# CheatSheet INF102

Erik Fjelltveit Nyhuus

November 2024

## Contents

# 1 Sorting Algorithms

## 1.1 Selection sort

Time Complexity = $O(n^2)$
Sorts an array by repeatedly selecting the smallest or largest element from the unsorted portion and swapping it with the first unsorted element. Continues until list is sorted. Two pointers one at the first in the unsorted protion and one iterating through list to find the next smallest or largets element.

## 1.2 Insertion sort

Time Complexity = $O(n^2)$
Simple sorting algorithm that works by iteratively inserting each element of an unsorted list into its correct position in a sorted portion of the list. The same algorithm you use when sorting playing cards. You pick a card and insert it into the correct relative position. When you find a pair that is unorder, swap them and continue swapping the unsorted elements through the sorted part of the array until it is in the right space.

## 1.3 Bubble sort

Time Complexity = $O(n^2)$
Works by repeatedly swapping the adjacent elements if they are in the wrong order. Each pass lokking ofr two elements that are in the wrong order. Continues this process until it has a pass with know swaps.

## 1.4 Quick sort

Worst Case = $O(n^2)$ Occures with poor pivot.
Average Case = $\theta(nlog(n))$
Based on divide and conquer. Quick sort chooses a random pivot P, and swaps P with the last element. Two pointer, left bound and right bound points to index 0 and n - 1. Left bound moves to the right until it hits an element $\geq P$ or crosses the right bound. Right bound does the samle until i finds an element $\leq P$. Swap elements that right and left bound points to. Continue this process until either they crosses, this indicates that all elements smaler than $P$ is on the left and larger are one the right. The last element right or left bound points to swap places with pivot. Since pivot was left at the end in first step. Repeat this with left and right sub-list.

## 1.5 Merge sort

Average Case = $(nlog(n))$
Divide the list into two smaller sublists, continues on dividing until list has size 1. Merges each of the smaller lists in correct order until everything is sorted.

## 1.6 Bucket sort

Average Case = $(nlog(n))$
Works if you have repeated elements in a list. Adds elements into different groups based on size. Sort each bucket on it own afterwards.

## 1.7 Radix sort

Rather than comparing elements directly, Radix Sort distributes the elements into buckets based on each digit's value. By repeatedly sorting the elements by their significant digits, from the least significant to the most significant.

## 2 ArrayList vs. LinkedList

| Operation | ArrayList | LinkedList |
|---|---|---|
| size() | O(1) | O(1) |
| add() | O(n)* | O(1) |
| contains(obj) | O(n) | O(n) |
| remove(obj) | O(n) | O(n) |
| toArray() | O(n) | O(n) |
| indexOf(obj) | O(n) | O(n) |
| get(int i) | O(1) | O(n) |
| set(int i, E e) | O(1) | O(n) |
| addFirst() | O(n) | O(1) |

- *O(1) in amortized time (when resizing is not needed)

## 3 ArrayList vs. LinkedList (Queue/Stack)

| | **ArrayList** | | **LinkedList** | |
|---|---|---|---|---|
| | **Queue** | **Stack** | **Queue** | **Stack** |
| offer / push | O(n) | O(n)* | O(1) | O(1) |
| poll / pop | O(1) | O(1) | O(1) | O(1) |
| peek | O(1) | O(1) | O(1) | O(1) |

- *O(1) in amortized time (when resizing is not needed)

## 4 PriorityQueue

### 4.1 PriorityQueue

| Operation | Time Complexity |
|---|---|
| add() | O(log(n)) |
| remove(Head) | O(log(n)) |
| remove(Specific object) | O(n) |
| poll() | O(log(n)) |
| peek() | O(1) |
| size() | O(1) |

### 4.2 PriorityQueue - SortedList

| Operation | Time Complexity |
|---|---|
| add(T element) | O(n) |
| T findMin() | O(1) |
| T removeMin() | O(1) |

### 4.3 PriorityQueue - LinkedList

| **Operation** | **Time Complexity** |
|---|---|
| add(T element) | O(1) |
| T findMin() | O(n) |
| T removeMin() | O(n) |

