1. Consider the following algorithm for sequential search in an array $A$ of size $n$:

```python
def sequential_search(A, n, x):
    for i = 0 to n-1:
        if A[i] == x:  return i
    return -1
```

Assume that the $n$ elements of $A$ are distinct.

In each part of this question we make an assumption about the probability distribution of the presence and location of $x$ in the array. For each part, compute the expected number of times the comparison “if $A[i] = x$ . . .” is executed if the given assumptions hold.

(a) The item $x$ is in the array. It is equally likely to be in any of the $n$ locations in the array.

(b) The probability that $x$ is in the array is 0.5. If it is in the array, it is equally likely to be in any of the $n$ locations in the array.

2. In class, we discussed the problem of searching for an item $x$ in a sorted array $L$ with $n$ entries. The returned value is supposed to be -1 if the value $x$ is not in the array, and the index $i$ such that $L[i] = x$ if $x$ is in the array. Assume that $L$ does not contain duplicate values.

Now consider the decision trees of algorithms for solving this problem. To make the decision tree more explicit, we can augment the tree with square nodes, which mean that a value of -1 is returned, indicating that $x$ does not appear in the array $L$. The decision tree for binary search with $n = 13$, augmented in this fashion, is shown below.
Below, you are given two algorithms for searching in a sorted array. Each function is a minor modification of the binary search algorithm discussed in class.

**Algorithm 1**

```plaintext
integer function BS1(n, L, x);
integer first, last;
begin {BS1}
  first = 0;
  last = n - 1;
  while first < last do
    index = ⌊(first + last)/2⌋;
    if x == L[index] then return(index);
    else if x < L[index] then
      last = index - 1
    else first = index + 1;
  return(-1);
end {BS1}
```

**Algorithm 2**

```plaintext
integer function BS2(n, L, x);
integer first, last;
begin {BS2}
  first = 0;
  last = n - 1;
  while first ≤ last do
    index = ⌊(first + last)/2⌋;
    if x == L[index] then return(index);
    else if x < L[index] then
      last = index
    else first = index + 1;
  return(-1);
end {BS2}
```

(a) Draw the decision tree for Algorithm 1 with \( n = 7 \). If the tree has more than four levels, you only need to draw the first four levels (i.e., levels 0, 1, 2, and 3, where 0 is the root level).

(b) Based on your answer to (a), comment on the correctness of Algorithm 1.

(c) Draw the decision tree for Algorithm 2 with \( n = 7 \). If the tree has more than four levels, you only need to draw the first four levels (i.e., levels 0, 1, 2, and 3, where 0 is the root level).

(d) Based on your answer to (c), comment on the correctness of Algorithm 2.