Algorithms: Design and Correctness

Prepared by Hyrum D. Carroll

(based on slides from Suk Jin Lee)

Algorithms Representation: Pseudocode

$\leftarrow$	Assignment	$i \leftarrow j+1$ $\quad i \leftarrow i-1$
for…do	**for** loop	**for** $i \leftarrow 1$ to n **do** **loop body** i is a loop counter (loop variable)
while…do	**while** loop	**while** (logical condition) **do** **loop body**
[1..n]	a range within an array	a[1..n]
a[i]	the i^{th} element of the array	
length[a]	length of array a	
//	the reminder of the line is a comment	
if then else	conditional branching	if <logical condition> then <statement(s)> else <statement(s)>

Sorting Problem

- Input
  - A sequence of n numbers $(a_1, a_2, \ldots, a_n)$
- Output
  - A permutation (reordering) $(a_1', a_2', \ldots, a_n')$ such that $a_1' \leq a_2' \leq \ldots \leq a_n'$

Insertion Sort – 1st algorithm

- INSERTION SORT[1] (A)

```
for j ← 2 to length[A]
   do { key ← A[j]
        // Insert A[j] into the sorted sequence A[1…j–1]
        i ← j – 1
        while (i > 0) and (A[i] > key)
              do { A[i+1] ← A[i]
                   i ← i – 1
                   A[i + 1]  ← key}
        }
```

Insertion Sort: Example

Array 5 2 4 6 1 3

j = 2: <u>5 **2**</u> 4 6 1 3 → **2** 5 4 6 1 3

j = 3: 2 <u>5 **4**</u> 6 1 3 → 2 **4** 5 6 1 3

j = 4: 2 4 5 **6** 1 3 →

j = 5: 2 4 5 <u>6 **1**</u> 3 → 2 4 <u>5 **1**</u> 6 3 → 2 <u>4 **1**</u> 5 6 3 → <u>2 **1**</u> 4 5 6 3 →**1** 2 4 5 6 3

j = 6: 1 2 4 5 <u>6 **3**</u> → 1 2 4 <u>5 **3**</u> 6 → 1 2 <u>4 **3**</u> 5 6 → 1 2 **3** 4 5 6

9 swapping operations, 30 logical conditions (comparisons),

`for` loop with 5 iterations

Insertion Sort – $1^{st\text{-}a}$ algorithm

- INSERTION SORT$^{1\text{-}a}$(A)

for $j \leftarrow 2$ to **length**[A]

  do { **key** $\leftarrow A[j]$

    // Insert $A[j]$ into the sorted sequence $A[1\ldots j-1]$

    $i \leftarrow j - 1$

    while ($i > 0$) and ($A[i] >$ key)

      do { A[i+1] $\leftarrow$ A[i]

        $i \leftarrow i - 1$}

    $A[i+1] \leftarrow$ key

    }

Insertion Sort: Example

Array 5 2 4 6 1 3

j = 2: 5 **2** 4 6 1 3 → □ **5** 4 6 1 3 → **2** 5 4 6 1 3

j = 3: 2 5 **4** 6 1 3 → 2 □ **5** 6 1 3 → 2 **4** 5 6 1 3

j = 4: 2 4 5 **6** 1 3 →

j = 5: 2 4 5 6 **1** 3 → 2 4 5 □ **6** 3 → 2 4 □ **5** 6 3 → 2 □ **4** 5 6 3 → □ **2** 4 5 6 → **1** 2 4 5 6 3

j = 6: 1 2 4 5 6 **3** → 1 2 4 5 □ **6** → 1 2 4 □ **5** 6 → 1 2 □ **4** 5 6 → 1 2 **3** 4 5 6

4 swapping operations, 9 movements of the array elements, 30 logical conditions (comparisons), `for` loop with 5 iterations

Correctness of an algorithm

- Any algorithm must be **correct**, which means that it has to produce the desired result (some value or set of values) from the relevant input
- A proof of the correctness is a very important task!

Method to prove the Correctness of an algorithm

- Loop Invariant
- Induction
- Many other methods that are out of scope of this course

Loop Invariant

- **Loop Invariant** is a logical statement, which can help us understand why an algorithm, which is implemented as a loop (iterative process) gives the correct answer
- Loop invariant can be used to prove the correctness of both a "while" loop and a "for" loop

Loop Invariant Properties

- **Initialization**: it is true prior to the first iteration of the loop
- **Maintenance**: if it is true before an iteration of the loop, it remains true before the next iteration
- **Termination**: when the loop terminates, the invariant – usually along with the reason that the loop terminated – gives us a useful property that helps to show that the algorithm is correct

Loop Invariant – Example. 1st Sorting Algorithm (Insertion-Sort)

- At the start of each iteration of the "outer" (for) loop, which is indexed by j, the subarray consisting of elements $A[1 \mathinner{..} j-1]$ is already sorted

Loop Invariant – Example.
1st Sorting Algorithm (Insertion-Sort)

- **Initialization**
  - The loop invariant holds before the first loop iteration, when $j = 2$.
- **Maintenance**
  - Each iteration maintains the loop invariant until it finds the proper position for $A[j]$.
- **Termination**
  - Each loop iteration increases j by 1.
    $j >$ Length$[A] = n \quad \Rightarrow \quad j = n + 1$
    subarray $A[1 \ldots n]$ consists of the elements originally in $A[1 \ldots n]$, in sorted order.

