CS3230 Semester 1 2024/2025
Design and Analysis of Algorithms

Tutorial 03
Proof of Correctness and D&C (1)
For Week 04

Document is last modified on: August 17, 2024

1 Lecture Review: Proof of Correctness

We prove the correctness of an algorithm depending on its type:

- For iterative algorithm, we usually use loop invariant.
  Invariant is a condition which is TRUE at the start of EVERY iteration
  We can then use invariant to show the correctness:
  1. Initialization: It is true before iteration 1
  2. Maintenance: If it is true for iteration x, it remains true for iteration x+1
  3. Termination: When the algorithm ends, it helps the proof of correctness
- For recursive algorithm, we usually use proof by induction.
  1. Show the recursive algorithm is (trivially) correct on its base case(s).
  2. Inductive step: show that the recursive algorithm is correct, assuming that the smaller cases are all correct.

2 Lecture Review: D&C

Here are the usual steps for using Divide and Conquer (D&C) problem solving paradigm for problems that are amenable to it:

1. **Divide**: Divide/break the original problem into $\underline{\geq 1 \text{ smaller}}$ sub-problems.

2. **Conquer**: Conquer/solve the sub-problems recursively.

3. **Combine** (optional): Optionally, combine the sub-problem solutions to get the solution of the original problem.

The most classic D&C example is **Merge Sort**.

1. **Divide**: Divide/break the original problem of sorting n elements into 2 smaller sub-problems of sorting $\frac{n}{2}$ elements.

2. **Conquer**: Conquer/solve the sorting of $\frac{n}{2}$ elements recursively.

3. **Combine** ~~(optional)~~: Merge 2 already sorted $\frac{n}{2}$ elements.

3 Tutorial 03 Questions

Q1). Consider the following iterative sorting algorithm `InsertionSort`(A).

for $i \in [1..N-1]$ // outer for loop i

1. let X be $A[i]$ // X is the next item to be inserted into $A[0..i-1]$

2. for $j \in [i-1..0]$ (down) // inner for loop j

   (a) if $A[j] > X$, set $A[j+1] = A[j]$ // make a place for X

   (b) else, break

3. $A[j+1] = X$ // insert X at index $j+1$

Assuming the inner for loop j is correct, answer the following two questions:

1. What is the suitable loop invariant for the outer for loop i?

2. Show the invariant after initialization, maintenance, and termination.

Q2). Consider the following recursive sorting algorithm `StoogeSort`(A).

1. Let n be the length of array A.

2. If $n = 2$ and the first number is larger than the second number, swap the two numbers.

3. If $n > 2$, do the following three steps sequentially.

   (a) Apply `StoogeSort` to sort the initial $\lceil 2n/3 \rceil$ numbers recursively.

   (b) Apply `StoogeSort` to sort the final $\lceil 2n/3 \rceil$ numbers recursively.

   (c) Apply `StoogeSort` to sort the initial $\lceil 2n/3 \rceil$ numbers recursively.

Answer the following three questions:

1. Prove that `StoogeSort`(A) correctly sorts the input array A.
   For the sake of simplicity, you may assume that all numbers in A are distinct.

2. Analyze the time complexity of `StoogeSort`.

Q3, Q4, Q5. involves Finding a Peak Problem
You are given a 2D array of m rows and n columns.
Each cell has a number
You want to find any single **peak**: A cell where the number is $\geq$ than all of its (up to) four (North/East/South/West) neighbors.

For example, given $m \times n = 3 \times 5$ grid below, there are 5 peaks (denoted with a '*'):

```
6  8* 7  7* 1
9* 3  1  7* 3
8  4  5* 3  2
```

Q3). Show that there is a peak in every 2D array!

We want to come up with a recursive algorithm to find any peak: `FindPeak(A)`:

1. If A has only $n = 1$ column, then Return the maximum element in the column.
2. Otherwise (if A has $n >= 2$ columns),
   (a) Consider the middle column of A,
   (b) Find a maximum element on that middle column
   (c) Check if that element is a peak
   (d) If yes, then Return that element
   (e) Otherwise,
      i. X = FindPeak(Left_Half_of_A_without_the_middle_column)
      ii. Y = FindPeak(Right_Half_of_A_without_the_middle_column)
      iii. If either X or Y is a peak, Return it, else Return None // see Question Q3.

Q4). What in the runtime complexity of `FindPeak(A)` algorithm?

Q5). Should `FindPeak(A)` algorithm do both step 2.(e).i and step 2.(e).ii.?