CSC236H
Introduction to the Theory of Computation

Bahar Aameri

Fall 2019 – Week 12

This week's plan:

- Final Exam Information.
- Course Evaluation.
- Review and Problem Solving: Run-time complexity and Program Correctness.
- Review: Regular Languages, Induction, PWO

- A3 Solutions will be posted by Saturday Morning (Dec 07).
- **Assignment 3** TA Office Hours:
  - Tuesday, **Dec 03, 3pm – 5pm** Location: **BA2283**
  - Wednesday, **Dec 04, 3pm – 7pm** Location: **BA4290** (Time or location might change)
- **Final Exam** TA Office Hours:
  - Friday, **Dec 06, 3 – 5pm** Location: **BA3201**
  - Monday, **Dec 09, 5pm – 7pm** Location: **BA3201**
- **Bahar**'s Pre-exam Office Hours:
  - Thursday, **Dec 05, 3pm – 5pm** in **BA2283**
  - Monday, **Dec 09, 1:30am – 3:30pm** in **BA2283**

Final Exam: Info

- You will be allowed to bring one **double-sided, handwritten 8.5x11 aid sheet**.
- *Printed, typed, or photocopied sheets are **NOT** allowed.*
- To **pass** the course you must achieve at least **35% on the final exam**.
- Topics:
  - Proofs by PWO and induction. → 1 question
  - Designing DFA's, NFA's, and regular expressions.
  - Proving a given language is regular/non-regular.

  } 3 questions
  - Program correctness. → 2 questions
  - Converting recurrence to closed-form expressions.
  - Running-time analysis of recursive programs.

  } 1 question

Final Exam: Preparation

(Best to be done in the following order)

1. Review all **lecture slides** and course notes.

2. Review all **tutorials**, **exercises**, and **assignments**.
   Redo the questions that you haven't done correctly, or another person in your group did. Make sure to understand **all details** of the posted solutions.

3. Past Exams (from UofT Exam Repository): **Time yourself**!

4. Additional exercise, if needed (from the Course Notes or the suggested textbook).

Final Exam: Tips for wrinting the exam

- **Time yourself** on each question.

- Do **not** spend too much time on a question.
  Figure out ahead of time how much time to devote to each question (based on how much it's worth), and stick to that estimate as much as possible.

- If you cannot figure out how to solve a question in 10 minutes, **move on** to the next question.

- If you have an **idea** how to solve a question but no time to do it in detail, write down your idea.
  Partial marks will be given for proof structures/ideas

- Do **NOT** feel like you must fill all of the available space. It is quite possible that a correct answer will require only part of the space for some questions.

Course Evaluations

- Available at http://uoft.me/openevals

Review – Run-time Analysis of Recursive Algorithms

1. Define the **input size**.

2. Identify parts of the algorithm which their running time **depends on the input size** (e.g., loops and recursive calls).

3. Give a recursive **definition for** $T(n)$.

4. If possible, use the **Master Theorem** to find an upper-bound for $T(n)$.
   If not, use **repetitive substitution** to find a find an upper-bound for $T(n)$.

5. **Note:** The Master Theorem can **only** be used for identifying upper-bounds. If a question asks for a closed-form expression for a function, repeated substitution must be used.

Review – Closed-form of Recursive Functions

1. **Substitute** the recursive term a few time to find a pattern.
2. **Guess** the recurrence formula after i **substitutions**.
3. **Solve** i for (each) the based case.
4. **Plug** i into the formula from Step 2 to find a potential closed form.
5. **Prove** the correctness of the potential closed-form expression by induction.

Review – Correctness of Recursive Programs

1. Define **input size**.

2. Formulate a predicates which is **defined over the input size** and denotes the assertion that if the pre-conditions hold, then the program terminates and post-conditions are satisfied.

3. Prove the predicate by **induction**.

Review – Correctness of Recursive Programs

1. Formulate a **loop invariant**.
2. Prove the LI using **Induction**:
   a) **(Base Case)**: Assume the precondition holds, prove then the LI holds on entering the
   b) **(Induction Step)**: Assuming $LI(k)$ holds and $k+1$ iterations exists, prove $LI(k+1)$
3. Use the LI to prove **partial correctness**:
   a) Assume the loop terminates after t iterations. Then use **loop exit condition** and $LI(t)$ to prove **post-condition**.
4. Define a **loop measure** m and use the pre-condition and LI to prove that
   a) the value of m is a natural number on entering the loop, and after every iteration.
   b) the value of m decreases with every iteration.
   Note: The loop measure does Not have to be exactly equal to the maximum number of remaining iterations.
   It could be an upper-bound for the maximum number of remaining iterations.

