CPSC 5031:
Data Structures & Algorithms

Lecture 1: Math Review

(Sequences, Summations, Induction, and Recursion)

Sequences

- sequence: A function from a subset of the set of integers to a set S.
- term: A single entity of the sequence; the value from the codomain assigned to n, denoted using a_n.
- The subset of integers is usually {0, 1, 2, …} or {1, 2, 3, …}. Often, such sequences are listed in ascending order.
- Example: {1, 3, 5, … } where $a_0 = 1$, $a_1 = 3$, $a_2 = 5$, …, $a_n = 2n+1$. We will assume the former (the first term is a_0) unless specified otherwise. The book likes to use both so be careful.
- geometric progression: A sequence of the form a, ar, ar^2, …, ar^n where a is the initial term and r is the common ratio.
- arithmetic progression: A sequence of the form a, $a + d$, $a + 2d$, …, $a + nd$ where a is the initial term and d is the common difference.

Sequences

- Example: Determine if the sequence is a geometric progression, arithmetic progression, or neither.

a. 3, 6, 12, 24, 48, …
geometric, a = 3, r = 2

b. 5, 14, 23, 32, 41, 50, …
arithmetic, a = 5, d = 9

c. 6, 2, 2/3, 2/9, 2/27, …
geometric, a = 6, r = 1/3

d. 4, -4, 4, -4, 4, …
geometric, a = 4, r = -1

e. 1, -3, 5, -7, 9, …
neither, $a_n = (-1)^n(1 + 2n)$

f. 0, 1, 4, 9, 16, 25, …
neither, $a_n = n^2$

Note: The book states that if you only know a few of the starting terms, there is an infinite number of sequences that start with those terms and the best you can do is make an educated conjecture. However, you can generally assume that if someone writes something like { 0, 2, 4, 6, 8, …. } that the obvious pattern will hold. Otherwise, they would write a more precise sequence.

Summations

- A useful operation on a sequence is to find the sum of some of its terms. The notation is as follows:

$$\sum_{j=m}^{k} a_j = a_m + a_{m+1} + \ldots + a_{k-1} + a_k \qquad k \geq m$$

This denotes the sum of terms *m* through *k*. To find the sum of the first *n* terms of a sequence, use *m* = 0 and *k* = $\prod$*n*-1 (for sequences with first term a_0) or *m* = 1 and *k* = *n* (for sequences with first term a_1). The variable *j* is arbitrary in the summation. Any letter could have been used.

- Example: Compute the following:

$$\sum_{j=3}^{6} j^2 = 3^2 + 4^2 + 5^2 + 6^2 = 86$$

```
1   int sum = 0;
2
3   for( int j=3; j≤6; j++ ) {
4       sum = sum + j*j;
5   }
```

There is also an operator used to multiply sequence terms:

$$\prod_{j=m}^{n} a_j = a_m \text{ x } a_{m+1} \text{ x } \ldots \text{ x } a_{k-1} \text{ x } a_k$$

Summations

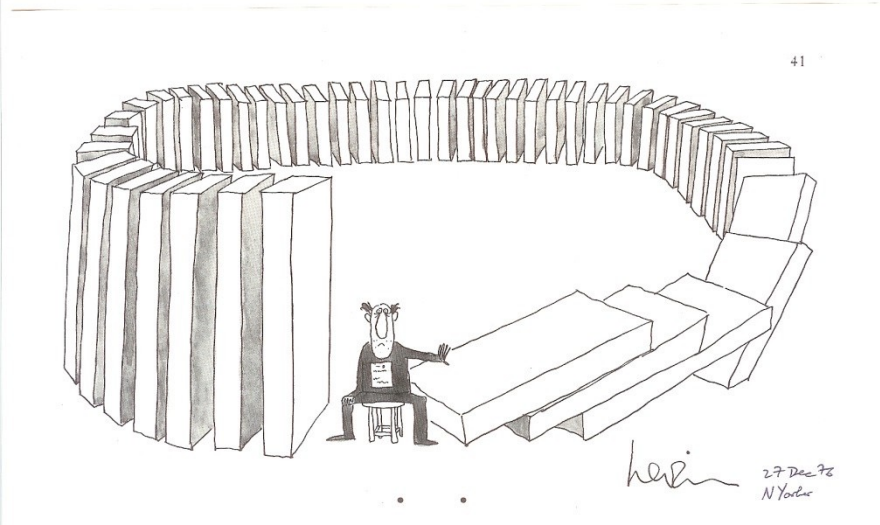
- How do you use these formulas for different indices?
- Example:

$$\sum_{k=5}^{n} k = \sum_{k=1}^{n} k - \sum_{k=1}^{4} k = \frac{n(n+1)}{2} - \frac{4(5)}{2} = \frac{n(n+1)}{2} - 10 = \frac{1}{2}n^2 + \frac{1}{2}n - 10$$

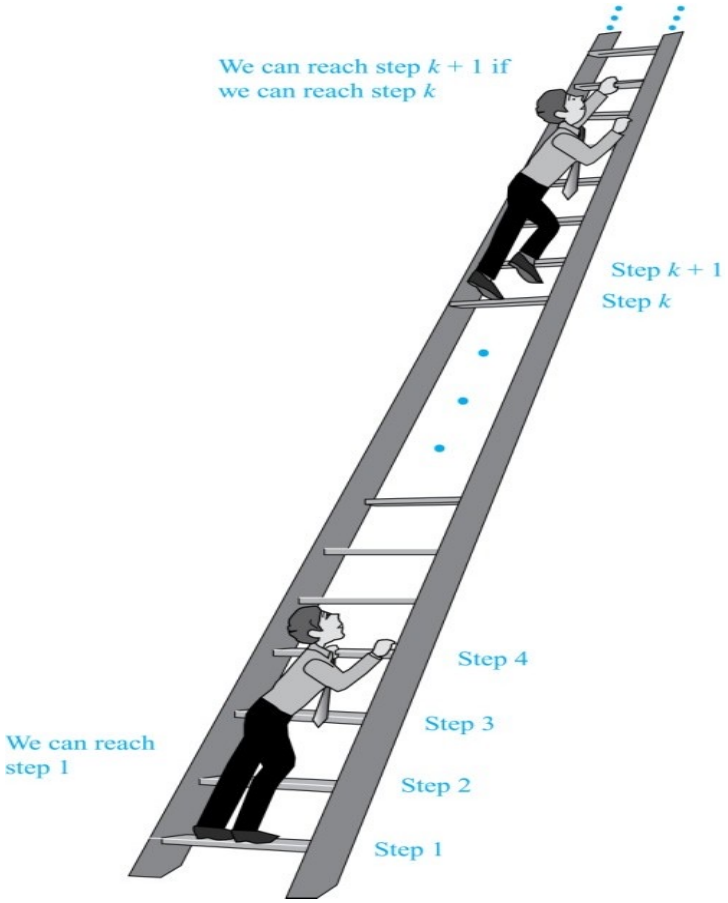
- Another solution for this example is to introduce a variable *j* such that formula will work. Use $j = k - 4$. Substitute $k = j + 4$ inside the summation. Then reduce the lower and upper bounds by 4. This means $j = 1$ when $k = 5$:

