Lecture 10

Mathematical Analysis of Recursive Algorithms and Solving Recurrences: Iteration Method, Substitution Method

MergeSort Algorithm Process

Here is one way to implement merge:

- Create an empty list called the result list.
- Do the following until one of the input lists is empty:
  - Remove the first element of the list that has a lesser first element and append it to the result list.
- When one of the lists is empty, append all elements of the other list to the result.

Use the merge algorithm to find the third step of the merge of A and B.
Here are the first two steps:

Initial State	A = [2,4,9]	B = [1,7,13,15]	Results = []
First Step	A = [2,4,9]	B = [7,13,15]	Results = [1]
Second Step	A =[4,9]	B = [7,13,15]	Results = [1,2]

- ◯ A = [9] B = [7,13,15] Results = [1,2]
- ◯ A = [9] B = [13,15] Results = [1,2,4,7]
- ◯ A = [] B = [] Results = [1,2,4,7,9,13,15]
- ◯ A = [9] B = [7,13,15] Results = [1,2,4]

MergeSort Algorithm Process

› For the third step, you compare the smallest of the first (smallest) elements of A and B, move this element over to Results. In this case, the smaller value is 4 (in element A), which gets moved to the end of Results.

$$\boxed{A = [9], B = [7, 13, 15], \text{Results} = [1, 2, 4]}$$

The 3rd STEP of MERGE algorithm for merging A and B is as below

Initial State	A = [2,4,9]	B = [1,7,13,15]	Results = [ ]
First Step	A = [2,4,9]	B = [7,13,15]	Results = [1]
Second Step	A =[4,9]	B = [7,13,15]	Results = [1,2]

- A = [9] B = [7,13,15] Results = [1,2]
- A = [9] B = [13,15] Results = [1,2,4,7]
- A = [ ] B = [ ] Results = [1,2,4,7,9,13,15]

The correct of the 3rd STEP of merging A and B is

- ✓ A = [9] B = [7,13,15] Results = [1,2,4]

MergeSort Steps to Implement in Python

```python
def merge(left, right):
    result = []
    left_idx, right_idx = 0, 0
    while left_idx < len(left) and right_idx < len(right):
        # change the direction of this comparison to change the direction of the sort
        if left[left_idx] <= right[right_idx]:
            result.append(left[left_idx])
            left_idx += 1
        else:
            result.append(right[right_idx])
            right_idx += 1

    if left:
        result.extend(left[left_idx:])
    if right:
        result.extend(right[right_idx:])
    return result

def merge_sort(m):
    if len(m) <= 1:
        return m

    middle = len(m) // 2
    left = m[:middle]
    right = m[middle:]

    left = merge_sort(left)
    right = merge_sort(right)
    return list(merge(left, right))
```

Recurrence relation for complexity analysis

- Already discussed the way to perform analysis of loops
- Many algorithms are recursive, so recurrence relation for time complexity.
- We find running time on input size n (smaller sizes) as a function of n, e.g., Merge Sort
  - In merge sort, array is divided in two halves with recursive repetition until to get merge results
  - Thus, $T(n)=2T(n/2) +cn$
  - Other algorithms are Binary search, Tower of Hanoi etc.

Information about Sorting Algorithms:

3-way Merge Sort, Selection Sort, Bubble Sort, Insertion Sort, Merge Sort, Heap Sort, QuickSort, Radix Sort, Counting Sort, Bucket Sort, ShellSort, Comb Sort

Recurrence Methods

1. Iteration Method
2. Substitution Method
3. Recurrence Tree Method
4. Master Theorem

The Iteration Method

- *Convert the recurrence into a summation and try to bound it using known series*
  - *Iterate the recurrence until the initial condition is reached.*
  - *Use back-substitution to express the recurrence in terms of n and the initial (boundary) condition.*

The Iteration Method (Cont !!!)