Review – Tips for Identifying Loop Invariants

- Loop invariants provide a formal description of relationships between variables in the loop.
- Loop invariants are incremental: they should be about a **portion** of the values that are being processed.
- A loop invariant should **hold** even when the loop condition is false (after termination).

Trial-and-error process for identifying loop invariants:

1. Understand what the loop does.
   How the loop's function is related to the correctness of the program?
   Tracing the program sometimes help.
2. Formulate a candidate statement as our proposed loop invariant.
3. Check to see if the proposed loop invariant is sufficient to prove LI, termination, and partial correctness.
4. If the answer in Step 3 is negative, then we repeat the above three steps.

Review – Correctness of Recursive Programs Containing Loops

- **Step 1:** Prove the correctness of the loop.
  - Must specify **preconditions** of the loop.
- **Step 2:** Prove the correctness of the program like a regular recursive program, and using the LI proved in Step 1.
  1. Show that the preconditions of the loop hold, hence (by Step 1) it terminates after t iterations and after the termination $LI(t)$ holds.
  2. Use $LI(t)$ and IH to prove the induction step.

Review – Example

$n = Len(A)$

Assume n is a power of 2

```
# A is a list.
    def fun(A):
1.      if len(A) < 2:
2.          return 1
3.      else:
4.          m = len(A)//2        # Integer division
5.          return fun(A[0 : m]) * fun(A[m : len(A)])
```

$n < 2$ → a

$n \geq 2$ → $T(n/2)$ → $T(\frac{n}{2})$

$A[0 : m]$: $\frac{n}{2}$; $A[m : len(A)]$: $\frac{n}{2}$

Give an asymptotic **upper-bound** for the worst-case running time of the algorithm.

$$T(n) = \begin{cases} a & \text{if } n < 2 \\ f(n) \; [b + 2T(\frac{n}{2})] & \text{if } n \geq 2 \end{cases}$$

c d

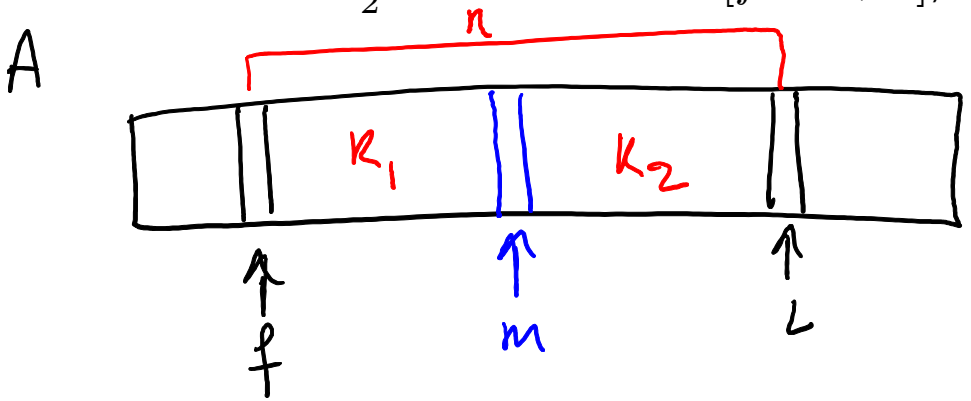
$f(n) \in \Theta(n^0) \Rightarrow k \geq 0$

$\log_d c = \log_2 2 = 1 > k$

$\Rightarrow$ by Master Theorem $T(n) \in \Theta(n)$

Review – Example

Preconditions: A is an array of integers. f and ℓ are integers such that $0 \leq f \leq \ell < len(A)$.
Postconditions: For all integers b, if b occurs more than $\frac{len(A[f:\ell+1])}{2}$ times in $A[f : \ell + 1]$, then $maj(A, f, \ell)$ returns b.



$n = K_1 + K_2$

```
def maj(A, f, ℓ):
1.      if f == ℓ:
2.          return A[f]
3.      m = (f + ℓ)//2        # Integer division
4.      x = maj(A, f, m)
5.      c = 0
6.      i = f
7.      while i ≤ ℓ:
8.          if A[i] == x:
9.              c = c + 1
10.         i = i + 1
11.     if c > (ℓ − f + 1)/2:
12.         return x
13.     return maj(A, m + 1, ℓ)
```

Prove the **correctness** of maj with respect to the given specification.

