CS 381 – Fall 2021

Week 1, Lecture 1

Part 2

Warm Up: Sorting

Input: n numbers $a_1, \ldots, a_n$ stored in an array A

Output: sorted sequence of size n (increasing; aka non-decreasing)

<u>Algorithms you should know</u>

- Bubble Sort, Insertion Sort, Selection Sort
- Merge Sort, Quicksort, Heapsort
- Radix Sort, Bucket Sort, Counting Sort

Why are the algorithms shown in three groups?

Characteristics

- asymptotic performance, stable, in place

Insertion Sort (Section 2.1, 2.2)

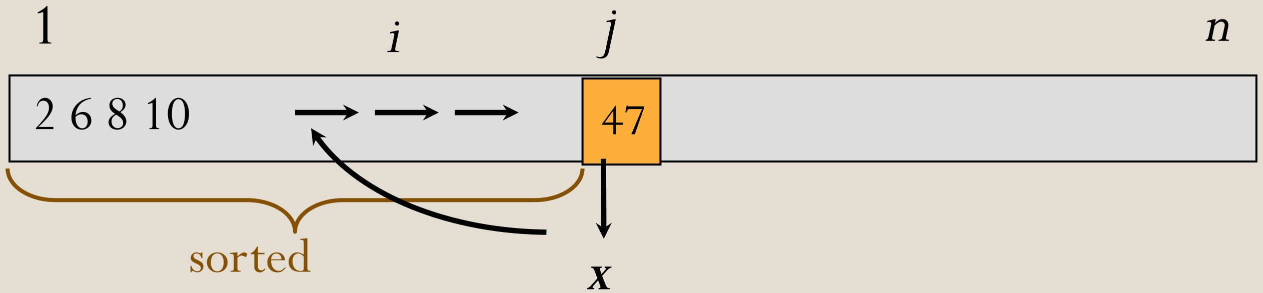
- Create the sorted sequence in an incremental way
- Start with a sorted sequence of length 1 and insert one more element in each iteration
- How does insertion sort work?

Insertion Sort (Section 2.1, 2.2)

- Create the sorted sequence in an incremental way
- Start with a sorted sequence of length 1 and insert one more element in each iteration

```
INSERTION-SORT ($A$, $n$)
for $j \leftarrow 2$ to $n$ do
    $x \leftarrow A[j]$
    $i \leftarrow j - 1$
    while $i > 0$ and $A[i] > x$ do
        $A[i+1] \leftarrow A[i]$
        $i \leftarrow i - 1$
    $A[i+1] \leftarrow x$
```

Note: algorithm descriptions in 381 generally don't need to be low-level.



What is the running time of Insertion Sort?

Number of times the statements in the while-loop are executed depends on the input

- increasingly sorted input is fast; decreasing is slow.

- Best case time: O(n)
- Worst case time: $\sum_{j=2}^{n} j = \frac{n(n+1)}{2} - 1 < n^2$

What all do we count/have to count when analyzing time?

- In (internal) sorting algorithm we generally count the number of comparisons

Asymptotic Performance

Pseudo code has two nested loops

- while loop moves left from position j to 1
- total time won't be more than quadratic.

Note: A doubly nested loop does not necessarily result in quadratic time

Worst case: $T(n) = O(n^2)$

- Work is bounded by summing the first n-1 integers which is equal to $n(n-1)/2$
- Time is proportional to n^2

Insertion Sort: Correctness

```
INSERTION-SORT $(A, n)$
for $j \leftarrow 2$ to $n$ do
    Pre-Condition: $A[1] \leq A[2] \ldots \leq A[j-1]$
    $x \leftarrow A[j]$
    $i \leftarrow j-1$
    while $i > 0$ and $A[i] > x$ do
        $A[i+1] \leftarrow A[i]$
        $i \leftarrow i-1$
    $A[i+1] \leftarrow x$
    Post-Condition: $A[1] \leq A[2] \ldots \leq A[j]$
```

Prove that invariant conditions are preserved by inner loop

When j=n, the array is sorted

Proofs

T(n) is a claim (n is a positive integer).

Prove T(n) correct.

To show that T(n) is false, give a counterexample.

