Lecture 6: Recurrences

CSE 332: Data Structures & Parallelism

Yafqa Khan

Summer 2025

Announcements

- EX 1 Due Today
- EX 2 Due Friday
- EX 3 Released Today
- Exam 1 Week 5 (Friday)

Today

- Asymptotic Analysis: Recursion
  - Writing a Recurrence Relation
  - Solving a Recurrence Relation 1: Unrolling
  - Solving a Recurrence Relation 2: Tree Method

Recap: Counting code constructs

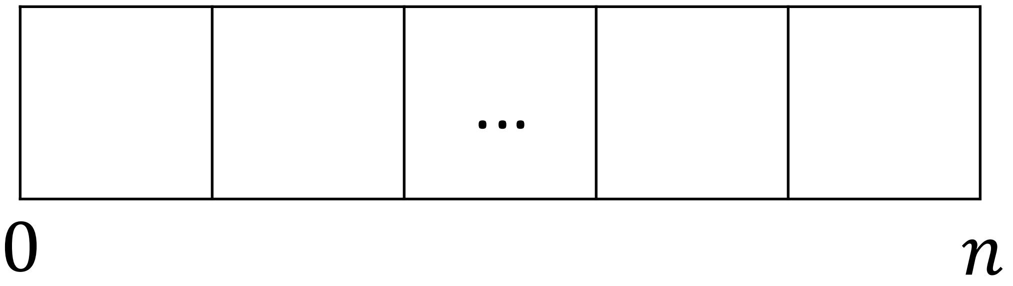
Assume basic operations take some amount of constant time

- Arithmetic (1+1), assignment (int b = 3), array index(arr[i]), etc.

This approximates reality: a very useful "lie"

Code Construct	How much Time?
Consecutive Statements	Sum of time of each statement
Loops	Sum of loop body iterations
Conditionals	Time of condition + executed branch
Function (method) Calls	Time of function’s body
Recursion	Solve recurrence equation

Counting Recursive Code



- Very interesting with recursive code
- Analogy: Perform some computation recursively on a list of size n
  - Each recursive method call:
    - Perform some **Non-Recursive work** $w(n)$
    - Call the method $T(n)$ with a smaller part of the list $T(n-1)$
  - Base Case: **base case work** $b(n)$
- So, if we do $w(n)$ work per step and use a smaller list with 1 less element, we do total work:
  - $T(n) = w(n) + T(n-1)$
- Eventually base case with 1 element that does base case work $b(n)$
  - $T(1) = b(n)$

Recurrence: Terminology

Terminology	Recurrence Function/Relation	General formula	Closed form
Definition	Piecewise function that **mathematically models** the runtime of a recursive algorithm (might want to define constants)	Function written as the number of expansion i and recurrence function (might have summations)	General formula evaluated without recurrence function or summations (force them to be in terms of constants or n)
Example	$T(n) = \begin{cases} c_0 & \text{for } n = 1 \\ T\left(\frac{n}{2}\right) + c_1 & \text{otherwise} \end{cases}$	$T(n) = T\left(\frac{n}{2^i}\right) + i \cdot c_1$	Let $i = \log n$, $T(n) = T\left(\frac{n}{2^{\log n}}\right) + \log n \cdot c_1$ $= T(1) + \log n \cdot c_1$ $= \boldsymbol{c_0 + \log n \cdot c_1}$

Writing a Recurrence Function/Relation

```
int sum(int[]arr,int n) {
  if(n==0)
    return arr[n];
  return arr[n] + sum(arr,n-1);
}
```

Q: How can we count `sum(arr,arr.length);`?

A: By using recursive formulas (called recurrence function/relation)!

1. Base Case Work

2. Non-Recursive Work + Recursive Work

Any Questions?

Today

- Asymptotic Analysis: Recursive
  - Writing a Recurrence Relation
  - Solving a Recurrence Relation 1: Unrolling
  - Solving a Recurrence Relation 2: Tree Method

Solving Recurrence 1: Unrolling

1. Write a Recurrence

$$T(n) = \begin{cases} c_0, & \text{for } n = 1 \\ c_1 + T(n-1), & \text{otherwise} \end{cases}$$

2. Find **General Formula**
   1. Expand:

$$\begin{aligned} T(n) &= c_1 + T(n-1) \\ &= c_1 + \left(c_1 + T\left((n-1)-1\right)\right) = c_1 + c_1 + T(n-2) \\ &= c_1 + c_1 + \left(c_1 + T\left((n-2)-1\right)\right) = c_1 + c_1 + c_1 + T(n-3) \\ &= \cdots \\ &= \boldsymbol{c_1 i + T(n-i)} \end{aligned}$$

3. Find **Closed Form**
   1. Find when the base case occurs
      - When $n - i = 1$ (i.e., $i = n - 1$)
   2. Get to the base case

$$T(n) = c_1(n-1) + T(1) = \boldsymbol{c_1 n - c_1 + c_0}$$

4. Asymptotic Analysis

$$T(n) \in \Theta(n)$$

Any Questions?

