(h) | Θ(h) | Θ(h) | h: O(logn/logd * logd) = O(log n) |

| Hashing | Search | Insert | Delete |
|---|---|---|---|
| Linear Probing | 1/(1 – α)^2 | 1/(1 – α)^2 | 1/(1 – α) |
| Double Hashing | 1/(1 – α) | 1/(1 – α) | 1/α * log(1/(1 – α)) |
| Cuckoo Hashing | 1 | α/(1 – 2α)^2 | 1 |
| Extendible Hashing | CPU time: Θ(log S) **Page faults**: 1 | Θ(S) without directory grow\shrink, otherwise Θ(2^d) **Page faults**:1 or 2 | same as Insert |

| Advantages of Balances Search Trees | Advantages of Hash table |
|---|---|
| O(log n) worst-case operation cost | O(1) cost, but only on average |
| Does not require any assumptions, special functions, or known properties of input distribution | Flexible load factor parameters |
| No wasted space | Cuckoo hashing achieves O(1) worst-case for search & delete |
| Never need to rebuild the entire structure | |
| Both approaches can be adopted to minimize page faults | Both approaches can be adopted to minimize page faults |

| Search | Runtime |
| --- | --- |
| Binary Search | **Worst Case** O(log n) **Best Case** O(1) **Average Case** O(log n) |
| Interpolation Seaerch | O(loglog n) **on average** (keys uniformly distributed) **worst case** O(n) |
| Gallop Search | O(log m) comparisons (m: location of k in A) |
| Skip List | **Expected Space**: O(n) **Expected height** O(log n) **Search, Insert, Delete**: O(log n) |
| Range Search on BST | O(log n + k) k: number of reported items |

| Multi-Dimensional Data | Build initial tree | Range Search | Note |
| --- | --- | --- | --- |
| Quadtree | **worst-case** Θ(#nodes) = Θ(nh) | **worst-case** O(nh) | height ∈ Θ(log2 (dmax/dmin)) |
| Kd-trees (d dimensional) | O(n log n) | O(n ^(1−1/d) + k) | **Storage**: O(n) |
| Range Tree (d dimensional) | O(n (log n) ^(d−1)) | O((log n)^d + k) | **Storage**:O(n (log n)^(d −1)) |

| | Brute-Force | KMP | Boyer-Moore | Suffx trees |
| --- | --- | --- | --- | --- |
| **Preprocessing** | / | O(m) | O (m + \|Σ\|) | O(n^2) |
| **Search time** | O(nm) | O(n) | O(n) often better | O(m) |
| **Extra space** | / | O(m) | O (m + \|Σ\|) | O(n) |

| Algorithm | Property | Extra data | Runtime |
|---|---|---|---|
| **REL** | Variable-width, multiple-character encoding | Nothing | |
| **Huffman** | Variable-width, single-character | Compressed Trie | build decoding trie: O(\|S\| + \|Σ\| log \|Σ)) |
| **MTF** | Adaptive, transform to smaller integers, Must be followed by variable-width integer encoding | Initial order dictionary | |
| **LZW** | Adaptive,fixed-width,multiple-character encoding, Augments dictionary with repeated substrings | Nothing *(decoder will make a same dictionary by Text)* | |
| **BWT** | Block compression method, must be followed by MTF | Nothing | Encoding: O(n^2) Decoding: O(n^2) |