Common proof methods

- Direct proof
- Indirect proof
  - By contraposition, by contradiction
- Mathematical induction
  - weak and strong induction

What do we need to prove in 381?

- Establish the claim of a run time

  A recurrence of a recursive algorithm, sum for an iterative solution, an amortized analysis, etc.

- Correctness of your algorithm

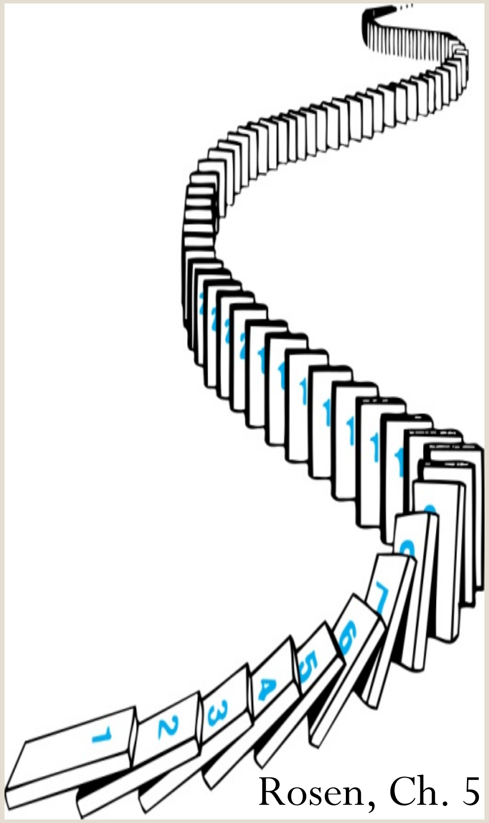
  An inductive argument (even for iterative solutions), proof by contradiction, etc.

- Showing NP-completeness of a problem
- Lower bound of a problem

Remember How Mathematical Induction Works

Consider an infinite sequence of dominoes, 1,2,3, …, where each domino is standing.

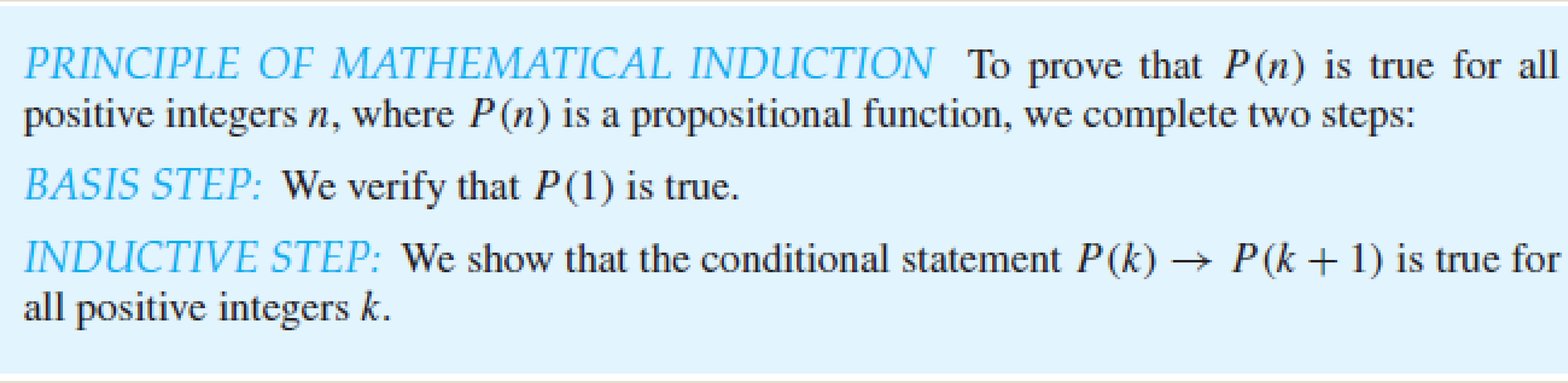
Let $P(n)$ be the proposition that the nth domino is knocked over.



We know that the first domino is knocked down, i.e., $P(1)$ is true.

When the k-th domino is knocked over, it knocks over the $(k + 1)$st domino; i.e, $P(k) \rightarrow P(k + 1)$ is true for all positive integers k.