$$\sum_{j=1}^{n-4} (j+4) = ?$$

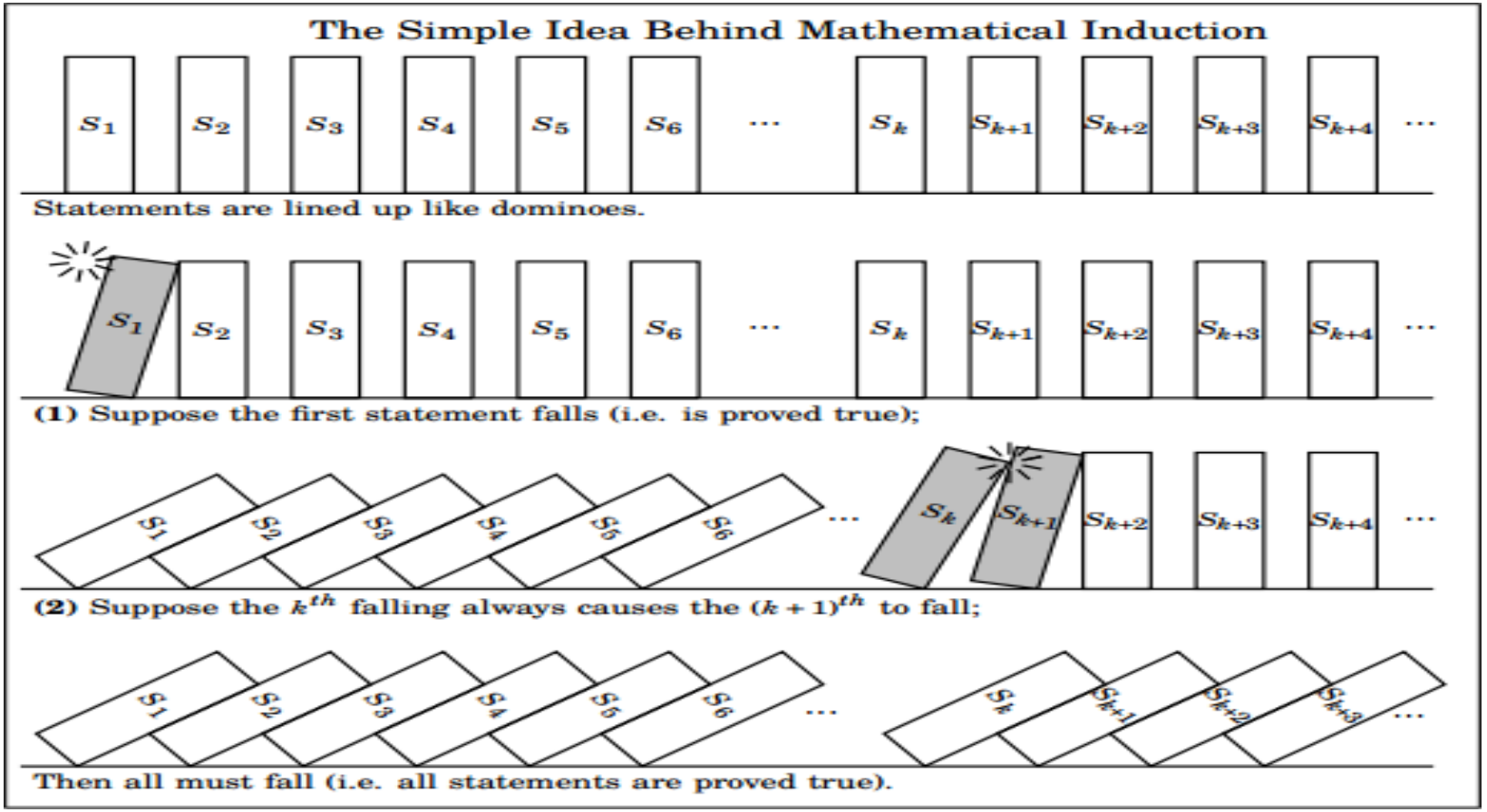
Idea of Induction



What is the idea behind induction?



Idea of Induction



Mathematical Induction

- Mathematical induction: a proof technique to prove statements of the sort: "P(n) is true for all positive integers n". It consists of two steps:
  - ✓ Basis step: Show that P(1) is true.
  - ✓ Inductive step: Assume P(k) holds for an arbitrary positive integer k, show that P(k + 1) must be true. (i.e. Prove that P(k) → P(k + 1) is true for every positive integer k).

  The term P(k) is called the *inductive hypothesis*
- Example: Show that we can reach every rung on the infinite ladder.
  - ✓ Basis step: We can reach step 1.
  - ✓ Inductive step: Assume that we can reach rung k, then we can reach rung k + 1.

  So by mathematical induction, we know that we can reach every rung on the infinite ladder.

Examples

Example: Prove using mathematical induction that $\sum_{j=1}^{n} j = \frac{n(n+1)}{2}$, n is a positive integer.

Solution: Let P(n) be the proposition that $\sum_{j=1}^{n} j = \frac{n(n+1)}{2}$. We want to show that P(n) is true for every n in the domain n ≥ 1.

Basis step: P(1) is true because $\sum_{j=1}^{1} j = 1$, and $\frac{1(1+1)}{2} = 1$.

