Search Method	Method Description	Data Structure	Time Complexity	Space Complexity	Optimal for finite input?	Complete for finite input?	Notes
Breadth-first Search (BFS)	Always expand the shallowest node, put new nodes at the end of queue.	First-in-first-out Queue	**$O(b^{d+1})$** as each level of depth (n) adds b^n to running time	**$O(b^d)$** assuming only fringes stored in memory	Yes if no difference in cost of arcs.	Yes	Space is a bigger problem than time.
Depth-first Search (DFS)	Always expand the first deepest node in queue. Put new nodes at the start of queue.	Last-in-first-out Queue	**$O(b^m)$**. Note that it is maximum possible depth instead of goal node's depth	**O(bm)**. Linear space complexity in relation to maximum breadth and depth.	No. May find an suboptimal solution first.	Yes.	Not complete for infinite sets as it could run forever down an infinite branch.
Depth-limited Search (DLS)	Same as DFS, but drop instead of expanding nodes in queue exceeding maximum depth limit.	Last-in-first-out Queue with additional constraints.	**$O(b^l)$** where l is the maximum depth limit.	**O(bl)**. Linear space complexity in relation to maximum breadth and depth limit.	No.	No when solution depth exceeding maximum depth.	
Iterative Deepening Search (IDS)	For each *max_d* in range [0..d], expand from the root node in every iteration, add new nodes to the front of the queue. However, do not expand if the maximum depth in the queue will exceed the current *max_d* after expansion.		**$O(b^d)$** sum of multiplications of remaining depth and b^n at each level of depth (n).	**O(bd)**. Linear space complexity in relation to maximum breadth and depth.	Yes, if uniform step cost.	Yes.	As new nodes are added to the front, the version in lectures is likely ID(DF)S.
Uniformed-cost search (UCS)	Expand the current lowest cost node in the queue, and add new nodes to the queue, then	Queue	**$O(b^d)$** worst case.	**$O(b^d)$** worst case.	Yes.	Yes for full-positive transition costs.	Remember to always expand nodes in order of cost, even if a

	reorder queue by cost.						goal node is in queue already.
Greedy Best-first Search (GBFS)	Ignore the path cost, expand the node in queue with lowest heuristic cost, then add new nodes to queue, and sort the queue by heuristic cost.	Queue	**$O(b^m)$** with significant improvements from good heuristics.	**$O(b^m)$** as all nodes must be stored in memory.	No.	No, it could get stuck in a loop.	Heuristic cost is the estimated cost from that node to goal. For heuristic cost to be admissible, it must not ever be larger than actual cost.
A* Search	Similar to GBFS, but consider expand choice and ordering by the sum of cost to reach a node so far, and that node's heuristic cost.	Queue.	**$O(b^m)$** unless the error function grows no faster than logarithmic scale.	**$O(b^m)$** as all nodes must be stored in memory.	Yes.	Yes.	Remember to always expand nodes in order of total cost.

Disclaimer:

This is a collection of search algorithms covered in the first part of the Artificial Intelligence (ARIN) module, initially produced for my own revision needs. This is by no means complete, may contain errors, and you should revise through all materials given in the module. **I could not be held responsible for any deviations of this collection from the original course content.** However, if you spot any problems in this collection, please do contact me through the email address on the top-right corner, so I can correct them for future readers.

This collection is based extensively on (public) lecture and practical materials produced by module leaders of ARIN Part I, most recently by Dr D. Kazakov. You should consult copyright holders and the department before using it for any purposes other than personal academic revisions. Source: 2015-2016 materials from http://www-module.cs.york.ac.uk/arin/ .