Hence, all dominos are knocked over.
$P(n)$ is true for all positive integers n.



PRINCIPLE OF MATHEMATICAL INDUCTION To prove that $P(n)$ is true for all positive integers n, where $P(n)$ is a propositional function, we complete two steps:

BASIS STEP: We verify that $P(1)$ is true.

INDUCTIVE STEP: We show that the conditional statement $P(k) \rightarrow P(k+1)$ is true for all positive integers k.

Rosen, Chapter 5

Weak Induction

- Basis Step: T(1) holds (basis is often 1; can be a larger value)
- Inductive Steps: assume that for all k>1, ***T(k) holds***.
- Using the induction hypothesis, show that T(k+1) holds.
- It follows that T(n) holds for all n and the claim is proven.

Weak Induction

- Basis Step: T(1) holds (basis is often 1; can be a larger value)
- Inductive Steps: assume that for all k>1, ***T(k) holds***.
- Using the induction hypothesis, show that T(k+1) holds.
- It follows that T(n) holds for all n and the claim is proven.

Strong Induction

- Basis Step: T(1) holds
- Inductive Step: assume that for all k>1, ***T(k) holds for 1, 2, … k***
- Using the induction hypothesis, show that T(k+1) holds.
- It follows that T(n) holds for all n and the claim is proven.

Claim: $n! > 2^n$ for $n>3$ (Rosen, Ch. 5)

Task: Prove inequality by induction. How would this work?

Claim: $n! > 2^n$ for $n>3$ (Rosen, Ch. 5)

Basis Step for n=4:

$$4! = 4 \times 3 \times 2 \times 1 = 24 > 2^4 = 16$$

Claim: $n! > 2^n$ for $n>3$ (Rosen, Ch. 5)

Basis Step for n=4:

$4! = 4 \times 3 \times 2 \times 1 = 24 > 2^4 = 16$

Inductive Hypothesis:

Inequality holds for all n=k > 3: $k! > 2^k$

Claim: $n! > 2^n$ for n>3 (Rosen, Ch. 5)

Basis Step for n=4:

$4! = 4 \times 3 \times 2 \times 1 = 24 > 2^4 = 16$

Inductive Hypothesis:

Inequality holds for all n=k > 3: $k! > 2^k$

Inductive Step:

Assuming inductive hypothesis, show the claim for k+1:

$$\begin{aligned}(k+1)! = (k+1)\cdot k! &> (k+1)\cdot 2^k && \text{(by IH)}\\ &> 2\cdot 2^k && \text{(since } k > 3\text{)}\\ &\geq 2^{k+1}\end{aligned}$$

It follows that the claim holds for all n.

Other Relevant/Common Sums

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

$$\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$$

$$\sum_{i=0}^{n} 2^i = 2^{n+1} - 1$$

$$\sum_{i=0}^{n} c^i = \frac{c^{n+1}-1}{c-1}$$

…

Exact bounds and asymptotic bounds

CLRS, 182 Text, TCS Cheat Sheet (on Piazza), etc.

Claim: $\sum_{i=1}^{n} i \cdot 2^i = (n-1) \cdot 2^{n+1} + 2$

Proof?

