CheatSheet INF102

Erik Fjelltveit Nyhuus

November 2024

Contents

1 Search 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.

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.

1.3 Bubble sort

Time Complexity = $O(n^2)$
Works by repeatedly swapping the adjacent elements if they 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, picks an element that is used as a pivot and partitions the array by moving all elements greater then pivot to the right of pivot and elements less than pivot on the left side. Calls the same algorithm to the know two sub-arrays and repeats the process.

1.5 Merge sort

1.6 Bucket sort

1.7 Radix sort

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)

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

	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)

3 ArrayList vs. LinkedList (Queue/Stack)

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

4 PriorityQueue

4.1 PriorityQueue - SortedList

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

4.2 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 runtime

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

7 Graph Datastructures

7.1 Adjacency Set

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

7.2 Adjacency List

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

7.3 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