Unrolling: Example, Binary Search

2	3	5	16	37	50	73	75	126

Find an integer in a *sorted* array

```
// returns whether k is in array
boolean binarySearch(int[]arr, int k, int lo, int hi) {
   int mid = (hi+lo)/2; //i.e., lo+(hi-lo)/2
   if(lo==hi)        return false;
   if(arr[mid]==k)   return true;
   if(arr[mid]< k)   return binarySearch(arr,k,mid+1,hi);
   else              return binarySearch(arr,k,lo,mid);
}
```

Q: Cost of `binarySearch(arr,k,0,arr.length);`?

1. Write a Recurrence

$$T(n) = \begin{cases} \end{cases}$$

Unrolling: Example

1. Write a Recurrence

$$T(n) = \begin{cases} c_0, & \text{for } n = 1 \\ c_1 + T\left(\frac{n}{2}\right), & \text{otherwise} \end{cases}$$

2. Find **General Formula**
   - Expand:

$$T(n) = c_1 + T\left(\frac{n}{2}\right)$$
$$=$$
$$=$$
$$=$$

3. Find **Closed Form**
   - Find when the base case occurs

   - Get to the base case

$$T(n) =$$

4. Finding Big-Theta

$$T(n) \in \Theta(\quad)$$

Unrolling: Example (Soln.)

1. Write a Recurrence

$$T(n) = \begin{cases} c_0, & \text{for } n = 1 \\ c_1 + T\left(\frac{n}{2}\right), & \text{otherwise} \end{cases}$$

2. Find **General Formula**
   - Expand:

$$T(n) = c_1 + T\left(\frac{n}{2}\right)$$

$$= c_1 + \left(c_1 + T\left(\frac{\frac{n}{2}}{2}\right)\right) = c_1 + c_1 + T\left(\frac{n}{4}\right) \qquad = c_1 + c_1 + \left(c_1 + T\left(\frac{\frac{n}{4}}{2}\right)\right) = c_1 + c_1 + c_1 + T\left(\frac{n}{8}\right)$$

$$= \cdots = c_1 i + T\left(\frac{n}{2^i}\right)$$

3. Find **Closed Form**
   - Find when the base case occurs
     - When $\frac{n}{2^i} = 1$ (i.e., $i = \log n$)
   - Get to the base case

$$T(n) = c_1 i + T\left(\frac{n}{2^i}\right) = c_1 \log n + T(1) = c_1 \log n + c_0$$

4. Finding Big-Theta

$$T(n) \in \Theta(\quad)$$

Iterative vs Recursive: `sum()`

- Iterative `sum()`:
  - "Obviously" linear

```
int sum(int[] arr){
  int ans = 0;
  for(int i=0; i<arr.length; ++i)
     ans += arr[i];
  return ans;
}
```

- Recursive `sum()`:
  - Recurrence is $c_1 + c_1 + \cdots + c_1 + c_0$ for n times so linear

```
int sum(int[]arr,int n) {
  if(n==0)
    return arr[n];
  return arr[n] + sum(arr,n-1);
}
```

Any Questions?

Recap: Algorithm Analysis of Recursive Code

```
boolean binarySearch(int[]arr, int k, int lo, int hi) {
    int mid = (hi+lo)/2;
    if(lo==hi)        return false;
    if(arr[mid]==k)   return true;
    if(arr[mid]< k)   return binarySearch(arr,k,mid+1,hi);
    else              return binarySearch(arr,k,lo,mid);
}
```