# 5 HashSet vs. TreeSet

| **Operation** | **HashSet** | **TreeSet** |
|---|---|---|
| add() | O(1)* | O(log(n)) |
| remove() | O(1)* | O(log(n)) |
| contains(obj) | O(1)* | O(log(n)) |
| findMin | O(n) | O(log(n)) |
| findMax | O(n) | O(log(n)) |

- *HashSet har O(1) i snitt, men O(n) i worst case

# 6 Heap

In a heap, the parent of a node $k$ has the position $k/2$, and the two children in position $2k$ and $2k + 1$. A binary heap is a set of nodes with keys arranged in a complete heap-ordered binary tree, represented in level order in an array (not using the first entry). We can travers the heap with simple arithmerics, to move up set $k$ in the $array[k]$, to $k/2$. Or $2k + 1$ and $2k$ to move down the binary tree.

| **Operation** | **Time Complexity** |
|---|---|
| add(T element) | O(log(n)) |
| T peekMin() | O(1) |
| T removeMin() | O(log(n)) |
| Construct heap | O(n) |
| delete() | O(log(n)) |

# 7 Graph Datastructures

## 7.1 EdgeList

| **Metode** | **Kjøretid** |
|---|---|
| Adjacent | O(M)* |
| Vertices | O(M) |
| Edges | O(N) |
| Neighbours | O(M)* |
| AddVertex | O(1)* |
| AddEdge | O(1)* |

## 7.2 Adjacency Set

## 7.3 Adjacency List

| Metode | Kjøretid |
|---|---|
| Adjacent | O(1)* |
| Vertices | O(1) |
| Edges | O(M) |
| Neighbours | O(1)* |
| AddVertex | O(1)* |
| AddEdge | O(1)* |

| Method | Runtime |
|---|---|
| Adjacent | O(degree) |
| Vertices | O(1) |
| Edges | O(M) |
| Neighbours | O(1)* |
| addVertex | O(N) |
| addEdge | O(degree) |

### 7.4 Adjacency Matrix

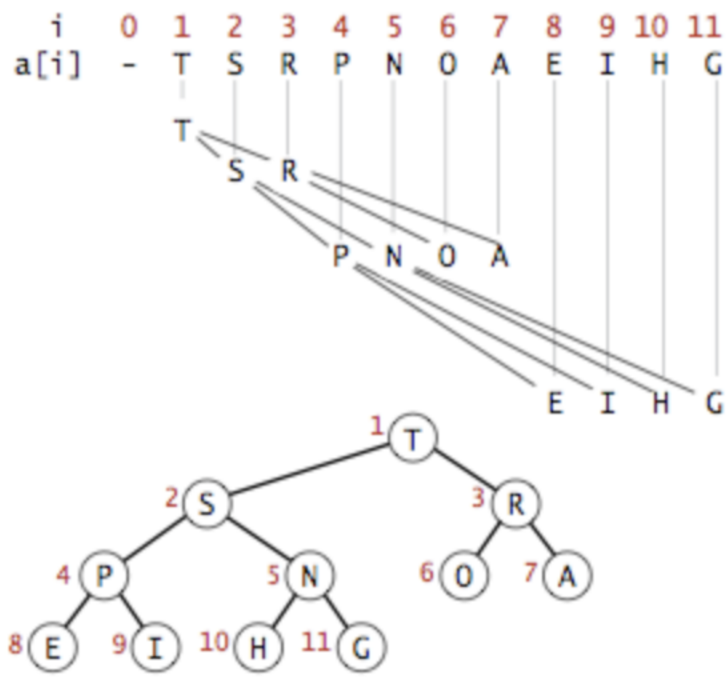
| Method | Runtime |
|---|---|
| Adjacent | $O(1)$ |
| Vertices | $O(1)$ |
| Edges | $O(N^2)$ |
| Neighbours | $O(N)$ |
| addVertex | $O(N^2)$ or $O(N)$ |
| addEdge | $O(1)$ |

