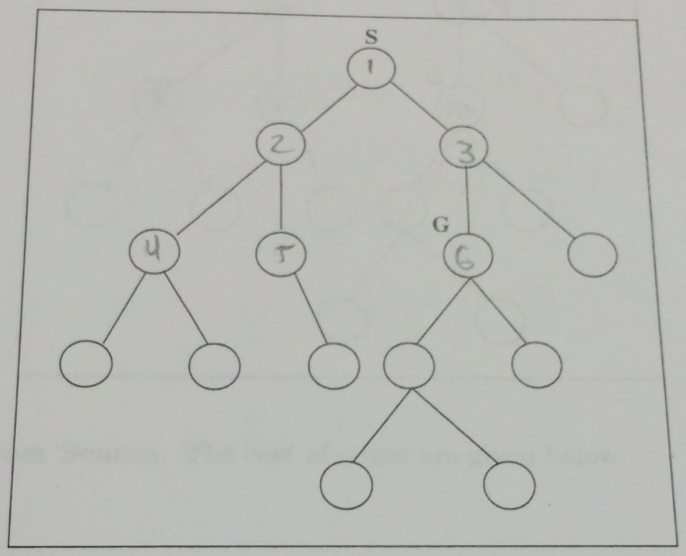
Name	ID Number
Andrew Shih	

Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Total
20	10	8	10	15	12	5	10	90

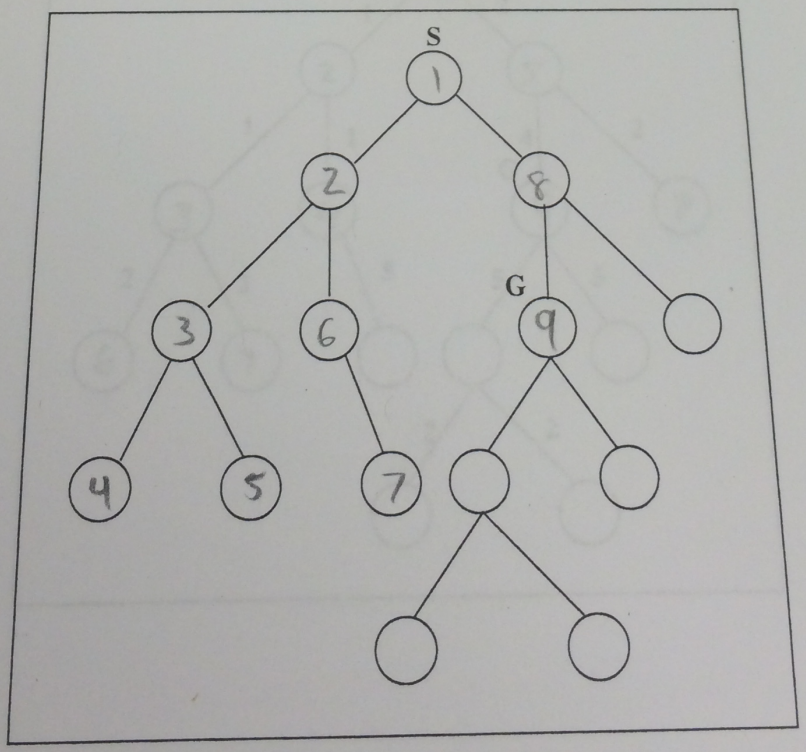
	BFS	DFS	
time	b^d	b^m	b^{cl}
space	b^d	b^m	bcl

1. (20pts) You are given a search tree with one node labeled as a start state **S** and another node labeled as a goal state **G**. Number the nodes in the tree according to the order in which they will be expanded (not the order in which they are generated). When the order of expansion is arbitrary, assume that nodes are expanded from left to right.