Induction

- Suppose
  - Basis: $S(j)$ is true for fixed constant k
    - Often $k = 0$ or $k = 1$, but can be any integer
  - Inductive hypothesis: $S(n)$ is true
  - Inductive step: If $S(n)$ is true ➠ $S(n+1)$ is true too
- Then $S(j)$ is true for all $j \geq k$
- This means that if $S(k)$ is true for the fixed constant k and it follows from $S(j)$ is true for $j = n$ that $S(j)$ is true for $j = n + 1$, then $S(j)$ is true for all $j \geq k$

Proof By Induction

- **Claim: S (j) is true for all $j \geq k$**
- **Basis**:
  - Show a statement is true when $j = k$
- **Inductive hypothesis**:
  - Assume the statement is true for an arbitrary $j = n$
- **Step**:
  - Show that implication $S(n) \rightarrow S(n+1)$ is true, thus the statement is then true for any j

Induction Example: Arithmetic Progression

- Arithmetic Progression is a sequence of numbers such that the difference of any two successive members of the sequence is a constant:

$$a_1, a_1 + d, a_1 + 2d, \ldots, a_1 + nd$$

- d is the difference of the progression
- a_n=?

Induction Example 1: Arithmetic Progression

- **Prove** $\boxed{a_n = a_1 + (n-1)d}$
  - Basis:
    - If $\boldsymbol{n = 1}$, then $\boldsymbol{a_1} = a_1 + (1-1)d = a_1 + 0d = \boldsymbol{a_1}$
  - Inductive hypothesis (assume true for n):
    - Assume $a_n = a_1 + (n-1)d$
  - Step (show true for $n + 1$)
    - a_{n+1} $= a_n + d = \underline{a_1 + (n-1)d} + d$
      $= a_1 + nd - d + d = a_1 + ((n+1) - 1)d$

Induction Example 2: Arithmetic Progression

- The sum of the first n members of the arithmetic progression is given by

$$S = \frac{a_1 + a_n}{2} n$$

- Particularly, for the progression with the first member 1 and the difference 1

$$S = \frac{1+n}{2} n = \frac{n(1+n)}{2}$$

Induction Example 2: Arithmetic Progression

- Prove $1 + 2 + 3 + \ldots + n = n(n + 1) / 2$
  - Basis:
    - If $n = 0$, then $0 = 0(0+1) / 2$
  - Inductive hypothesis (assume true for n):
    - Assume $1 + 2 + 3 + \ldots + n = n(n + 1) / 2$
  - Step (show true for $n + 1$):
    - $$\begin{aligned} 1 + 2 + \ldots + n + (n + 1) &= (1 + 2 + \ldots + n) + (n + 1) \\ &= n(n + 1) / 2 + 2(n + 1) / 2 \\ &= [n(n + 1) + 2(n + 1)] / 2 \\ &= (n + 1)(n + 2) / 2 \\ &= (n + 1)((n + 1) + 1) / 2 \end{aligned}$$

How to prove correctness of an algorithm using induction?

- ***Induction*** can be used to prove the correctness of any "for" loop and any algorithm whose main part is a "for" loop
- How it works?

How to prove correctness of an algorithm using induction?

- We have to check whether a loop whose correctness we need to prove produces a correct output from the smallest reasonable input (for the lowest reasonable end value of the loop variable j) (basis)
- Then, we assume that the loop works correctly for $j = 1 \ldots n$ (inductive hypothesis)
- Then we have to show that from the fact that the loop produces a correct result for $j=1\ldots n$, it follows that it produces the correct result for $j = n + 1$ (inductive step)

Strong Induction

- We've been using **weak induction**
- **Strong induction** means
  - Basis: show $S(k)$
  - Hypothesis: assume $S(j)$ holds for arbitrary $j \leq n$
  - Step: Show $S(n + 1)$ follows
- Another variation:
  - Basis: show $S(0)$, $S(1)$
  - Hypothesis: assume $S(n)$ and $S(n+1)$ are true
  - Step: show $(S(n) \cap S(n+1) \rightarrow S(n+2))$