## 8 Summary of Graph Algorithms

| Algorithm | Graph Type | Time Complexity |
|---|---|---|
| BFS | Unweighted | $O(m+n)$ |
| DFS | Unweighted | $O(m+n)$ |
| Dijkstra | Positive weights | $O(m \log m)$ |
| Bellman-Ford | Negative weights, no negative cycle | $O(n \cdot m)$ |
| Brute-Force | Negative weights | $2^{O(n)}$ |
| $A^*$ | Weighted | $mlog(n)$ |
| Kruskal's | Weighted | $O(m \log n)$ |
| Prim's | Weighted | $O(m \log n)$ |
| Union-Find | | $O(m \log n)^*$ |

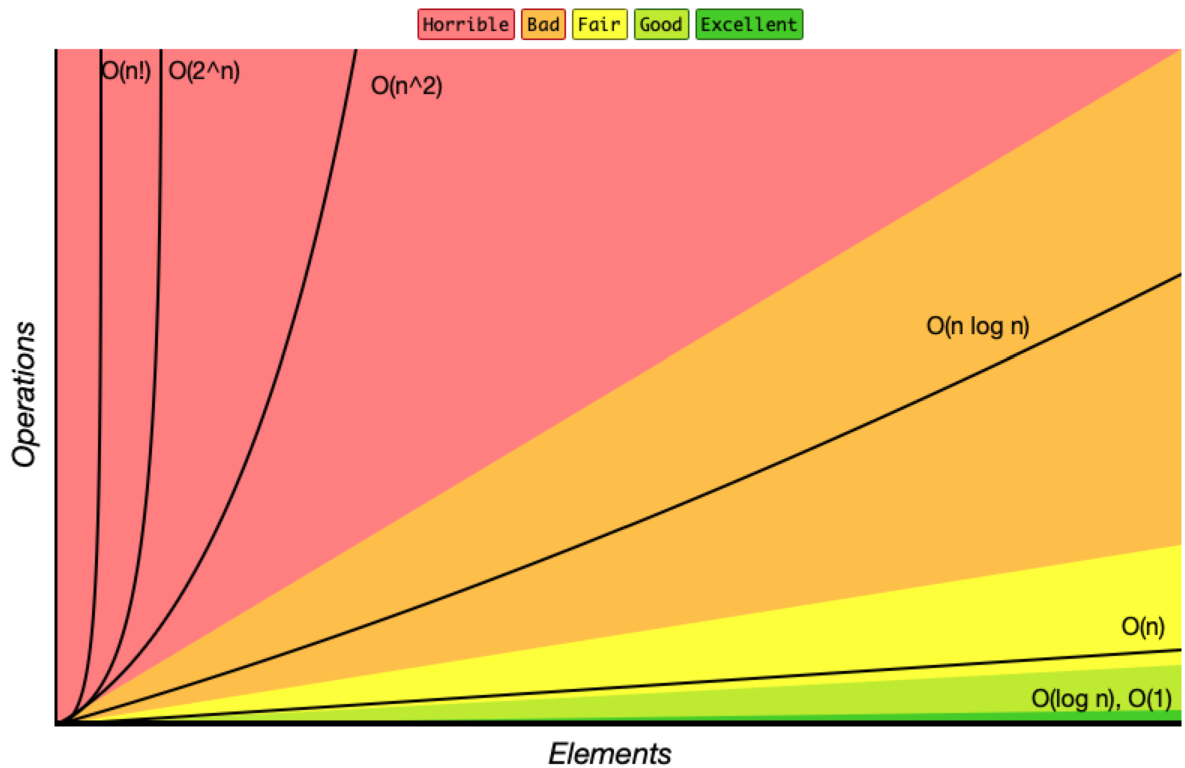
Table 1: Summary of Graph Algorithms

## 9 Images




Heap representations

## Big-O Complexity Chart




## Common Data Structure Operations

| Data Structure | Time Complexity | | | | | | | | Space Complexity |
|---|---|---|---|---|---|---|---|---|---|
| | Average | | | | Worst | | | | Worst |
| | Access | Search | Insertion | Deletion | Access | Search | Insertion | Deletion | |
| Array | Θ(1) | Θ(n) | Θ(n) | Θ(n) | O(1) | O(n) | O(n) | O(n) | O(n) |
| Stack | Θ(n) | Θ(n) | Θ(1) | Θ(1) | O(n) | O(n) | O(1) | O(1) | O(n) |
| Queue | Θ(n) | Θ(n) | Θ(1) | Θ(1) | O(n) | O(n) | O(1) | O(1) | O(n) |
| Singly-Linked List | Θ(n) | Θ(n) | Θ(1) | Θ(1) | O(n) | O(n) | O(1) | O(1) | O(n) |
| Doubly-Linked List | Θ(n) | Θ(n) | Θ(1) | Θ(1) | O(n) | O(n) | O(1) | O(1) | O(n) |
| Skip List | Θ(log(n)) | Θ(log(n)) | Θ(log(n)) | Θ(log(n)) | O(n) | O(n) | O(n) | O(n) | O(n log(n)) |
| Hash Table | N/A | Θ(1) | Θ(1) | Θ(1) | N/A | O(n) | O(n) | O(n) | O(n) |
| Binary Search Tree | Θ(log(n)) | Θ(log(n)) | Θ(log(n)) | Θ(log(n)) | O(n) | O(n) | O(n) | O(n) | O(n) |