Claim: $\sum_{i=1}^{n} i \cdot 2^i = (n-1) \cdot 2^{n+1} + 2$

Basis Step: holds for $n=1$ as $2=0+2$ true

Claim: $\sum_{i=1}^{n} i \cdot 2^i = (n-1) \cdot 2^{n+1} + 2$

Basis Step: holds for n=1 as 2=0+2 true

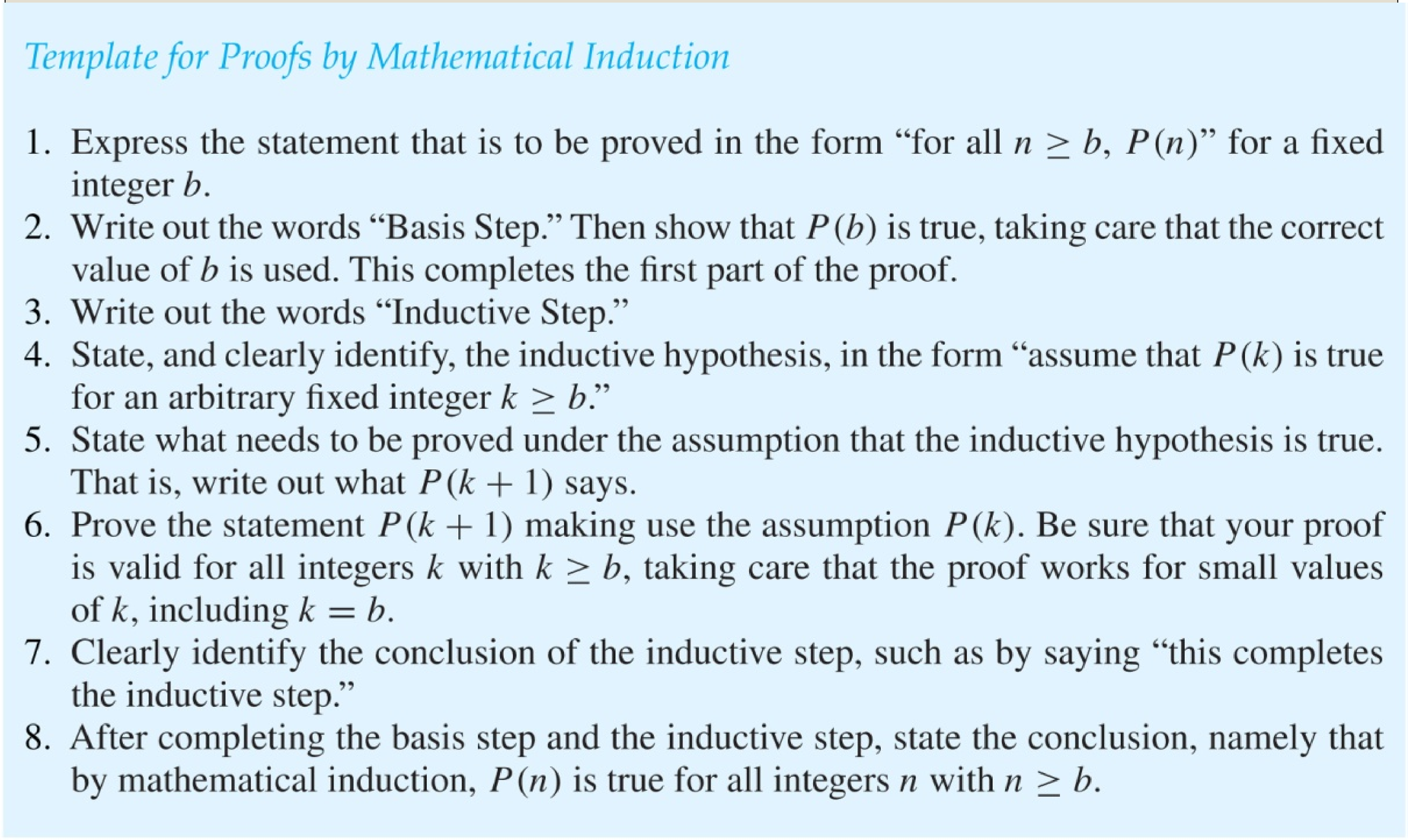
Inductive Hypothesis:

Assume the claim holds for k-1: $\sum_{i=1}^{k-1} i2^i = (k-2)2^k + 2$

Inductive Step. Prove the claim for k:

$$\begin{aligned}\sum_{i=1}^{k} \mathbf{i2^i} &= \sum_{\mathbf{i=1}}^{k-1} i\, 2^i + k2^k \quad \text{[use induction hypothesis]}\\ &= (k-2)2^k + 2 + k2^k\\ &= (k-2+k)2^k + 2\\ &= (2k-2)2^k + 2\\ &= \mathbf{(k-1)2^{k+1} + 2}\end{aligned}$$

It follows that the claim holds for all n.