$$\boldsymbol{T(n) = c + T(n/2)}$$

$$T(n/2) = c + T(n/4)$$

$$T(n/4) = c + T(n/8)$$

$$T(n) = c + T(n/2)$$

$$= c + c + T(n/4)$$

$$= c + c + c + T(n/8)$$

Assume $n = 2^k$

$$T(n) = \underbrace{c + c + \ldots + c}_{k\ times} + T(1)$$

$n/2^i=1$, $=> n=2^i => i=lg_2\, n/lg_2\, 2=lg_2\, n$

$$= ck+T(1) = clgn + T(1)\ ,\quad \text{Assume that } k=lg\ n$$

$$= \Theta(lgn)$$

Iteration Method – Example

$$T(n) = n + 2T(n/2)$$

Assume: $n = 2^k$

$$T(n/2) = n/2 + 2T(n/4)$$

$$\begin{aligned}
T(n) &= n + 2T(n/2) \\
&= n + 2(n/2 + 2T(n/4)) \\
&= n + n + 4T(n/4) \\
&= n + n + 4(n/4 + 2T(n/8)) \\
&= n + n + n + 8T(n/8) \\
\ldots &= in + 2^i T(n/2^i) \\
&= kn + 2^k T(1) \\
&= n\lg n + nT(1) = \Theta(n\lg n)
\end{aligned}$$

Substitution Method

- The substitution method for solving recurrences can be described in two steps:
  - Guess the form of the solution.
  - Use induction to show that the guess is valid.
- This method is especially powerful when we encounter recurrences that are non-trivial and unreadable via the master theorem.
- Substitution method can be used to establish both upper and lower bounds on recurrences.
- The name comes from the substitution of the guessed answer for the function when the inductive hypothesis is applied to smaller values.
- This method is powerful, but it is only applicable to instances where the solutions can be guessed.

Substitution method (Cont !!!)

- *Guess a solution*
  - $T(n) = O(g(n))$
  - *Induction goal: apply the definition of the asymptotic notation*
    - $T(n) \leq d\ g(n)$, *for some* $d > 0$ *and* $n \geq n_0$
  - *Induction hypothesis:* $T(k) \leq d\ g(k)$ *for all* $k < n$
- *Prove the induction goal*
  - *Use the **induction hypothesis** to find some values of the constants d and n_0 for which the **induction goal** holds*

Substitution Method: Example

- *Consider the recurrence T(n) = 2T(n/2) + n to show that* is O(n lg(n)).
- *Solution:*
  - *To prove* $T(n) \leq cn\, lg(n)$, we assume that bound holds for $\frac{n}{2}$.
  - $T(n) \leq c(\frac{n}{2})\, lg(n/2)$ + n
  - $\leq cn\, lg(n/2)$ + n
  - $\leq cn\, lg(n)$ - $cn\, lg(2)$ + n
  - $\leq cn\, lg(n)$ + n(1- $clg(2)$)
  - $\leq cn\, lg(n)$ + n(1- c)
  - $\leq cn\, lg(n)$, for any c>1. we are done with it.

Recurrence Relation [T(n)= n*T(n-1)] Substitution Method

- $T(n) = \begin{cases} 1 & if\ n = 1\ \ldots\ldots\ldots\ldots\ldots..\ Base\ Condition \\ n * T(n-1) & if\ n > 1\ \ldots\ldots\ldots\ldots recurrence\ relation \end{cases}$
- Solution
  - $T(n) = n * T(n-1)$ ---→ (1)
  - $T(n-1) = (n-1) * T\big((n-1) - 1\big)$
  - $T(n-1)$ $= (n-1) * T\big((n-2)\big)$ ---→(2)
  - $T(n-2) = (n-2) * T(n-3)$ ---→(3)
  - Substitute (3) in (2) and then (2) in (1) to know the trend as below
  - $T(n) = n * (n-1) * (n-2)T(n-3)$ To eliminate $T(n-3)$ take it upto n-1 steps.
  - $T(n) = n * (n-1) * (n-2) * (n-3) \ldots T\big(n - (n-1)\big)$
  - $T(n) = n * (n-1) * (n-2) * (n-3) \ldots T(n - n + 1)$
  - $T(n) = n * (n-1) * (n-2) * (n-3) \ldots T(1)$
  - $T(n) = n * (n-1) * (n-2) * (n-3) \ldots 3 * 2 * 1$
  - $= n * n\left(1 - \frac{1}{n}\right) * n\left(1 - \frac{2}{n}\right) * n\left(1 - \frac{3}{n}\right) \ldots n\left(\frac{3}{n}\right) * n\left(\frac{2}{n}\right) * n(\frac{1}{n})$
  - $T(n) = O(n^n)$ which is a factorial time multiplication

