Algorithms and Data Structures (CSci 115)

California State University Fresno
College of Science and Mathematics
Department of Computer Science
H. Cecotti

Learning outcomes

- Divide and conquer principles
- Master theorem
  - Definition + application

  - See section 1.3.4 in the pdf on Canvas

Rationale

- Many algorithms: **recursive** in structure
  - to solve a given problem, they call themselves **recursively** 1 or more times to deal with closely related sub-problems
  - They typically have a ***divide-and-conquer*** approach:
    1. You break the main problem into several sub-problems
       - **similar** to the original problem but smaller in size
    2. You solve the sub-problems recursively
    3. You combine these solutions to create a solution to the original problem

Divide and conquer

1. **Divide** the problem into a number of sub-problems that are smaller instances of the same problem.
2. **Conquer** the sub-problems by solving them recursively.
   - ➢ If the sub-problems are large enough to solve recursively
     - the ***recursive case***.
   - ➢ If the sub-problem sizes are small enough
     - the **base case**
     - → solve the sub-problems in a straightforward manner.
3. **Combine** the solutions to the sub-problems into the solution for the original problem.

Loop invariant

- What to show:
  - **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 gives us a useful property that helps show that the algorithm is correct.
- **Loop invariant**
  - Condition that is true **before** the loop, **and after each iteration**
    - Example: Array is sorted between n and m before, Array is still sorted after each iteration ☺
- Goal
  - To show that an algorithm is correct

Recurrences

- Recurrence
  - An equation or inequality describing a function in terms of its value on smaller inputs
  - A natural way to characterize the running times of divide-and-conquer algorithms
- Remark
  - Sub-problems
    - Unequal sizes
      - a 3/4-to-1/4 split
    - not necessarily constrained to being a **constant** fraction of the original problem size.

Methods for solving recurrences

- **Goal**: to obtain the asymptotic Θ or *O* bound on the solution
  - Substitution method
    - Guess a bound and then use math **induction** to prove that the hypothesis is correct
  - Recursion-tree method
    - Conversion of the recurrence into a tree
      - Nodes: costs incurred at the levels of the recursion/tree
  - Master method
    - $T(n)=aT(n/b)+f(n)$
      - Divide and conquer algorithm
        - n: size of an input problem
        - Creation of *a* sub-problems
        - Each sub-problem is *1/b* the size of the original main problem
        - Divide + Combine operations = *f(n)*
    - With **$a \geq 1$, $b > 1$, $f(n)$ a given function**
    - $T(n)=\Theta(1)$ when n > k (some bound) > 0
      - Smallest input size leading to a recursive call

Master theorem

- **3 cases**
  - Recursion tree is **leaf heavy** ($aT(n/b) > f(n)$)
  - Split/recombine, **same** as sub-problems ($aT(n/b) = f(n)$)
  - Recursion tree is **root heavy** ($aT(n/b) < f(n)$)
- $C_{crit} = \log_b a$

  $= \log(\#\text{subproblems})/\log(\text{relative subproblem size})$

crit = critical exponent

Master theorem: Case 1

- **Description**
  - Work to split/recombine a problem is dwarfed by sub-problems
    - the recursion tree is leaf-heavy
- Condition on f(n) based on $\log_b a$
  - if $f(n)=O(n^c)$ where $\mathbf{c<c_{crit}}$
  - Then $T(n)=\Theta(n^{c_{crit}})$
    - Recursive tree structure dominates
- Example
  - If $b=a^2$ and $f(n)=O(n^{1/2-\varepsilon})$ → $c_{crit}=1/2$
    - The ε makes f(n) less "complex" than the left side of T(n)
    - $b=a^2$ → Examples: 2T(n/4) , 4T(n/16)
  - Then $T(n)= \Theta(n^{1/2}) = \Theta(sqrt(n))$