| Cartesian Tree | N/A | Θ(log(n)) | Θ(log(n)) | Θ(log(n)) | N/A | O(n) | O(n) | O(n) | O(n) |
| B-Tree | Θ(log(n)) | Θ(log(n)) | Θ(log(n)) | Θ(log(n)) | O(log(n)) | O(log(n)) | O(log(n)) | O(log(n)) | O(n) |
| Red-Black Tree | Θ(log(n)) | Θ(log(n)) | Θ(log(n)) | Θ(log(n)) | O(log(n)) | O(log(n)) | O(log(n)) | O(log(n)) | O(n) |
| Splay Tree | N/A | Θ(log(n)) | Θ(log(n)) | Θ(log(n)) | N/A | O(log(n)) | O(log(n)) | O(log(n)) | O(n) |
| AVL Tree | Θ(log(n)) | Θ(log(n)) | Θ(log(n)) | Θ(log(n)) | O(log(n)) | O(log(n)) | O(log(n)) | O(log(n)) | O(n) |
| KD Tree | Θ(log(n)) | Θ(log(n)) | Θ(log(n)) | Θ(log(n)) | O(n) | O(n) | O(n) | O(n) | O(n) |

## Array Sorting Algorithms

| Algorithm | Time Complexity | | | Space Complexity |
|---|---|---|---|---|
| | Best | Average | Worst | Worst |
| Quicksort | Ω(n log(n)) | Θ(n log(n)) | O(n^2) | O(log(n)) |
| Mergesort | Ω(n log(n)) | Θ(n log(n)) | O(n log(n)) | O(n) |
| Timsort | Ω(n) | Θ(n log(n)) | O(n log(n)) | O(n) |
| Heapsort | Ω(n log(n)) | Θ(n log(n)) | O(n log(n)) | O(1) |
| Bubble Sort | Ω(n) | Θ(n^2) | O(n^2) | O(1) |
| Insertion Sort | Ω(n) | Θ(n^2) | O(n^2) | O(1) |
| Selection Sort | Ω(n^2) | Θ(n^2) | O(n^2) | O(1) |
| Tree Sort | Ω(n log(n)) | Θ(n log(n)) | O(n^2) | O(n) |
| Shell Sort | Ω(n log(n)) | Θ(n(log(n))^2) | O(n(log(n))^2) | O(1) |
| Bucket Sort | Ω(n+k) | Θ(n+k) | O(n^2) | O(n) |
| Radix Sort | Ω(nk) | Θ(nk) | O(nk) | O(n+k) |
| Counting Sort | Ω(n+k) | Θ(n+k) | O(n+k) | O(k) |
| Cubesort | Ω(n) | Θ(n log(n)) | O(n log(n)) | O(n) |