Question 1:
The table of Section a) as shown below:

	Start 10	Start 12	Start 20	Start 30	Start 40
UCS	2565	Mem	Mem	Mem	Mem
IDS	2407	13812	5297410	Time	Time
A*	33	26	915	Mem	Mem
IDA*	29	21	952	17297	112571

‘Mem’ means it shows ‘out of global stack’, and ‘Time’ means this program has no result returned until 5 minutes’ limit.

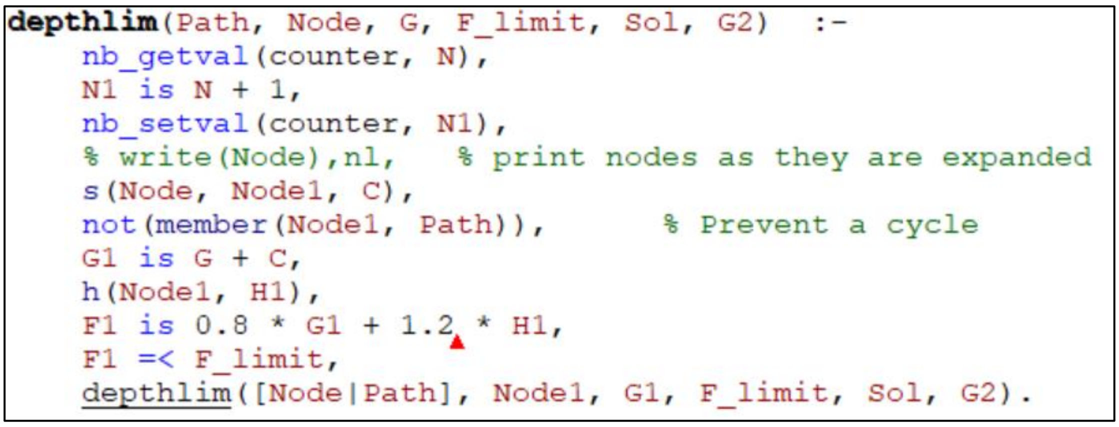
Section b):
Iterative Deepening A-Star Search is the most efficient among the four methods, and it is the only method succeeds in Start 40 task. As it deepens for each step, including repeating search, the time complexity will be more than A-Star Search ($O(b^{\varepsilon d})$), but the space complexity is $O(bd)$, which is extremely less than A-Star Search.
Uniform Cost Search can be the least efficient when the worst condition happens, the space complexity as well as time complexity can be up to $O(b^{\lfloor 1+(C^*/\varepsilon)\rfloor})$, which may use the most space and result in out of memory.
Iterative Deepening Search generates the most nodes until Start 20. The space complexity $O(bd)$ can be reasonable, but the time complexity is $O(b^d)$, which results in ‘Time-out’ when runs in more complex condition.
A-Star Search is the second efficient, based on the relative error $\varepsilon = (h^* - h)/h^*$, the time complexity is $O(b^{\varepsilon d})$ and can be up to $O(b^d)$. The drawback is it will store all generated nodes in the memory. For the worst condition, the space complexity can be up to $O(b^d)$, which will also result in out of memory.

Question 2:

The table of Section a) and Section c) as shown below:

	Start 50		Start 60		Start 64	
IDA*	50	1462512	60	321252368	64	1209086782
1.2	52	191438	62	230861	66	431033
1.4	66	116342	82	4432	94	190278
1.6	100	33504	148	55626	162	235848
Greedy	164	5447	166	1617	184	2174

Section b), the change of code and the replacement as shown below:



```
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 0.8 * G1 + 1.2 * H1,
    F1 =< F_limit,
    depthlim([Node|Path], Node1, G1, F_limit, Sol, G2).
```

This replacement is based on the function of search cost, which is:

$$f(n) = (2-\omega)g(n) + \omega h(n)$$

It means that when ω has been changed to 1.2, the $2-\omega$ should be changed to 0.8.