Master theorem: Case 2

- **Description**
  - Work to split/recombine a problem is “**comparable**” to sub-problems
- Condition on f(n) based on $\log_b a$
  - If $f(n)=\Theta(n^{ccrit} \log^k n)\ \forall\ k \geq 0$ // check for what small k it is true
  - Then $T(n)= \Theta(n^{ccrit} \log^{k+1} n)$
    - **Log augmented by a power 1**
- Example 1
  - If $b=a^2$ and $f(n)= \Theta(n^{1/2})$
  - Then $T(n)=\Theta(n^{1/2}\log n)$
- Example 2
  - If $b=a^2$ and $f(n)= \Theta(n^{1/2} \log n)$
    - $b=a^2 \rightarrow 2T(b/4)$, $4T(b/16)$
  - Then $T(n)=\Theta(n^{1/2}\log^2 n)$

Master theorem: Case 3

- **Description**
  - To split/recombine a problem dominates sub-problems
- Condition on f(n) based on $\log_b a$
  - When $f(n)=\Omega(n^c)$ where $\mathbf{c>c_{crit}}$
  - $a.f(n/b) \leq k.f(n)$ for a constant $k<1$ and n large enough
    - Warning: it is $a.f(n/b) \leq k.f(n)$ not $a.T(n/b) \leq k.f(n)$
    - → You substitute f(n) to the left side
  - Then it is dominated by the splitting term f(n)
  - → $T(n)=\Theta(f(n))$
- Example
  - If $b=a^2$ and $f(n)= \Omega(n^{1/2+\varepsilon})$
  - Then $T(n)= \Theta(f(n))$

Examples:

- To do for practice:
  - $T(n)=8T(n/2)+1000n^2$
    - **Case 1**: $a=8$, $b=2$, $c_{crit}=3$, $f(n)=O(n^2)$ → $c=2$, $c<c_{crit}$
  - $T(n)=2T(n/2)+10n$
    - **Case 2**: $a=2$, $b=2$, $c_{crit}=1$, $f(n)= \Theta(n)$ → $c=1$
  - $T(n)=2T(n/2)+n^2$
    - **Case 3**: $a=2$, $b=2$, $c_{crit}=1$, $f(n)= \Omega(n^2)$ → $c=2$, $c>c_{crit}$
      - $a.f(n/b) \leq k.f(n)$ → $2(f(n)/4) \leq 1/2f(n)$ (works with $k=1/2$)
      - $2f(n/2) \leq k.f(n) = 2n^2/4 = n^2/2 \leq 1/2 * n^2$
- Steps to follow:
  1. Does it satisfy the rules (positive terms)?
  2. What rule you have to apply?
  3. Apply the rule

Master theorem (in 1 slide)

- $T(n) = aT(n/b) + f(n)$
  - where
    - $a \geq 1$ and $b > 1$ are constants
    - $f(n)$ is an asymptotically positive function
- 3 cases
  1. If $f(n) = O(n^{\log_b a - \varepsilon})$ for some constant $\varepsilon > 0$
     - then $T(n) = \Theta(n^{\log_b a}) = \Theta(n^{c_{crit}})$
  2. If $f(n) = \Theta(n^{\log_b a} \log^k n)$ with $k \geq 0$
     - then $T(n) = \Theta(n^{\log_b a} \log^{k+1} n)$
  3. If $f(n) = \Omega(n^{\log_b a + \varepsilon})$ with $\varepsilon > 0$, and $f(n)$ satisfies the regularity condition,
     - then $T(n) = \Theta(f(n))$.
     - Regularity condition: $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n.

Conclusion

- What you need to know!
  - **Definitions**:
    - Divide and Conquer (and Combine)
    - Master Theorem
  - **Targets**: Translate an algorithm into the Master theorem equations
    - You must be able to apply the Master theorem

    (it will happen in the midterm and final exam)
- More examples during the lab.

Questions ?

- Reading
  - ➢CSci115 book: section 1.3.4
  - ➢Chapter **4**, “Introduction to Algorithms”, 3rd Edition