Template for Proofs by Mathematical Induction

1. Express the statement that is to be proved in the form "for all $n \geq b$, $P(n)$" for a fixed integer b.
2. Write out the words "Basis Step." Then show that $P(b)$ is true, taking care that the correct value of b is used. This completes the first part of the proof.
3. Write out the words "Inductive Step."
4. State, and clearly identify, the inductive hypothesis, in the form "assume that $P(k)$ is true for an arbitrary fixed integer $k \geq b$."
5. State what needs to be proved under the assumption that the inductive hypothesis is true. That is, write out what $P(k+1)$ says.
6. Prove the statement $P(k+1)$ making use the assumption $P(k)$. Be sure that your proof is valid for all integers k with $k \geq b$, taking care that the proof works for small values of k, including $k = b$.
7. Clearly identify the conclusion of the inductive step, such as by saying "this completes the inductive step."
8. After completing the basis step and the inductive step, state the conclusion, namely that by mathematical induction, $P(n)$ is true for all integers n with $n \geq b$.

[Rosen, chapter 5]

What is next?

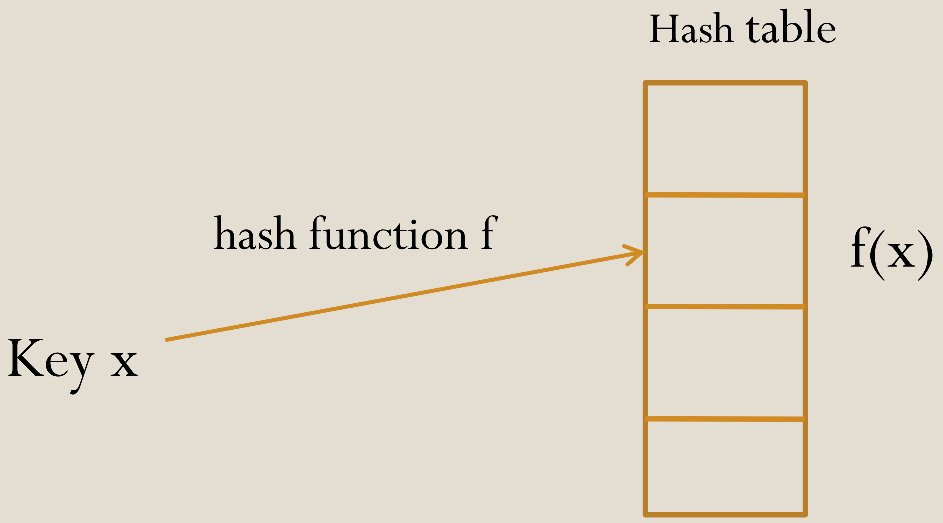
Basics of algorithm analysis

- O, Θ, Ω notation
- Analyzing asymptotic performance of code segments
- Ranking and comparing running times
- Common complexity classes

Can we use hashing?

Hashing (insert, delete, search) is important and effective.

Note algorithms in CS381 cannot use hashing unless explicitly stated.



Collisions can lead to poor worst-case performance.

- To get O(1) expected performance/operation, one needs the right table size, a good hash function and good collision resolution.

Example: 2-sum problem

Given: unsorted array A of n integers and an integer T.

Determine whether there exist x and y in A such that x+y = T

8	5	-3	6	10	1

T = 14

Algorithm to solve this problem?

Example: 2-sum problem

Given: unsorted array A of n integers and an integer T.

Determine whether there exist x and y in A such that x+y = T

8	5	-3	6	10	1

T = 14

Algorithmic solution

- Sort entries in A in O(n log n) time
- Then, for each entry x, binary search for T-x. Or scan from both ends. Runtime?

Example: 2-sum problem

Given: unsorted array A of n integers and an integer T.

Determine whether there exist x and y in A such that x+y = T

8	**5**	**-3**	**6**	**10**	**1**

T = 14

Algorithmic solution

- Sort entries in A in O(n log n) time
- Then, for each entry x, binary search for T-x. Or scan from both ends. Runtime: O(n log n) worst case

Example: 2-sum problem

Given: unsorted array A of n integers and an integer T.

Determine whether there exist x and y in A such that x+y = T

8	**5**	**-3**	**6**	**10**	**1**

T = 14

Algorithmic solution

- Sort entries in A in O(n log n) time
- Then, for each entry x, binary search for T-x. Or scan from both ends. *Runtime:* O(n log n) worst case

Hashing solution

- Insert the entries in A into a hash table H
- For each entry x in A, search for T-x in H.
- *Runtime:* O(n) expected time if hashing done right. Worst case O(n^2)