1.

(a)

	start10	start20	start27	start35	start43
UCS	G = 10 N = 2565	Mem	Mem	Mem	Mem
IDS	G = 10 N = 2407	G = 20 N = 5297410	Time	Time	Time
A*	G = 10 N = 33	G = 20 N = 915	G = 27 N = 1873	Mem	Mem
IDA*	G = 10 N = 29	G = 20 N = 952	G = 27 N = 2237	G = 35 N = 215612	G = 43 N = 2884650

(b)

The above table compares the runtime of four complete and optimal path finding algorithm. We can see that uninform searches are all very inefficient (UCS and IDS) due to expanding too much useless nodes while informed search algorithms (A* and IDA*) expands less nodes by considering the heuristics. However, compared with IDA* which has similar space complexity with IDS, A* would store the entire level of states in the memory and would cause memory limit exceeded in large inputs. The time and space complexity could be described as follows (let b be the branching factor and d be the depth).

	time complexity	space complexity
UCS	$O(b^d)$	$O(b^d)$
IDS	$O(b^d)$	$O(bd)$
A*	depending on the heuristics, exponential in relative error h * length of solution, worst case $O(b^d)$	depending on the heuristics, keeps all nodes in memory, worst case O(V) = $O(b^d)$
IDA*	depending on the heuristics, exponential in relative error h * length of solution, worst case $O(b^d)$	$O(bd)$

2.

(a)

For start49:

S = [4/1,2/2,1/2,1/4,1/3,3/3,3/2,4/4,2/4,3/4,3/1,2/3,2/1,1/1,4/3,4/2].

H = 25.

For start51:

S = [2/1,3/4,4/2,2/4,4/4,3/1,4/1,1/1,3/3,1/4,1/3,1/2,4/3,2/2,2/3,3/2].

H = 43.

(b)

G = 51 and the number of nodes being expanded is 551168.

(c)

The performance of the searching algorithms based on heuristics are affected by the quality of the heuristics. Since when the program runs on these two starting points different heuristics are used which start49's heuristic much less than start51's heuristic. Lower heuristics would result in larger searching space for A* (and IDA*) algorithms, and hence a worse running time and space for start49.

3.

(a)(b)(c)

	start49		start60		start64	
IDA*	49	178880187	60	321252368	64	1209086782
1.2	51	988332	62	230861	66	431033
1.4	57	311704	82	4432	94	190278
Greedy	133	5237	166	1617	184	2174

The heuristic.pl changing part looks as follows (in bold):

```
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, % in part c change it to F1 is 0.6 * G1 + 1.4s * H1
   F1 =< F_limit,
   depthlim([Node|Path], Node1, G1, F_limit, Sol, G2).
```

(d)

Note that the only difference between these algorithms is the weight of the heuristics. F(n) = (2-w) G(n) + w*H(n). For the original IDA*, w = 1 which will always produce the optimal solution while w = 1.2, w = 1.4 and w = 2(greedy approach) all overestimate the heuristics and cannot produce the

optimal solution. It is not difficult to observe that the more an algorithm overestimate the heuristic, the less accurate the solution but the faster the algorithm would run.

4.

(a) The Manhattan distance heuristics: $h(x, y, xG, yG) = |xG - x| + |yG - y|$.

(b)

(i) No, consider (n+1)*(n+1) square that requires to move from (0, 0) to (n, n), the heuristic of the straight line distance is $\sqrt{2}n$, while the actual cost to move might be just n by moving through the diagonal. Which the heuristic could be overestimated and hence it is not admissible.

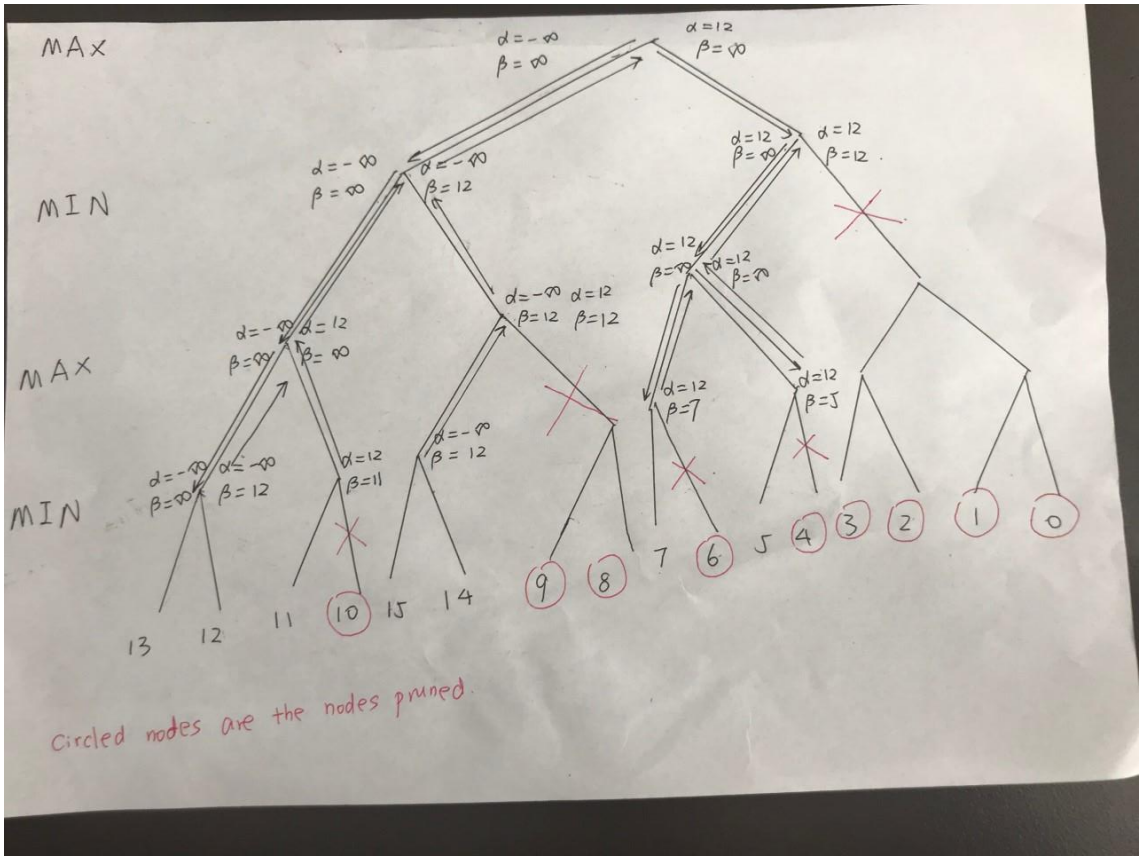
(ii)No, similar to part(i), when you consider the Manhattan distance moving from (0, 0) to (n, n), the heuristic is $2n$, while the actual cost to move might be just n by moving through the diagonal. Which the heuristic could be overestimated and hence it is not admissible.