Inductive step: Assume that P(k) is true for an arbitrary positive integer k. That is, $\sum_{j=1}^{k} j = \frac{k(k+1)}{2}$.

It follows that $\sum_{j=1}^{k+1} j$ = 1 + 2 + 3 + .. + k + (k + 1)

$$= \sum_{j=1}^{k} j + (k+1)$$

$$= \frac{k(k+1)}{2} + \frac{2(k+1)}{2}$$

$$= \frac{(k+1)(k+2)}{2}$$

$$= \frac{(k+1)((k+1)+1)}{2}$$, i.e. P(k + 1) is true

So by mathematical induction, we know that $\sum_{j=1}^{n} j = \frac{n(n+1)}{2}$, for all positive integers n.

Exercises

Exercise: Prove that $5^n - 1$ is evenly divisible by 4 for all $n \geq 1$.

Solution: Let P(n) be the proposition that $5^n - 1$ is evenly divisible by 4. We want to show that P(n) is true for all $n \geq 1$.

Basis step: P(1) is true because $5^1 - 1 = 4$ is evenly divisible by 4.

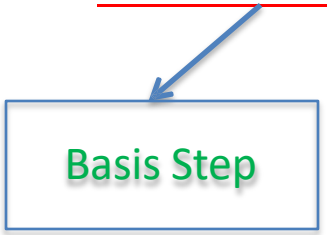
Inductive step: Assume that P(k) is true for an arbitrary integer k. That is to say, $5^k - 1$ is evenly divisible by 4.

It follows that $5^{k+1} - 1 = 5 \times 5^k - 1 = (5^k - 1) + 4 \times 5^k$. Since 4×5^k is evenly divisible by 4 and $5^k - 1$ is assumed to be divisible by 4, the sum $(5^k - 1) + 4 \times 5^k$ is divisible by 4. i.e. P(k + 1) is true.

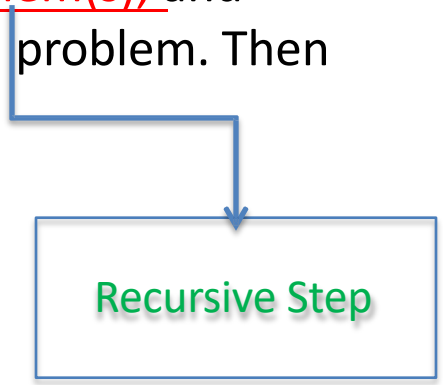
So by mathematical induction, we know that $5^n - 1$ is evenly divisible by 4 for all $n \geq 1$.

Recursion

- Are we done yet?
  - ✓ If so, return the results. Without such a termination condition, a recursion would go on forever.



  - ✓ If not, simplify the problem, solve the simpler problem(s), and assemble the results into a solution for the original problem. Then return that solution.



Recursively Defined Functions

- Recursive function: A function with the set of integers greater than or equal to some integer k (typically k = 0) as its domain defined using these two steps:
  - ✓ basis step: Specify the value of the function at k.
  - ✓ recursive step: Given a rule for finding its value at an integer from its values at smaller integers.

  The term *recurrence relation* is also used to describe recursive functions. Sometimes the basis cases are called *initial conditions*.
- Example: Suppose the recursive function f is defined by

basis step ← $f(0) = 3$

recursive step ← $f(n) = 2f(n - 1) + 3$ $\quad n > 0$

Find f(1), f(2), f(3).

Solution:

$f(1) = 2f(0) + 3 = 2 \times 3 + 3 = 9$

$f(2) = 2f(1) + 3 = 2 \times 9 + 3 = 21$

$f(3) = 2f(2) + 3 = 2 \times 21 + 3 = 45$

Recursively Defined Functions

Example: The Fibonacci numbers are defined as follows:

$f_0 = 0, \quad f_1 = 1$ → basis step

$f_n = f_{n-1} + f_{n-2}$ for $n > 1$ → recursive step

find f_2 , f_3 , f_4 .

Solution:

$f_2 = f_1 + f_0 = 1$

$f_3 = f_2 + f_1 = 2$

$f_4 = f_3 + f_2 = 3$

……

This gives the sequence: 0, 1, 1, 2, 3, 5, 8, 13, 21, …

Why do we need two base values for this sequence?