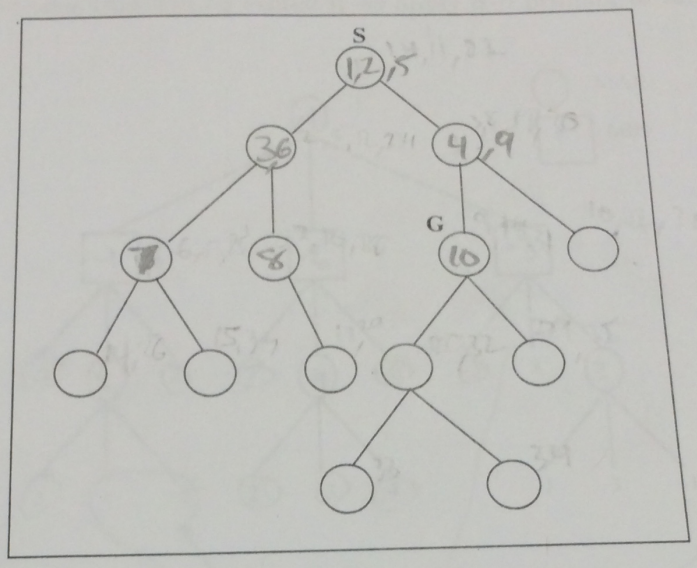
5pts **Breadth First Search.**



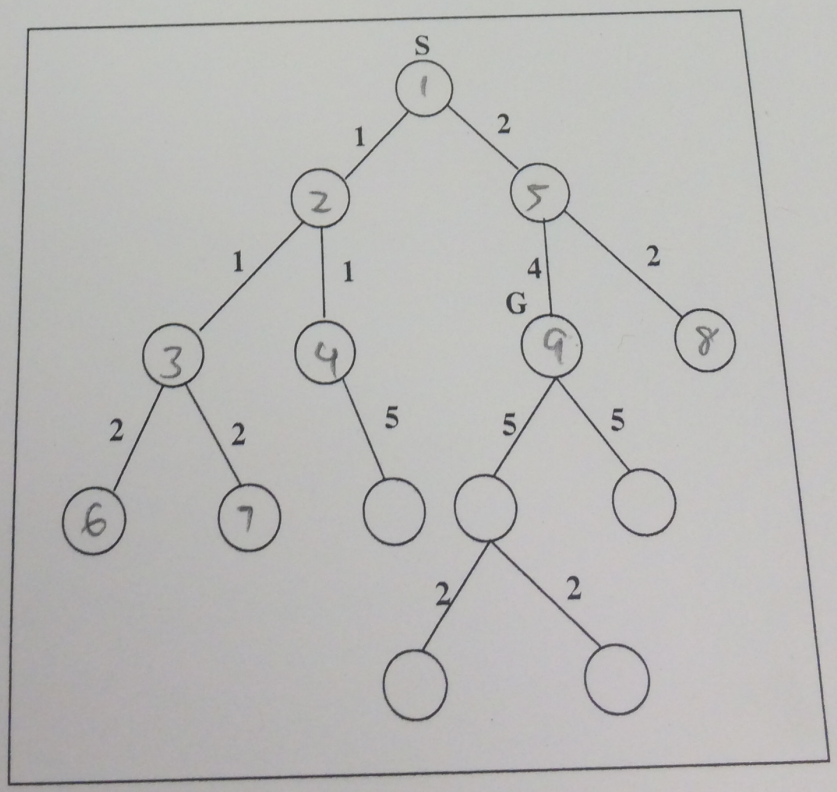
5pts **Depth First Search.**



5pts **Iterative Deepening Search.**

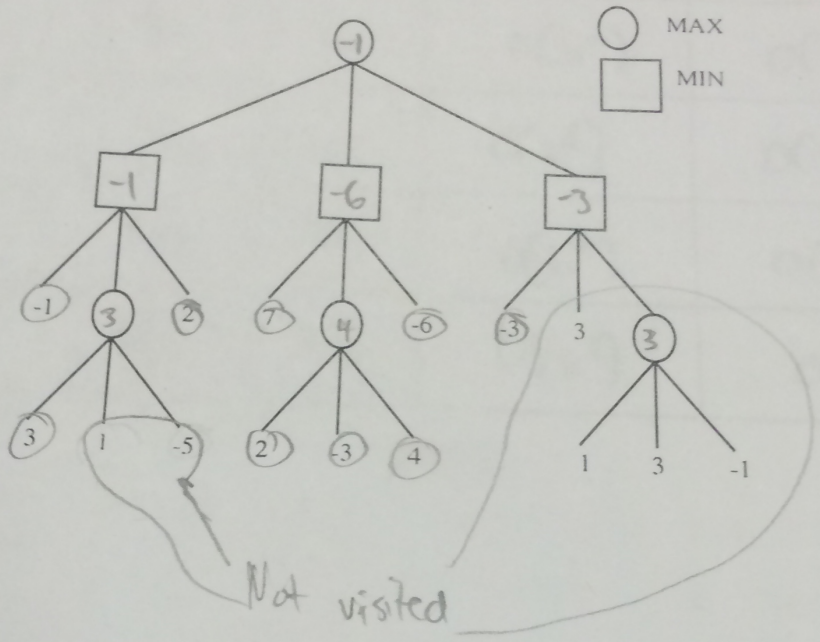


5pts **Uniform Cost Search.** The cost of edges are given below.