Section d):

Greedy Search is the most efficient as it tries to find the least search cost in each step, but the quality of its solution is the least optimal, as it uses the most steps to solve the puzzle.

Iterative Deepening Search is the least efficient, as it generates the most nodes, however, the solution can be optimal compared to others'.

For other 3 heuristic methods, when heuristic varies, the speed and the quality will also be changed. From this table we can infer that when ω increases, the method will be more like Greedy Search, thus, the speed will increase. However, the quality of solutions will become less optimal which is the tradeoff of different heuristics.

Question 3:

Section a), an admissible heuristic is shown below:

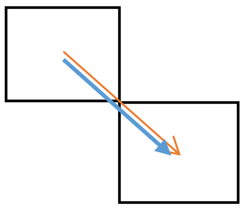
$$h(x, y, x_G, y_G) = |x - x_G| + |y - y_G|$$

This heuristic is based on Manhattan Distance, calculating the absolute difference between two points in two axes respectively, therefore, the distance will no less than the Straight-Line-Distance. When obstacles added in maze, using this heuristic will extend the length of the solution path, but still a solution will be generated. For this reason, it is admissible, but it dominates Straight-Line-Distance because the solution will be always longer than that.

As this heuristic avoids computation of square roots or powers and it will expand less nodes than Straight-Line-Distance heuristic, this heuristic can be better when used in maze.

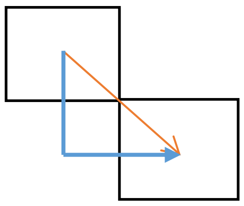
Section b) (i):

No. A heuristic is admissible when it will not over-estimate the cost than the lowest possible cost. The Straight-Line-Distance heuristic will always calculate actual distance between two points. But the definition says when moving diagonally, the cost is still considered as one step. Suppose the goal is one diagonal step away, the lowest possible cost is one, but the estimated cost will be $\sqrt{2}$ (although same path). As Straight-Line-Distance over-estimate the cost, it should not be admissible.



Section b) (ii):

No. It is similar as above. The Manhattan-Distance heuristic will always calculate absolute difference between two points with coordinate X and coordinate Y. But the definition says when moving diagonally, the cost is still considered as one step. Suppose the goal is one diagonal step far away, the lowest possible cost is one, but the estimated cost will be 2. As Manhattan-Distance over-estimate the cost, it should not be admissible.



Section b) (iii):

The admissible heuristic is shown below:

$$h(x, y, x_G, y_G) = \max(|x - x_G|, |y - y_G|)$$

This heuristic is known as Chebyshev-Distance, which means it counts the maximum of coordinate differences (either X or Y) when moving. Therefore, even the agent moves diagonally, the distance will also be counted as one. Thus, this heuristic is admissible.

Question 4:

(a): When $k = 0$, the $M(n, 0)$ for $1 \leq n \leq 21$ is computed shown in the table as below:

n	The Optimal Sequence	+	o	-	$M(n, 0)$
1	+ -	1	0	1	2
2	+ o -	1	1	1	3
3	+ o o -	1	2	1	4
4	+ + - -	2	0	2	4
5	+ + - o -	2	1	2	5
6	+ + o - -	2	1	2	5
7	+ + o - o -	2	2	2	6
8	+ + o o - -	2	2	2	6
9	+ + + - - -	3	0	3	6
10	+ + + - - o -	3	1	3	7
11	+ + + - o - -	3	1	3	7
12	+ + + o - - -	3	1	3	7
13	+ + + o - - o -	3	2	3	8
14	+ + + o - o - -	3	2	3	8
15	+ + + o o - - -	3	2	3	8
16	+ + + + - - - -	4	0	4	8
17	+ + + + - - - o -	4	1	4	9
18	+ + + + - - o - -	4	1	4	9
19	+ + + + - o - - -	4	1	4	9
20	+ + + + o - - - -	4	1	4	9
21	+ + + + o - - - o -	4	2	4	10

(b): The explanation is given below:

When $n \geq 0$ and $k = 0$, we assume $M(n, 0)$ could follow the identity:

$$M(n,0) = \lceil 2\sqrt{n} \rceil = \begin{cases} 2s+1 & if\ s^2 < n \leq s(s+1) \\ 2s+2 & if\ s(s+1) < n \leq (s+1)^2 \end{cases}$$

The s is defined as the maximum speed (the number of '+' symbols).