Example: Binary Search

$$\boldsymbol{T(n) = c + T(n/2)}$$

- *Guess:* $T(n) = O(lgn)$
  - *Induction goal:* $T(n) \leq d\ lgn$, *for some* d *and* $n \geq n_0$
  - *Induction hypothesis:* $T(n/2) \leq d\ lg(n/2)$
- *Proof of induction goal:*

$$T(n) = T(n/2) + c \leq d\ lg(n/2) + c$$

$$= d\ lgn - d + c \leq d\ lgn \text{ if: } -d + c \leq 0,\ d \geq c$$

Example

$$\boldsymbol{T(n) = T(n-1) + n}$$

- *Guess:* $T(n) = O(n^2)$
  - *Induction goal:* $T(n) \leq c\, n^2$, *for some* c *and* $n \geq n_0$
  - *Induction hypothesis:* $T(n-1) \leq c(n-1)^2$ *for all* $k < n$
- *Proof of induction goal:*

$$T(n) = T(n-1) + n \leq c\,(n-1)^2 + n$$

$$= cn^2 - (2cn - c + n) \leq cn^2$$

$$\textit{if:}\ \ 2cn - c + n \geq 0 \Leftrightarrow c \geq n/(2n-1) \Leftrightarrow c \geq 1/(2 - 1/n)$$

  - *For* $n \geq 1 \Rightarrow 2 - 1/n \geq 1 \Rightarrow$ *any* $c \geq 1$ *will work*

Example

$T(n) = 3T(n/4) + cn^2$

- *Guess:* $T(n) = O(n^2)$
  - *Induction goal:* $T(n) \leq dn^2$, *for some* d *and* $n \geq n_0$
  - *Induction hypothesis:* $T(n/4) \leq d\ (n/4)^2$
- *Proof of induction goal:*

$$T(n) = 3T(n/4) + cn^2$$

$$\leq 3d\ (n/4)^2 + cn^2$$

$$= (3/16)\ d\ n^2 + cn^2$$

$$\leq d\ n^2 \qquad \text{if: } d \geq (16/3)c$$

- *Therefore:* $T(n) = O(n^2)$

Substitution Method: Example

- *void RecRel (int n)* ------ *T(n)*
  - *IF n>1*
    - *FOR (i=0;i<n, i++)* ------- *n*
      - *<< Statements>>*
    - *RecRel(n/2)* ------ *n/2*
    - *RecRel(n/2)* ------ *n/2*

$$T(n)=\begin{cases} \mathbf{1} & \boldsymbol{if\ n=1} \\ \boldsymbol{2T\left(\frac{n}{2}\right)+n} & \boldsymbol{if\ n>1} \end{cases}$$

$$T(n) = 2T\left(\frac{n}{2}\right) + n2T\left(\frac{n}{2}\right) + n \quad \dashrightarrow (1)$$

$$= 2[2T\left(\frac{n}{2^2}\right) + \frac{n}{2}] + n$$

$$= 2^2(T\left(\frac{n}{2^2}\right)) + n + n \quad \dashrightarrow (2)$$

$$= 2^2[2T\left(\frac{n}{2^3}\right) + \frac{n}{2^2}] + 2n$$

$$= 2^3(T\left(\frac{n}{2^2}\right)) + 3n \quad \dashrightarrow (3)$$

$$T(n) = 2^k(T\left(\frac{n}{2^k}\right)) + kn$$

Assume

$$T\left(\frac{n}{2^k}\right) = T(1)$$

$\therefore n/2^k = 1$, $n=2^k \rightarrow k=\log n$

$$T(n) = 2^k T(1) + kn$$

$$T(n) = nX1 + n\log n = \boldsymbol{O(n\log n)}$$

Thank You!!!

Have a good day