- Writing Recurrences (i.e., count recursive code)
  1. Split to Cases
     - Base Case: Base Case Work (e.g., $T(1) = c_0$)
     - Recursive Case: Non-Recursive + Recursive Work (e.g., $T(n) = c_1 + T\left(\frac{n}{2}\right)$)
- Solving Recurrences (e.g., specifically with **Unrolling** here)
  2. Find **General Formula**
     - Expand by substitution until pattern emerges (e.g., $T(n) = \cdots = c_1 i + T\left(\frac{n}{2^i}\right)$)
  3. Find **Closed Form**
     - Find when the base case occurs (e.g., when $T\left(\frac{n}{2^i}\right)$ = base case = $T(1)$ (i.e., $\frac{n}{2^i} = 1$ or $i = \log n$))
     - Get to the base case (e.g., Substitute $i = \log n$ to $T(n) = c_1 i + T\left(\frac{n}{2^i}\right) = c_1 \log n + T(1) = c_1 \log n + c_0$)
- Asymptotic Analysis or Finding Big-Theta (e.g., informally as here)
  4. $T(n) = \cancel{c_1} \log n + \cancel{c_0} \in \Theta(\log n)$

Any Questions?

Iterative vs Recursive: `sum()`

- Iterative:
  - "Obviously" linear

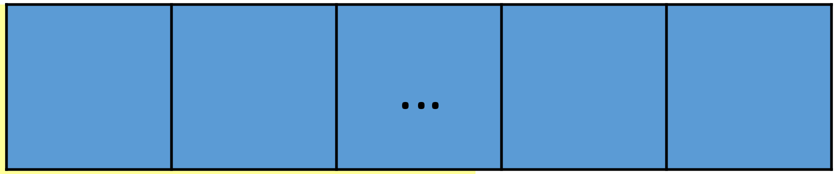
```
int sum(int[] arr){
  int ans = 0;
  for(int i=0; i<arr.length; ++i)
     ans += arr[i];
  return ans;
}
```

- Recursive:
  - Recurrence is $c_1 + c_1 + \cdots + c_1 + c_0$ for n times so linear

```
int sum(int[]arr,int n) {
  if(n==0)
    return arr[n];
  return arr[n] + sum(arr,n-1);
}
```

`sum` but weird (`binarySum`)

```
int binarySum(int[] arr, int lo, int hi) {
    if(lo==hi)    return 0;
    if(lo==hi-1)  return arr[lo];
    int mid = (hi+lo)/2;
    return binarySum(arr,lo,mid) + binarySum(arr,mid,hi);
}
```



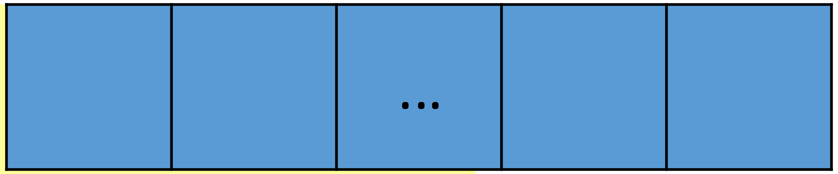
Q: How can we count `binarySum(arr,arr.length);`?

1. Write a Recurrence

$$T(n) = \begin{cases} \\ \end{cases}$$

sum but weird (binarySum)

```
int binarySum(int[] arr, int lo, int hi) {
    if(lo==hi)   return 0;
    if(lo==hi-1) return arr[lo];
    int mid = (hi+lo)/2;
    return binarySum(arr,lo,mid) + binarySum(arr,mid,hi);
}
```



Q: How can we count `binarySum(arr,arr.length);`?

1. Write a Recurrence

$$T(n) = \begin{cases} c_0, & \text{for } n = 1 \\ c_1 + 2T\left(\frac{n}{2}\right), & \text{otherwise} \end{cases}$$

How to solve? 💀 Good luck doing Unrolling

2. Find General Formula: Draw Tree

1. Initialize Table
2. **Draw Actual Tree**
3. Misc. Details
   - Recursive Calls, # Nodes, Sum Work, etc.
4. Base Case
   - Find when the base case occurs
5. **Work Calculation**
   1. Total Base Case Work
   2. Total Non-Recursive + Recursive Work

1. Write a Recurrence

$$T(n) = \begin{cases} c_0, & \text{for } n = 1 \\ c_1 + 2T\left(\frac{n}{2}\right), & \text{otherwise} \end{cases}$$

2. Draw Tree

1. Write a Recurrence $T(n) = \begin{cases} c_0, & \text{for } n = 1 \\ c_1 + 2T\left(\frac{n}{2}\right), & \text{otherwise} \end{cases}$

i	Recursive Call	# Nodes	Tree	Sum Work
0	$T(n)$			
1				
2				
$\vdots$	$\vdots$	$\vdots$		
i			+	