For the table in (a), it can be concluded that the agent will not rest at the goal points, as it will move as fast and stop with no velocity at the goal points. Thus, the rest points can only show between accelerating points and goal points. And, as for the rest points can be one or two, it is clear that:

- Rest points show after accelerating points, before goal points.
- For every $n = a^2$ $(a \in N\ and\ a \geq 2)$, $M(n, 0) = M(n-1, 0)$. For example, when $n = 4\ and\ n = 3$, the $M(4, 0) = M(3, 0) = 4$. This is because changing one 'o' to '+' cancels out changing one 'o' to '-'.
- The s is adding up one for every n square (similar as above statement).

As the third statement shows, this identity can be proved that when n is less than $(s+1)^2$, which means 'before adding '+' symbol', the $M(n, 0)$ can have two rests, which is $2s + 2$. Also, as the second statement shows, the time adding '+' symbol is equal to the previous time, thus, $n \leq$

$(s+1)^2$. And, it can be inferred from the first statement, adding rest points from one rest to two rests is before $s(s+1)$, because otherwise the goal point will be a rest, or, the change will repeat.

(c): The explanation is given below:

When $n \geq \frac{k(k-1)}{2}$ and $k \geq 0$, we assume $M(n,k)$ could follow the identity:

$$M(n,k) = \left\lceil 2\sqrt{n + \frac{k(k+1)}{2}} \right\rceil - k$$

To prove this, the total distance can be divided into the first half and the second half. As for the condition, when $n \geq \frac{k(k-1)}{2}$, then $2n \geq k(k-1)$. This means agent will stop at or before the goal. We set up the second half distance $M(n,0)$. When extra velocity $k \geq 0$ will shorten the time to accelerate, it can have this identity:

$$M(x,0) = \lceil 2\sqrt{x} \rceil - k$$

Consider the extra distance part of x for accelerating velocity up to k (make it run to the goal), we have the summation function as:

$$\frac{k(k+1)}{2}$$

This is the extra distance, therefore, replace x as $n + \frac{k(k+1)}{2}$, the identity should be:

$$M(n,k) = \left\lceil 2\sqrt{n + \frac{k(k+1)}{2}} \right\rceil - k$$

(d): The explanation is given below:

Similar as (c), when the $n < \frac{k(k-1)}{2}$, it means that agent will stop after the goal and try to reverse.

The total time before starting to reverse is:

$$\left\lceil 2\sqrt{\frac{k(k+1)}{2} + \frac{k(k-1)}{2}} \right\rceil = 2k$$

And the reverse time should be:

$$\left\lceil 2\sqrt{\frac{k(k-1)}{2} - n} \right\rceil$$

Then sum up this equation, and minus the accelerating time, the identity should be:

$$2k + \left\lceil 2\sqrt{\frac{k(k-1)}{2} - n} \right\rceil - k = \left\lceil 2\sqrt{\frac{k(k-1)}{2} - n} \right\rceil + k$$

(e): The admissible heuristic is shown below:

$$h(r, c, u, v, r_G, c_G) = max(M(r_G - r, u), M(c_G - c, u))$$

when $M(r_G - r, u) = \left\lceil 2\sqrt{r_G - r + \frac{u(u+1)}{2}} \right\rceil - u$

and $M(c_G - c, u) = \left\lceil 2\sqrt{c_G - c + \frac{u(u+1)}{2}} \right\rceil - u$

The reason for proving its admissibility is that it only considers the maximum difference between the two coordinates either r or c, which means even the agent take diagonal distances, this heuristic is still admissible as it counts diagonal distances as straight-line ones, calculating them as normal.