LI(k): If the loop iterates at least k times

(1) C_k is equal to the number of occurrences of x in $A[f : i_k]$.

(2) $f \leq i_k \leq L+1$

Loop measur: $m_k = L + 1 - i_k$

Review – Example

Preconditions: A is an array of integers. f and ℓ are integers such that $0 \leq f \leq \ell < len(A)$.
Postconditions: For all integers b, if b occurs more than $\frac{len(A[f:\ell+1])}{2}$ times in $A[f : \ell + 1]$, then $maj(A, f, \ell)$ returns b.

```
    def maj(A, f, ℓ):
1.      if f == ℓ:
2.          return A[f]
3.      m = (f + ℓ)//2      # Integer division
4.      x = maj(A, f, m)
5.      c = 0
6.      i = f
7.      while i ≤ ℓ:
8.          if A[i] = x:
9.              c = c + 1
10.         i = i + 1
11.     if c > (ℓ − f + 1)/2:
12.         return x
13.     return maj(A, m + 1, ℓ)
```

n = len(A[f:L+1])

n = L − f + 1

P(n): for all A, and all f, L ∈ ℕ and all b ∈ ℤ, if the Precond holds and n = L − f + 1, then maj(A, f, L) terminates and the Postcond holds.

Fact: If $u \leq v$, $v+1 \leq w$, and b occurs more than $s+s'$ times in $B[u:w+1]$, then either b occurs more than s times in $B[u:v+1]$ or b occurs more than s' times in $B[v+1:w+1]$.

(Recall that $B[i:j+1]$ is the slice of B from i to j)

B.C.: Let $n=1$.

Then $f=2$, and so the if-cond on Line 1 holds. Moreover, $A[f:L+1] = A[f]$ and so $A[f]$ occures more than $\frac{L-f+1}{2} = 1/2$ times in $A[f:L+1]$. Thus $maj(A,f,L)$ terminates on Line 2 and returns the correct value.

IS: Let $n \in \mathbb{N}$, $n \geq 1$. Assume for all $j \in \mathbb{N}$, $1 \leq j < n$, $P(j)$ holds. [IH]

WTP: P(n) holds.

Assume b occures more than $n/2$ times in $A[f:L+1]$ (otherwise P(n) is vacously true). (*)

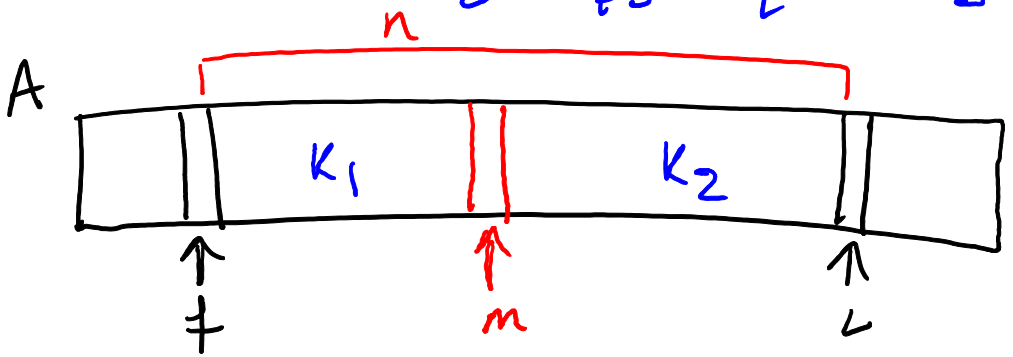
Since $n>1$, we have $f<L$ and so $f+f<f+L<L+L$.

so $f<\frac{f+L}{2}<L$ and $f\leq \underbrace{\lfloor \frac{f+L}{2} \rfloor}_{m} < L$.

Thus, $0\leq f\leq m < m+1 \leq L < Len(A)$, and so Preconds of both recursive calls holds.

Assume the while loop on Line 7 terminates after t iterations. By loop_exit cond. $i_t > L$. By $LI(t)$ $i_t \leq L+1$.

So $i_t = L+1$. Also, by $LI(t)$, C_t is equal to the number of occurrences of x in $A[f : i_t] = A[f : L+1]$. (1)



$n = K_1 + K_2$

$K_1 = Len(A[f : m+1])$

$K_2 = Len(A[m+1 : L+1])$

By the Fact and Assumption (*), either b occurs more than $\frac{K_1}{2}$ in $A[f : m+1]$ or b occurs more than $\frac{K_2}{2}$ in $A[m+1 : L+1]$.

