Q1(a) in 11.2

For any binary tree of height h and number of leaves ℓ, we have as shown below, by induction, that $2^h \geq \ell$, for $h \geq 0$

- For the trivial case $h=0$, $2^0 \geq 1$.

- Now, assume for the inductive step that $2^h \geq \ell$ holds for any binary tree with height at most h. We need to show that any binary tree T with height $h+1$ satisfies the above ineq.

Let T_L & T_R denote the left and right subtrees of T, respectively. The heights of T_L and T_R are at most h and the number of leaves in T is equal to the number of leaves in T_L plus the number of leaves in T_R. We have $\ell(T_L) \leq 2^h$ and $\ell(T_R) \leq 2^h$.

Thus, we obtain the following ineq. for the number of leaves $\ell(T)$ for the tree T:

$$\ell(T) = \ell(T_L) + \ell(T_R) \leq 2^h + 2^h$$

$$\boxed{\ell(T) \leq 2^{h+1}}$$

(end of proof by induction)

- taking bin logarithm of ineq. $\ell \leq 2^h$, we get

$$\log_2 \ell \leq h$$

- Since h is an integer, $h \geq \lceil \log_2 \ell \rceil$

Q.E.D.

Q2. in 11.1

- The following is a proof by induction that any algorithm solving the problem must make a number of disk moves $M(n)$ that is at least equal to $2^n - 1$, where n is the number of disks.

That is, $M(n) \geq 2^n - 1$ for $n \geq 1$ ... (1)

- First, for the trivial case $n = 1$, $M(1) = 2^1 - 1 = 1$. Thus, ineq (1) holds.

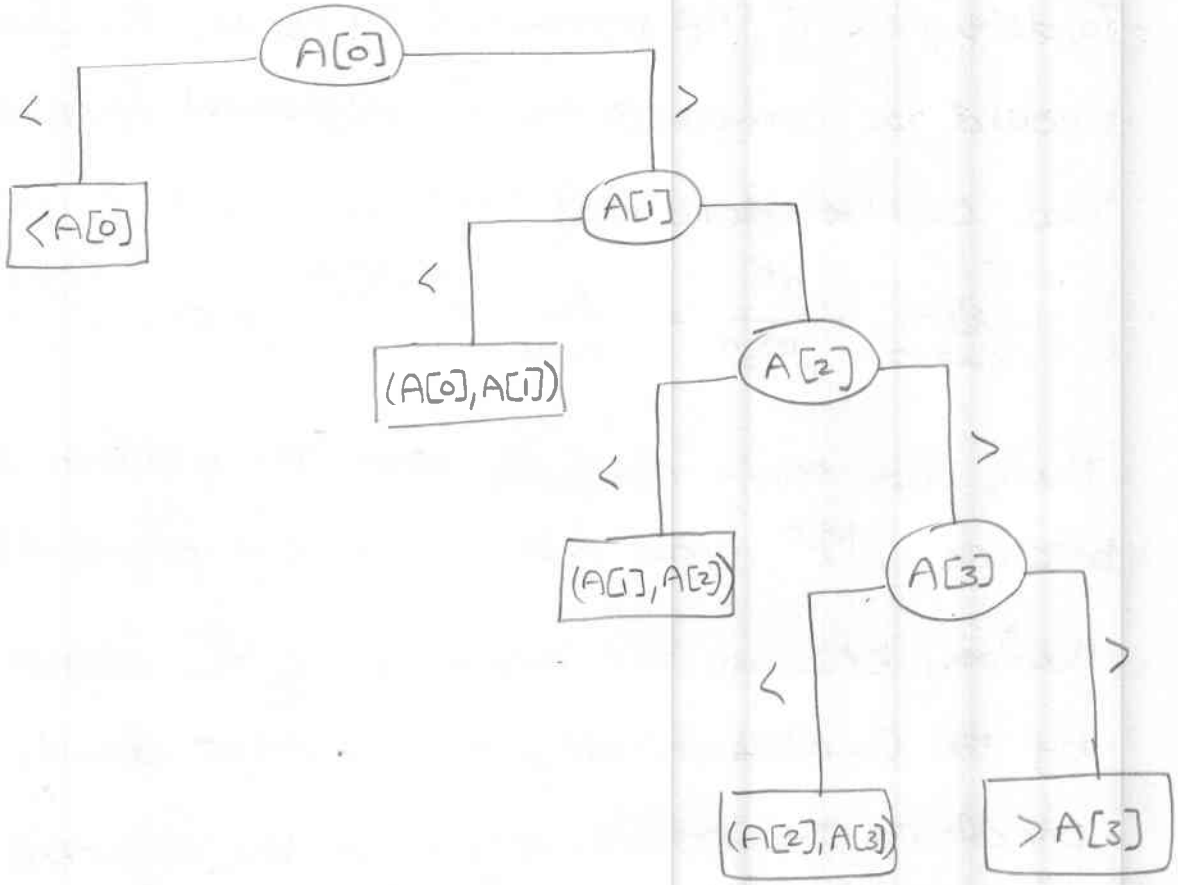
- Assume that ineq. (1) holds for $n \geq 1$ disks and show that the case of $n+1$ disks follows from the inductive assumption.