2. *(10 pts)* Show the utility of each node in the following game tree as computed by the MINIMAX algorithm. Mark the nodes that will be visited if we apply α-β pruning to the tree.

O = visited



3. (10 pts) Fill the following table, assuming that b is the branching factor, d is the solution depth, m is the search tree depth, and l is the depth limit.

Strategy	*Complete?*	*Optimal?*	*Time Complexity*	*Space Complexity*
Depth First	No	No	$O(b^m)$	$O(b^m)$ -1
Breadth First	Yes	Yes	$O(b^d)$	$O(b^d)$
Iterative Deepening	Yes	Yes	$O(b^d)$	$O(bd)$
Depth-Limited	No	No	$O(b^l)$	$O(b^l)$ -1

4. (12 pts) Consider a situation where we have n blocks $B_1, \ldots, B_n$ and a table T. A block can be located on top of another block or on the table. The table has enough room for all blocks. Initially, blocks $B_3, \ldots, B_n$ are on the table; block B_1 is on block B_2 which is on block B_3. We have a robot that can move blocks around. That is, it can move any block B_i from its current location to the table or to the top of any other block B_j, as long as blocks B_i and B_j are clear. The goal is to define a search problem the solution of which will be a set of actions that if carried out by the robot will rearrange the blocks into a stack in which: block B_n is on the table, and each block $B_i, 1 \leq i < n$, is on top of block B_{i+1}.

a 3pts Define the state space, the initial state and the final state for this search problem.

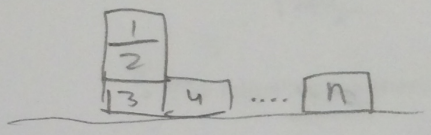
b 3pts Define the set of operators. For each operator instance, indicate the states to which it can be applied and the state it leads to.

c 3pts Give a tight upper bound on the branching factor for the search space in terms of n.

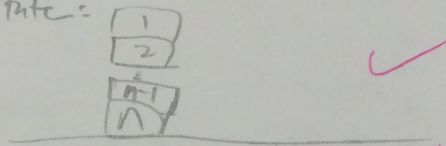
d 3pts Assume that we will use A^* to conduct the search. Provide an admissible heuristic function for this problem (other than the trivial one of $h = 0$). Explain why the heuristic is admissible.

a. state space: n blocks occupy an nxn grid where one block must be on the base for another block to occupy the next level ✓

initial state:



final state:



✓

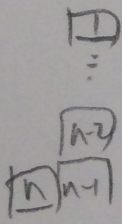
b. Operators: Move block from top to table (can be applied to a state that not all boxes are on table)

Move block from table to top (can be applied to a state that not all boxes are stacked)

c. Branching factor: # of blocks on table which is? -1 (n^2)

d. heuristic = # of boxes stacked in which base is not B_n and block on top has lower block # than the one below it

This is admissable because it will not overestimate. The max value is n-1 and that is when it is in the state:



which it takes at least 2(n-1) moves to solve

(21 pts) True or false? 15

- Local search algorithms are complete.

False

- Backtracking search is a type of depth-first search.

True

- Arc consistency can be applied in $O(n^2 d)$ time, where n is the number of variables and d is the number of values per variable.

False

- Tree-structured CSPs can be solved in time linear in the number of CSP variables.

False -3

- The following inference rule is sound:

$$\frac{\neg\alpha,\ \beta}{(\alpha \Rightarrow \gamma) \wedge (\neg\beta \Rightarrow \gamma)}.$$

True

α	β	γ	$\alpha \Rightarrow \gamma$	$\neg\beta \Rightarrow \gamma$	$\alpha \Rightarrow \gamma \wedge \neg\beta \Rightarrow \gamma$
0	0	0	1	0	0
0	0	1	1	1	1
0	1	0	1	1	1
0	1	1	1	1	1
1	0	0	0	0	0
1	0	1	1	1	1
1	1	0	0	1	0
1	1	1	1	1	1

- Resolution is a complete inference rule.

