

Question1.

(a).

	Start10	Start12	Start20	Start30	Start40
Ucsdijkstra	2565	Stack limit	Stack limit	Stack limit	Stack limit
Ideepsearch	2407	13812	5297410	Spend too much time	Spend too much time
Astar	33	26	915	Stack limit	Stack limit
idastar	29	21	952	17297	112571

(b).

As we can see in the table above, ucsdijkstra spent the longest time and used the most memory. Ideepsearch is better than ucsdijkstra it won't save so many nodes but still spent a lot of time. Astar search spent very little time, but there is still a problem of memory overrun. Obviously, idastar is much better than others, it spent the least time and solve the problem about memory.

Question2.

(a).

	Start50		Start60		Start64	
IDA*	50	14642512	60	321252368	64	1209086782
1.2	52	191438	62	230861	66	431033
1.4	66	116174	82	3673	94	188917
1.6	100	34647	148	55626	162	235852
Greedy	164	5447	166	1617	184	2174

(b).

```
% Iterative Deepening A-Star Search
% COMP3411/9414/9814 Artificial Intelligence, UNSW, Alan Blair


% solve(Start, Solution, G, N)
% Solution is a path (in reverse order) from Node to a goal state.
% G is the length of the path, N is the number of nodes expanded.
solve(Start, Solution, G, N) :-
    nb_setval(counter,0),
    idastar(Start, 0, Solution, G),
    nb_getval(counter,N).

% Perform a series of depth-limited searches with increasing F_limit.
idastar(Start, F_limit, Solution, G) :-
    depthlim([], Start, 0, F_limit, Solution, G).

idastar(Start, F_limit, Solution, G) :-
    write(F_limit),nl,
    F_limit1 is F_limit + 2, % suitable for puzzles with parity
    idastar(Start, F_limit1, Solution, G).

% depthlim(Path, Node, Solution)
% Use depth first search (restricted to nodes with F <= F_limit)
% to find a solution which extends Path, through Node.

% If the next node to be expanded is a goal node, add it to
% the current path and return this path, as well as G.
depthlim(Path, Node, G, _F_limit, [Node|Path], G) :-
    goal(Node).

% Otherwise, use Prolog backtracking to explore all successors
% of the current node, in the order returned by s.
% Keep searching until goal is found, or F_limit is exceeded.
depthlim(Path, Node, G, F_limit, Sol, G2) :-
    nb_getval(counter, N),
    N1 is N + 1,
    nb_setval(counter, N1),
    % write(Node),nl, % print nodes as they are expanded
    s(Node, Node1, C),
    not(member(Node1, Path)), % Prevent a cycle
    G1 is G + C,
    h(Node1, H1),
    F1 is (2 - w) * G1 + w * H1,  change code here
    F1 <= F_limit,
    depthlim([Node|Path], Node1, G1, F_limit, Sol, G2).
```

For the code is “F1 is G1 + H1”, I changed it to F1 is (2 – w) \* G1 + w \* H1.

So if w is 1.2, this line will be “F1 is 0.8 \* G1 + 1.2 \* H1.

(c).

See table.

(d).

As we can see in the table above, the value of w close to 1, we will get a better solution (just means the length of the path short). Meanwhile the number of nodes expanded increases under normal circumstances. IDA\* expanded the most nodes and Greedy expanded the least nodes. So, as the value of w goes from 1(IDA\*) to 2(Greedy), the speed of finding a result will be faster but the quality of solution will be worse. (About speed: Greedy > 1.6 > 1.4 > 1.2 > IDA\*; About quality: Greedy < 1.6 < 1.4 < 1.2 < IDA\* under normal circumstances)