(iii) We shall consider the shortest move possible from (x,y) to (xG, yG).

$$h(x, y, xG, yG) = \max(|xG - x|, |yG - y|)$$

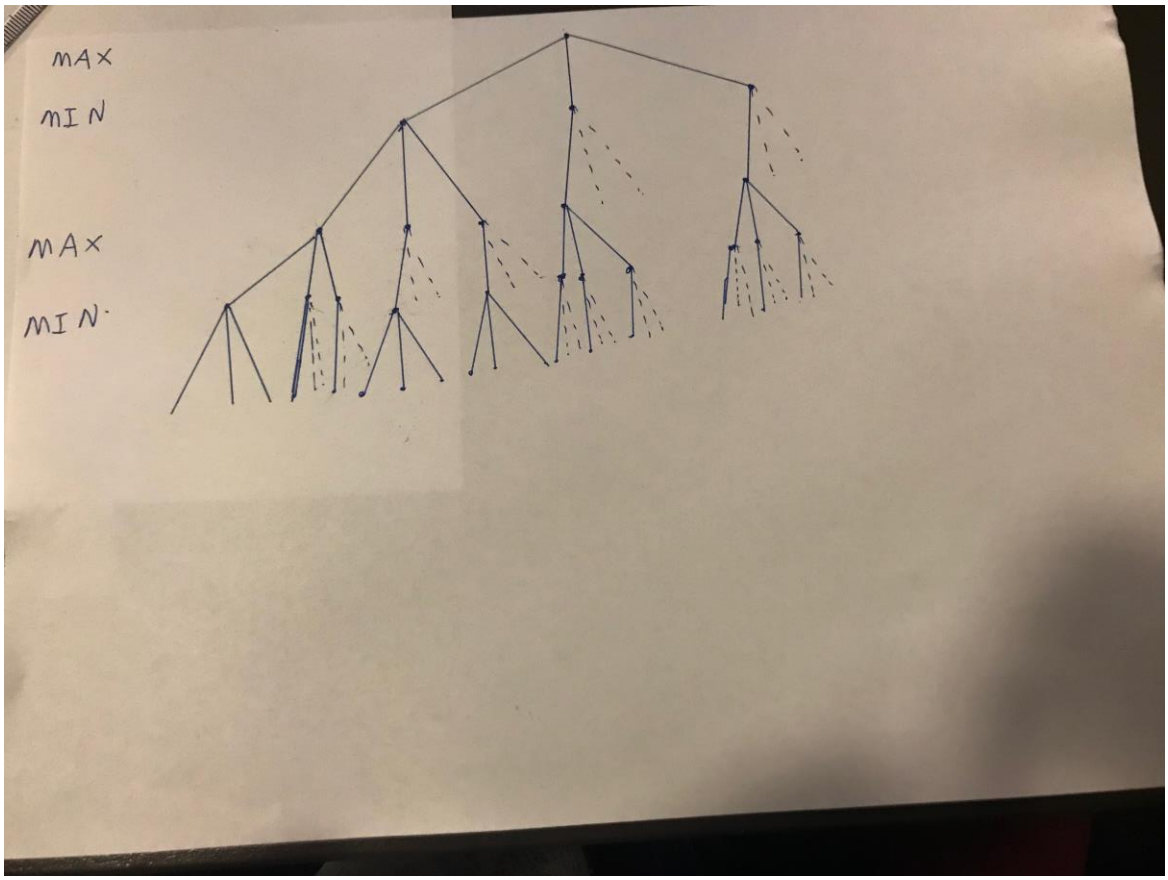
5.

(a)(b) In total 7 nodes are evaluated.



(c)

The pattern looks as follows:



Total of 17 nodes are evaluated

(d)

Best case of the alpha-beta search algorithm is O(b^(d/2)) which b is the branching factor and d is the depth of the search tree. From part(c) hint, we can see that for a search tree of depth d, the best case happens when every PV nodes would search all its child and non PV nodes would switch between search all its nodes and explores only one child, hence approximately in the best case time complexity is O(b*1*b*1*b*…) = O(b^(d/2)).