Graphs

- Topological Sort : reverse of DFS post-order, giving $O(V + E)$ time and $\Theta(V)$ memory
- Defn DAG : directed, acyclic graph
  - Topological Sort only possible for DAG
  - DAG SPT algorithm - relax in topological order, giving $O(V + E)$ time and $\Theta(V)$
  - Longest-Path Problem : best-known solution is exponential (for general graphs)
  - DAG LP - negate all values and run SPT, giving $O(V + E)$ time and $\Theta(V)$ memory
- Defn Reduction: If any subroutine for task Q can be used to solve P, then P reduces to Q.

Sorting

- Must obey
  - Trichotomy : exactly one of $a < b, a = b, a > b$ is true
  - Transitivity : $a > b, b > c$, then $a > c$
  - Any (comparison-based) Sorting algorithm has worst case at least $\Theta(N \log N)$
  - Worst-case runtime can be $\Theta(N)$ for non-comparison sort, e.g. Counting/Radix sort!
- Defn (Inversion) : a pair of elements that's out of order (sorting $\iff$ make # inversion $= 0$)
- Defn (Stability) : order of equivalent items is preserved
- Monotonically improving: Insertion, Selection, Merge, Quick, MSD; Not mono. improving: Heap, LSD. Double-checking by feeding a sorted list, if anything is moved, then not mono. improving

	Time Complexity (Best)	Time Complexity (Worst)	Stability
Selection Sort	$\Theta(n^2)$	$\Theta(n^2)$	No
Insertion Sort	$\Theta(n)$	$\Theta(n^2)$	Yes
Heapsort	$\Theta(n)$	$\Theta(n \log n)$	No
Mergesort	$\Theta(n \log n)$	$\Theta(n \log n)$	Yes
Quicksort (w/ Hoare Partitioning)	$\Theta(n \log n)$	$\Theta(n^2)$	No

Quicksort slow (3-array partitioning), $\Theta(N \log N)$, $\Theta(N^2)$, Yes

	Time Complexity (Best)	Time Complexity (Worst)	Stability
LSD Radix Sort	$\Theta(W(N+R))$	$\Theta(W(N+R))$	Yes
MSD Radix Sort	$\Theta(N+R)$	$\Theta(W(N+R))$	Yes

- Selection Sort
  - Procedure: Find the curMin and swap with curFront
  - Unstable example: $3a, 3b, 1$
  - $\Theta(1)$ space
  - Use a bar to indicate which part has been sorted already, finish when bar behind all)
- Insertion Sort (Use a bar similarly)
  - Procedure: For curElement, swap with its front until in-place
  - Runtime: $\Theta(N + K)$ for K inversions $\rightarrow \Theta(N^2)$ usually
  - Memory: $\Theta(1)$ (no extra space)
  - Special: Excellent on "almost sorted" arrays $-$ $\Theta(N)$ runtime $-$ proportional to # of inversions (Best sort for $N < 15$)
  - Use a bar similarly
- Merge Sort
  - Procedure: Split until each sub-array contains only 1 (base case $N = 1$), and then merge
  - Memory: $\Theta(N)$
  - Fastest "stable" sort
- Heap Sort
  - Procedure: Forms a max-heap with bottom-up heapification (sinks). Pop out maxElement each time, move it to the possible "end" of the array (swap with the last element in heap and complete the sink).
  - $\Theta(1)$ space
  - Unstable example: $1a, 1b, 1c$
- Quicksort
  - Procedure (L3S): partition into less + equal + greater, and recurse $-$ N.B. quicksorts one side at a time
  - In most cases, fastest sort with avg & expected $\Theta(N \log N)$; Memory $\Theta(\log N)$ expected (call stack)
  - To prevent worst case $\Theta(N^2)$, (1) randomly pick pivots, or (2) shuffle before start
  - Hoare examples:
    - Unstable : $3, 5a, 2, 5b, 1$
    - Best case : $3, 1, 2, 5, 4$ or $10, 10, 10, 10, 10$
    - Worst case : $1, 3, 3, 4, 5$
  - To pick exact median: QuickSelect uses partitioning $\Theta(N)$ on avg and worst case $\Theta(N^2)$. if use exact median pivots, then QS has the same best/worst case runtime.
  - Hoare partitioning:
    - Create L(ess) and G(reater) at left and right ends.
    - L pointer likes small, hates $\geq$; G pointer likes large, hates $\leq$.
    - Walk two pointers towards each other, pausing on hated respective item. When boht pointers pause, swap the two elements and move both forward by 1