2. Total Non-Recursive + Recursive Work:

$\vdots$	$\vdots$	$\vdots$		$\vdots$
	$T(1)$			

Find when the base case occurs:

1. Total Base Case Work:

2. Draw Tree (Soln.)

1. Write a Recurrence $T(n) = \begin{cases} c_0, & \text{for } n = 1 \\ c_1 + 2T\left(\frac{n}{2}\right), & \text{otherwise} \end{cases}$

i	Recursive Call	# Nodes	Tree	Sum Work
0	$T(n)$	1	c_1	$= c_1$
1	$T\left(\frac{n}{2}\right)$	2	$c_1 + c_1$	$= 2c_1$
2	$T\left(\frac{n}{2^2}\right)$	2^2	$c_1 + c_1 + c_1 + c_1$	$= 4c_1$
⋮	⋮	⋮		⋮
i	$T\left(\frac{n}{2^i}\right)$	2^i	$c_1 + c_1 + \cdots + c_1 + c_1$	$= 2^i c_1$

$+$

2. Total Non-Recursive + Recursive Work: $c_1 + 2c_1 + 4c_1 + \cdots + 2^i c_1 = \sum_{i=0}^{\log(n)-1} 2^i \cdot c_1$

⋮	⋮	⋮		⋮
$\log n$	$T(1)$	n	$c_0 + c_0 + \cdots + c_0 + c_0$	

Find when the base case occurs:
$\frac{n}{2^i} = 1$ (i.e., $i = \log n$)

1. Total Base Case Work: $c_0 + c_0 + \cdots + c_0 = \sum_{i=0}^{n-1} c_0$

Any Questions?

Solving Recurrence 2: Tree Method

2. Find **General Formula**

$$T(n) = \text{Total Base Case Work} + (\text{Total Total Non}-\text{Recursive}+\text{Recursive Work})$$
$$=$$

3. Find **Closed Form**

$$T(n) =$$
$$=$$
$$=$$
$$=$$

Finite Geometric Series!

$$\sum_{i=0}^{m} x^i = \frac{1 - x^{m+1}}{1 - x}$$

4. Finding Big-Theta

$$T(n) \in \Theta(\quad)$$

Solving Recurrence 2: Tree Method (Soln.)

2. Find **General Formula**

$$T(n) = \text{Total Base Case Work} + (\text{Total Non}-\text{Recursive}+\text{Recursive Work})$$

$$= \sum_{i=0}^{n-1} c_0 \qquad + \sum_{i=0}^{\log(n)-1} 2^i \cdot c_1$$

3. Find **Closed Form**

$$T(n) = c_0 n + c_1 \left(\frac{1 - 2^{\log n - 1 + 1}}{1 - 2} \right)$$

$$= c_0 n + c_1 \left(2^{\log n} - 1\right)$$

$$= c_0 n + c_1 (n - 1)$$

$$= (c_0 + c_1) n - c_1$$

Finite Geometric Series!

$$\sum_{i=0}^{m} x^i = \frac{1 - x^{m+1}}{1 - x}$$

4. Finding Big-Theta

$$T(n) \in \Theta(n)$$

Any Questions?

Common Recurrences (Memorize!)

Common Recurrence Function/Relation	Order of Growth	Example
$T(n) = T\left(\frac{n}{2}\right) + c$	$\in \Theta(\log n)$	Binary Search
$T(n) = 2T\left(\frac{n}{2}\right) + n$	$\in \Theta(n \log n)$	Merge Sort
$T(n) = T\left(\frac{n}{2}\right) + n$	$\in \Theta(n)$	
$T(n) = 2T\left(\frac{n}{2}\right) + c$	$\in \Theta(n)$	Recursive "binary" sum
$T(n) = T(n-1) + c$	$\in \Theta(n)$	Recursive sum
$T(n) = T(n-1) + n$	$\in \Theta(n^2)$	
$T(n) = 2T(n-1) + c$	$\in \Theta(2^n)$	

Any Questions?

Timeline

- Asymptotic Analysis: Recursive
  - Writing a Recurrence Relation
  - Solving a Recurrence Relation 1: Unrolling
  - Solving a Recurrence Relation 2: Tree Method
- Dictionary ADT
- Review: Binary Search Trees
  - Trees
  - Basics, Properties, Operations