- Since the recursive step depends on two previous values.

Recursively Defined Functions

Exercise: Suppose H is defined by

$H(1) = 1$

$H(n) = H(n-1) + 6n - 6$ if $n > 1$

Find H(2), H(3), H(4).

Solution:

$H(2) = H(1) + 6 \times 2 - 6 = 7$

$H(3) = H(2) + 6 \times 3 - 6 = 19$

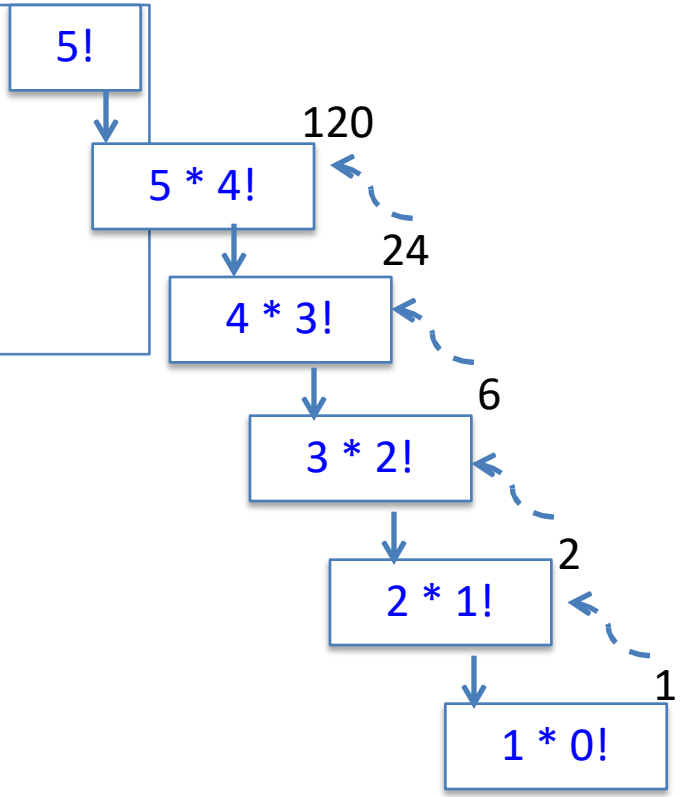
$H(4) = H(3) + 6 \times 4 - 6 = 37$

Recursive Algorithms - Factorial

- recursive algorithm: An algorithm that solves a problem by reducing it to an instance of the same problem with smaller input.

Example: How to recursively compute n!? (n is a nonnegative integer)

```
int factorial(int number)
{
    if (number == 0) return 1;
    else
        return number * factorial(number - 1);
}
```



Recursive Algorithms - Exponentiation

Example: How to recursively compute a^n?

```
int power(int base, int exp)
{
    if (exp == 0)
        return 1;
    else
        return base * power(base, exp - 1);
}
```

Can you rewrite this function to compute a^n without using recursion?

A recursively algorithm must satisfy these three criteria:

- ✓ Stopping case: One or more cases of the problem must have a straightforward, non-recursive solution.
- ✓ Smaller recursive cases: The other cases of the problem can be reduced (using recursion) to one or more problems that are closer to stopping cases.
- ✓ Will hit stopping case: Eventually the problem will be reduced to only stopping cases.

Recursive Algorithms – Linear Search

Example: Construct a recursive linear search algorithm to search for the first occurrence of a target value x in the sequence $a_0, a_1, \ldots, a_n$.

```
// input: (0, n, x)
// output: the location of x in a0, a1, …, an if it appears; otherwise it is -1.
// i, j are integers, 0 ≤ i ≤ j ≤ n
procedure search(i, j, x)
if ai = x then
   return i
else if i = j then
   return -1
else
   return search(i + 1, j, x)
```

The number in a_i matches the target value x → Found it!

Not found!