True -3

- Local search algorithms for CSPs can be used to prove $\Delta \models \alpha$ using the technique of proof by contradiction.

False

6. (12 pts) If a course is easy, some students are happy:

$$easy\_course(E) \Rightarrow some\_students\_are\_happy(H).$$

If a course has a final, no students are happy.

$$course\_has\_final(F) \Rightarrow \neg some\_students\_are\_happy(\neg H).$$

Show that that if a course has a final, the course is not easy:

$$course\_has\_final(F) \Rightarrow \neg easy\_course(\neg E).$$

That is, show that the first and second sentences above entail the third one. Show this using two different methods:

(a) (6 pts) Truth tables.

(b) (6 pts) Resolution.

a.

E	F	H	E⇒H	F⇒¬H	F⇒¬E	(E⇒H)∧(F⇒¬H)	((E⇒H)∧(F⇒¬H))⇒(F⇒¬E)
f	f	f	t	t	t	t	t
f	f	t	t	t	t	t	t
f	t	f	t	t	t	t	t
f	t	t	t	f	t	f	t
t	f	f	f	t	t	f	t
t	f	t	t	t	t	t	t
t	t	f	f	t	f	f	t
t	t	t	t	f	f	f	t

all t
true f
all wo

b. KB:

1. $E \Rightarrow H$
2. $F \Rightarrow \neg H$
3. $\neg E \vee H$
4. $\neg F \vee \neg H$

α: $F \Rightarrow \neg E$

5. $\neg(F \Rightarrow \neg E)$
6. $\neg(\neg F \vee E)$ (5)
7. F (6)
8. $\neg E$ (6)
9. $\neg H$ (4,6)
10. $\neg E$ (3,9)

contradict = valid ✓

(5 pts) Convert the following sentence into CNF:

$$(A \Leftrightarrow B) \vee \neg(C \wedge D).$$

Is the resulting set of clauses Horn? Why?

(5)

$((A \Rightarrow B) \wedge (B \Rightarrow A)) \vee (\neg C \vee \neg D)$ ✓

$((\neg A \vee B) \wedge (\neg B \vee A)) \vee (\neg C \vee \neg D)$ ✓

$(\neg A \wedge \neg B) \vee (A \wedge B) \vee \neg C \vee \neg D$

$(\neg A \vee B \vee \neg C \vee \neg D) \wedge (A \vee \neg B \vee \neg C \vee \neg D)$ ✓

Yes, a set of Horn clauses because it only contains one positive literal in each clause (B in first, A in second) ✓

8. (10 pts) Write a single LISP function SUB-SEQUENCE, which takes a list SEQ, two non-negative integers P and L, and returns in reverse order the sub-list of SEQ which starts at position P and has length L (the first element in SEQ has position 0). For example, (SUB-SEQUENCE '(a b c d) 1 2) returns (c b), and (SUB-SEQUENCE '(a b c d) 0 3) returns (c b a). If there are less than L elements starting at position P, then those elements should be returned. For example, (SUB-SEQUENCE '(a b c d) 2 5) should return (d c). Also, if P is not a valid position, then NIL should returned. For example, (SUB-SEQUENCE '(a b c d) 10 2) should return NIL. You can use cond, cons, car, cdr, null, list, append in addition to arithmetic operations and functions. Your function should be as efficient as possible.

Hint: Write a function for returning the sub-list in normal order, and then modify it to return the result in reverse order.

10

```
(defun SUB-SEQUENCE (SEQ P L)
  (cond ((null SEQ) NIL) ((= L 0) NIL)
        ((> P 0) (SUB-SEQUENCE (cdr SEQ) (- P 1) L)
        (T (append (SUB-SEQUENCE (cdr SEQ) P (- L 1)) (list (car S
```

Cleaner version of above:

```
(defun SUB-SEQUENCE (SEQ P L)
  (cond ((null SEQ) NIL)
        ((= L 0) NIL)
        ((> P 0) (SUB-SEQUENCE (cdr SEQ) (- P 1) L))
        (T (append (SUB-SEQUENCE (cdr SEQ) P (- L 1)) (list (car SEQ)) ...
```