Case 1: b occurs more than $\frac{k-1}{2}$ in $A[f : m+1]$.

Then, by IH, maj(A, f, m) terminates and set x to b (Line 4).

So, by (1), after the termination of the loop c_t is equal to the number of occurreces of b in $A[f : L+1]$. Thus $c_t > \frac{n}{2} = \frac{L-f+1}{2}$ and so the if_cond on Line 11 holds. So maj(A, f, L) terminates on Line 12 and returns x=b.

Case 2: b occurs more than $\frac{k_2}{2}$ times in $A[m+1 : L+1]$.

Case 2.1: $maj(A, f, m)$ (on Line 4) returns b ($x = b$)

Similar to case 1, $maj(A, f, L)$ terminates on Line 12 and returns b.

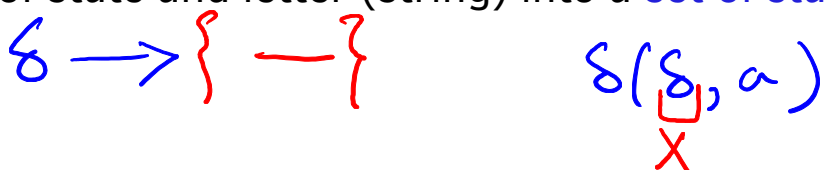
Case 2.2: $x \neq b$. Then x occurs fewer than $n/2$ times in $A[f : L+1]$. So, $c_t < \frac{n}{2} = \frac{L-f+1}{2}$. Thus the if-cond on Line 11 fails and $maj(A, f, L)$ executes Line 13. Then, by IH, $maj(A, m+1, L)$ terminates and returns b.

Review – DFA's, NFA's

- A **DFA** has exactly one transition rule for each symbol at each state.
  An **NFA** has at least one transition rule for each symbol at each state, and may also have $\epsilon-$transition.

- A **DFA** accepts a string w iff w take the DFA from the initial state to an accepting state.
  An **NFA** accepts a string w iff at least one of the possible states in which the NFA could be after processing input w is an accepting state.

- The transition function (and extended transition function) of a **DFA** maps each pair of state and letter (string) into one state.
  The transition function (and extended transition function) of an **NFA** maps each pair of state and letter (string) into a set of states.



- **Convention:** if at a state q there is no arrow for a possible symbol b, we assume that b takes the DFA/NFA from q to a **dead state**.
  - A **dead state** is a rejecting state that is basically a dead end.

Review – Proving Regularity of Languages

- A language L is **regular** iff there exists a **DFA/NFA/regex** that accepts/generates it.
- Proving L is regular:
  - If L is explicitly defined, then design a DFA/NFA/regex that accepts/generates L.
  - If L is defines in terms of other regular languages like L_1 and L_2
    * Show that L is obtained by applying operations that preserve regularity of languages (e.g., Union, Intersection, Concatenation, Complementation, Kleene Star, et.c) over some regular languages (e.g., L_1 and L_2).
    * Design an FSA for L based on FSA's of L_1 and L_2 and then prove the correctness of your FSA.
- **Correctness** of an FSA $\mathcal{M}$ w.r.t. a language L:

$$\mathcal{L}(\mathcal{M}) = L$$

$$\text{For all } w \in \Sigma^*,\ \left(w \in \mathcal{L}(\mathcal{M}) \quad \text{iff} \quad w \in L\right)$$

Review – Proving Irregularity of Languages

- A language L is **regular** iff there exists a **DFA/NFA/regex** that accepts/generates it.

- Proving L is not regular:

  - For a contradiction assume L is regular, and so correspond with a DFA with p states.
    Show that for every positive natural number p, there exists $w \in L$ with $|w| \geq p$ s.t. w fails at least one of the conditions of the Pumping Lemma.

  - Must define w in terms of p.

  - Usually, it's easier to assume that conditions (1) – (3) holds and show that condition (4) fails.

Review – Regular Expressions

- The **only** symbols that can be used in regular expressions are:
  $\varnothing$, ϵ, symbols in Σ, $*$, concatenation, $+$

$(01)^n$ ✗

- $*$ has higher precedent than concatenation,
  concatenation has higher precedent than $+$

$(01)^*$

01^*

- Do **NOT** use quantifiers over variables of a predicate inside the definition of the predicate.

✗ P(n): for all n∈ℕ, T(n) = ...

P(5): for all 5∈ℕ

- In inductive proofs, make sure to **explicitly mention IH** wherever you use it.