Search the rest of sequence

Recursive Algorithms – Binary Search

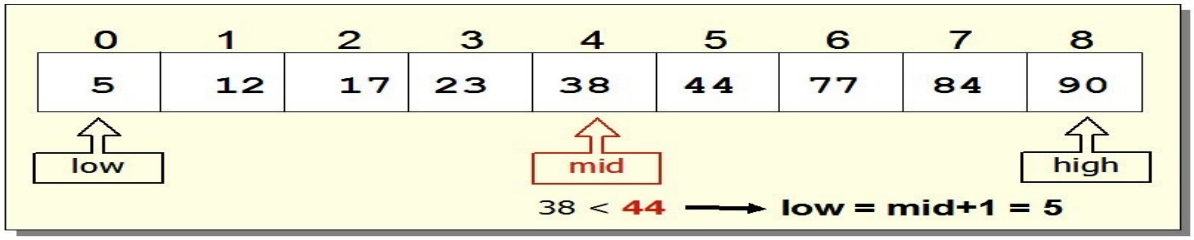
Example: Construct a recursive binary search algorithm to search for the first occurrence of a target value x in the sequence $a_0, a_1, \ldots, a_n$ of integers. Suppose that the sequence is in increasing order.

```
// input: (0, n, x)
// output: the location of x in a0, a1, …, an if it appears; otherwise it is -1.
// low, high are integers, 0 ≤ low ≤ high ≤ n
procedure binarySearch(low, high, x)
mid := ⌊(low + high) / 2⌋
if x = a_mid then
      return mid                    } Found it!
else if (x < a_mid and low < mid) then
      return binarySearch(low, mid – 1, x)    } Search the lower half of sequence
else if (x > a_mid and high > mid) then
      return binarySearch(mid + 1, high, x)   } Search the upper half of sequence
else return -1                      → Not found!
```

	low	high	mid
#1	0	8	4

search(44)

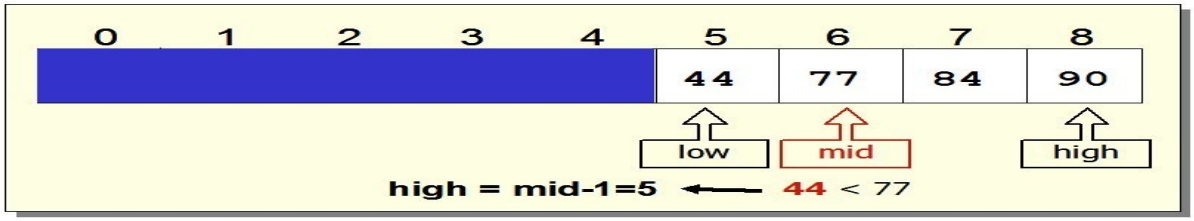
$$mid = \left\lfloor \frac{low + high}{2} \right\rfloor$$



	low	high	mid
#1	0	8	4
#2	5	8	6

search(44)

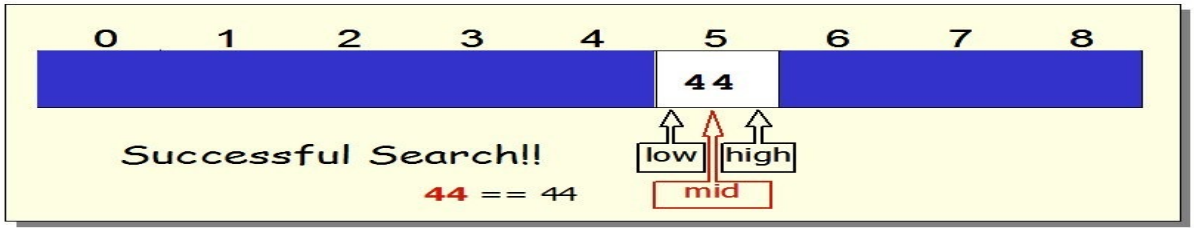
$$mid = \left\lfloor \frac{low + high}{2} \right\rfloor$$



	low	high	mid
#1	0	8	4
#2	5	8	6
#3	5	5	5

search(44)

$$mid = \left\lfloor \frac{low + high}{2} \right\rfloor$$



Modeling with Recursion

- In computer science, we use recurrence relations in the area of algorithm and counting (later). But recurrence relations can actually be used to model a wide variety of problems.

Example: The world population is currently 6.5 billion. If it increases at a rate of 1.3% annually, what will be the world population in n years?

Solution:

$a_0 = 6.5$ billion

$a_n = a_{n-1} + 0.013a_{n-1} = 1.013a_{n-1}$ $\quad n > 0$