        - When pointers cross, the walking is done
        - Swap pivot with G (the one started on the right end)

- Counting Sort (Stable)
    - Two-pass : 1 to count, 2 to put in stuff (complex case)
    - Runtime on N objects with alphabet size R is $\Theta(N+R)$ both time and space. Reasonable performance on $N \geq R$, i.e. smaller alphabet

- Radix sort
    - Both LSD and MSD are stable
    - LSD: For N elements, R-size alphabet, W-length words, runtime $\Theta(WN+WR)$ and space $\Theta(N+R)$. Since sorts from least significant, so best = worst
    - MSD: $\Theta(N+WR)$ space, sorts each sub-group separately (best $\neq$ worst)
    - For longer #s, change base into 256 and use Radix Sort $\gg$ QuickSort
    - Need padding for diff. length!

- Mergesort vs. Radix: For N words of length W
    - MS faster with highly dissimilar words, $\Theta(N \log N)$, while MSD takes $\Theta(WN)$
    - MSD faster for similar words or when N is sufficiently large, still $\Theta(WN)$ (if alphabet size is constant), while MS takes $\Theta(WN \log N)$

- Sorting Summary
    - Comparison
        - Selection (if heapify $\rightarrow$ Heap)
        - Merge
        - Partition (Quick)
        - Insertion (if insert into a BST, then $\rightarrow$ Partition above)
    - Alphabet (Counting)
    - Radix (LSD, MSD)

Extra Sanity Checks

- Always check for null before calling instance vars
- Must use .equals for comparing Strings and other Objects; Lists canNOT be compared directly with .equals
- For Hash(Tables), compute .hashCode() % buckets.length to get the bucket index for insertion; hashCode() must return an int and be the same for .equals(Object o) items
- Prim's consider vertices with distTo the MST-in-construction, rather than distTo source vertex
- N.B. difference between $\Omega(\approx$ but not really lower bound$), \Theta(=), O(\approx$ but not really upper bound$)$
- Sorting defaults to minSort (i.e. sort into ascending order)
- Topological Sort: reverse post-order DFS with runtime $\Theta(V + E)$
- Spindly graphs could potentially cause stackOverflow exception on recursive DFS (no overflow if using Stack)
- For DFS, must only mark nodes when they have actually been visited
- Trie usage: prefix operations
- Radix Sort: to work with words of different lengths, we must pad the end of the word with an empty character, which will be denoted by _ (if instead we were sorting numbers, we would "pad" the numbers with 0s at the front).
- Sorting vs. Searching
  - Comparison $\iff$ 2-3, LLRB, etc.
  - Alphabet $\iff$ separate chaining HT
  - Radix $\iff$ Tries
- Strings are immutable, so every append actually copies the entire string
- Compression
  - Huffman - implemented by Trie for (longest) prefix-matching
  - It is not possible for an algorithm to compress all bitstreams (else we can compress everything down to 1 bit eventually).
- Extra: Just-In-Time Compiler (JIT) - Java's secret optimization during execution, e.g. stops computing a number or etc. for a piece of code if it's not used
- Extra: It is impossible to write a program that even calculates the Kolmogorov Complexity of any bitstream
- Extra: $P = NP$?, e.g. no current efficient solution to longest-path in general graph (DAG is solved)
- Extra: The min-max heap property is: each node at an even level in the tree is less than all of its descendants, while each node at an odd level in the tree is greater than all of its descendants.