1. (5 points) Show the final B-tree that results from inserting the keys A, B, C, D, E, F, G, H, and I (in this order) into an empty B-tree, where minimum degree $t = 3$.

```
       C
      / \
     F   \
    /     \ 
   A   B   D E G H I
```

2. (3 points) Show the B-tree that results from deleting X from the following B-tree, where minimum degree $t = 3$.

```
       U
      /|
     X S T V W Y Z
```

This is Case 2c.
3. (2 points) Circle the answer below that is closest to the maximum number of keys that can be stored in a B-tree of height 2 and minimum degree $t = 512$.

(a) 1,000,000
(b) 1,000,000,000
(c) 1,000,000,000,000

The correct choice is (b). The maximum number of keys per node is $2t - 1 = 2(512) - 1 = 1023$. So, the root node can have 1023 keys and 1024 children. Each of these children can have 1023 keys for a total of $1024 \times 1023$ keys at height 1. This leads to $1024 \times 1024$ children at height 2, each having 1023 keys, for a total of $1024 \times 1024 \times 1023$ keys. Adding up the keys from each height results in a total of 1,073,741,823 keys.

4. (2 points) Define what it means for a problem to exhibit optimal substructure.

The optimal solution to the problem consists of optimal solutions to the subproblems.

5. (1 point) Besides optimal substructure, what is the other main element of dynamic programming?

Overlapping subproblems

6. (2 points) In addition to a problem exhibiting optimal substructure, what two characteristics must a greedy choice possess to yield an optimal solution (i.e., the two aspects of proving the greedy choice property)?

The greedy choice is in an optimal solution, and the greedy choice can be made first.
7. Consider the problem of computing Fibonacci numbers, where the Fibonacci number $F(n)$ of $n$ is defined as follows.

$$F(n) = \begin{cases} 1 & n < 3 \\ F(n - 1) + F(n - 2) & n > 3 \end{cases}$$

Recall from class that the straightforward implementation of a recursive algorithm to compute $F(n)$ is $\Omega(2^n)$. We want to design a $O(n)$ dynamic programming solution to this problem.

(a) (3 points) Draw the binary tree representing the recursive calls made by the recursive algorithm and indicate identical subproblems. The root of the tree is $F(n)$, the root’s two children are $F(n - 1)$ and $F(n - 2)$, and so on. Draw the tree deep enough to identify at least two different, overlapping subproblems.

![Binary Tree Representation](image)

(b) (2 points) In terms of $n$, how many unique subproblems are there for computing $F(n)$?

There are $n$ unique subproblems: $F(n), F(n - 1), \ldots, F(1)$.

(c) (5 points) Give pseudocode for a $O(n)$ algorithm $\text{FIBONACCI}(n)$ that uses a bottom-up, iterative design to compute and return $F(n)$.

```plaintext
FIBONACCI(n)
1 Allocate array $F[1 \ldots n]$
2 $F[1] = 1$
3 $F[2] = 1$
4 for $i = 3$ to $n$
5 $F[i] = F[i - 1] + F[i - 2]$
6 return $F[n]$
```