-Before the largest disk can be moved, all n smaller disks must be transferred to another peg. By assumption, this required at least $2^n - 1$ moves. Moving the largest disk will take at least one move. After moving the largest disk to the destination peg, the smaller n pegs will need to be moved on top of it, which will require at least $2^n - 1$ moves, by the inductive assumption. Thus,

$$M(n+1) \geq 2^n - 1 + 1 + 2^n - 1 = 2^{n+1} - 1$$

Q.E.D

Q6. in 11.2



Q 1. in 11.3

In principle, one can generate all the sequences of legal moves of both sides from the given position of chess pieces and determine whether one of them is a win for the side that is to move next (from the given position).

So, the problem is theoretically decidable.

Q 2. in 11.3

- To determine if the problem is tractable, the function $n^{\log_2 n}$ should be compared to a polynomial function n^k.

This can be done using limits.

$$\lim_{n\to\infty} \frac{n^k}{n^{\log_2 n}} = \lim_{n\to\infty} n^{k-\log_2 n} = 0$$

Thus, we can't conclude that the problem is tractable because $n^{\log_2 n}$ grows faster than any polynomial fn n^k.

However, because the run time of the algorithm is given by the O-notation rather than a tight bound, we can't preclude the possibility that the algorithm may be polynomial, or whether there exists an algorithm which is polynomial for this problem.

Thus, we can't make assertion (a) or (b), so, the correct answer is (c).

Q7. in 11.3

(a) The decision version of the knapsack problem can be stated as follows. Given a set of n items with weights w_i and values v_i for $i = 1, \ldots, n$ and a knapsack capacity W, determine whether there is a subset of the n items that fits into the knapsack and has a total value of at least a given positive integer m. ($v \geq m$).

To verify a proposed solution (a guess), we need to sum up the weights and sum up the values of the proposed subset, check that the weight sum doesn't exceed the knapsack's capacity, and that the value sum is not less than m.

```
Alg checkKnapsack (S, w, v, W, m)
    weightSum = 0, valueSum = 0
    for i = 1 to |S|
        weightSum += w[S[i]]
        valueSum += v[S[i]]
    endfor
    If weightSum ≤ W and valueSum ≥ m
        return YES
    endIf.
```

- The run time efficiency of this algorithm is $\Theta(n)$, since the max number of elements in S is n.

Q 8. in 11.3

The partition problem is stated as: Given n +ve int. $s_1, \ldots, s_n$, determine whether it is possible to partition them into two disjoint subsets with the same sum.

For the integers s_i, $i = 1 \ldots n$, let x_i, $i = 1 \ldots n$ denote corresponding binary 0-1 variables. The partition problem can be expressed as a selection of variables x_i's such that

$$\sum_{i=1}^{n} x_i s_i = \frac{1}{2} \sum_{i=1}^{n} s_i$$

Equivalently, we have

$$2 \sum_{i=1}^{n} x_i s_i = \sum_{i=1}^{n} s_i \text{ , which can be written as}$$

$$\sum_{i=1}^{n} x_i 2 s_i = \sum_{i=1}^{n} s_i$$

Let $w_i = v_i = 2s_i$, $W = \sum_{i=1}^{n} s_i$ and $m = W$ for the decision version of the knapsack problem. Thus, we obtain the following equivalent instance of the knapsack problem:

$$\sum_{i=1}^{n} x_i w_i \geq m \quad (m = W)$$

$$x_i \in \{0, 1\} \text{ for } i = 1, \ldots, n$$

$$\sum_{i=1}^{n} x_i w_i \leq W$$

Thus, an arbitray instance of the partition problem can be transformed in $\Theta(n)$ time into an instance of the knapsack problem (decision version).

A YES answer to transformed instance as a knapsack problem occurs only when $\sum_{i=1}^{n} x_i 2s_i = m$, which corresponds to a YES instance of the given instance of the partition problem.

Conversely, if the given instance of the partition problem satisfies that $\sum_{i=1}^{n} x_i s_i = \frac{1}{2}\sum_{i=1}^{n} s_i$ (i.e. is a YES instance), the new transformed instance of the knapsack has

$$\sum_{i=1}^{n} x_i s_i = m \quad \text{and} \quad \sum_{i=1}^{n} x_i w_i = W.$$

Hence, it represents a YES instance of the knapsack problem.

Therefore, we have a YES instance for knapsack iff we have a YES instance of the partition problem.

$\therefore$ partition Problem $\leq_p$ knapsack Problem.