CSCE 411: Design and Analysis of Algorithms

Lecture 1: Intro, Asymptotic Runtimes, Divide and Conquer

Date: January 13, 2026 *Nate Veldt, updated by Samson Zhou*

Course Logistics

- Read section 2.3, and chapter 4 for first week of classes.
- Read (or skim) chapters 1-3 to ensure familiarity with prerequisites
- Syllabus quiz is due Sat, Jan 17. HW 1 and intro video due Fri, Jan 23

1 Computational Problems and Algorithms

Definition 1. A ______________________________ is a general task defined by a specific type of __________ and an explanation for a desired ______________

A specific case of the problem is called an ______________

Example 1. Sorting

Input: A sequence of n numbers: $a_1, a_2, \ldots, a_n$

Output: A permutation σ of the input sequence so that

$$a_{\sigma(1)} \leq a_{\sigma(2)} \leq \cdots \leq a_{\sigma(n)}$$

An instance of this problem is the sequence

Example 2. Min element

Input: An array of n numbers: $[a_1, a_2, \ldots, a_n]$

Output: The smallest element in the array and its index.

Definition 2. An **algorithm** is a computational procedure that

- ______________________________
- ______________________________

An **algorithm** is said to be **correct** if it ______________________.

2 Asymptotic Runtime Analysis (Chapter 3)

2.1 Rules for runtime analysis

- n denotes the size of the input
- Each basic operation takes constant time
- We focus on the ____________________ runtime
- We only care about the ____________ of the runtime

2.2 Some initial examples

Question 1. Given an array of n items, find whether the array contains a negative number using the following steps:

```
for i = 1 to n do
    if a_i < 0 then
        Return (true, i)
    end if
end for
```

What is the runtime of this method?

- A $O(1)$
- B $O(n)$
- C $O(n^2)$
- D It depends
- E Other
- F Don't know, I need a reminder for how this works.

Question 2. Given an $n \times n$ matrix A, what is the runtime of summing the upper triangular portion using the following algorithm? (same answers).

```
sum = 0
for i = 1 to n do
    for j = i to n do
        sum = sum + a_ij
    end for
end for
Return sum
```

2.3 Formal Definitions

Let n be input size, and let f and g be functions over $\mathbb{N}$.

Definition 3. Big O notation.

A function $g(n) = O(f(n))$ (we say, "g is big-O of $f(n)$") means:

Definition 4. Big Ω notation.

$g(n) = \Omega(f(n))$ means:

Definition 5. Θ notation.

$g(n) = \Theta(f(n))$ means:

Equivalently, this means

Additional runtime examples

1. $4n^4 + n^3 \log n + 100n \log n$

2. $n + 2(\log n)^2$

3. $2^n + 10^{100} n^4 5$

Logarithms in Runtimes Which of the following runtimes are the same asymptotically? Which are not?

- $O(n \log n)$ and $O(n \lg n)$

- $O(\log n)$ and $O(\log^2 n)$

- $O(\log n)$ and $O(\log(n^2))$

- $O(n^{\log 100})$ and $O(n^{\lg 100})$

3 How to Present an Algorithm

Presenting and analyzing can be broken up into four steps.

1. **Explain**: the approach in basic English
2. **Pseudocode**: for formally presenting the algorithmic steps
3. Prove: the **correctness**
4. Analyze: the **runtime complexity**

As a rule it's a good idea to go through all steps when presenting an algorithm. Sometimes we will focus more on just a subset of these (e.g., you may be asked to prove a runtime complexity of an algorithm on a homework but not a correctness proof).

We will go through all four steps when we present the *merge sort* algorithm.

4 The Divide and Conquer Paradigm

The divide and conquer paradigm has three components:

- **Divide**:
- **Conquer**:
- **Combine**:

Example: Mergesort (Textbook, Chapter 2.3, 4) Given n numbers to sort, apply the following steps:

- Divide the sequence of length n into
- Recursively
- Combine

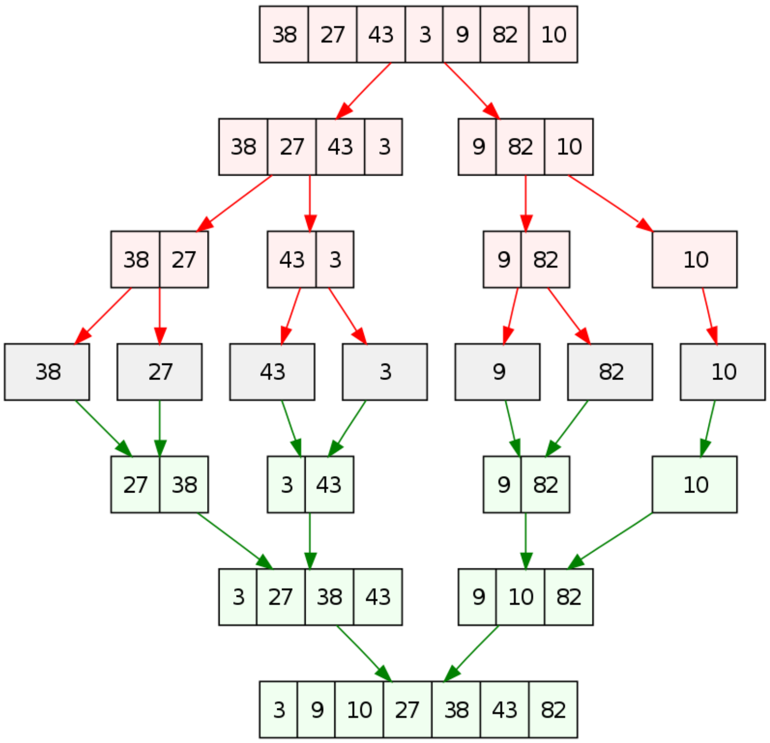

Image courtesy of Wikipedia: https://en.wikipedia.org/wiki/Merge_sort.

MergeSort(A)

  $n =$ **length**(A)

  if $n == 1$ **then**

  else

    $m = \lfloor n/2 \rfloor$

  end if

4.1 Analyzing The Merge Procedure

Correctness: To merge two sorted subarrays into a master array

- Maintain a pointer to the ______________________________
- At each step, ____
- Since subarrays are sorted, one of these numbers is ______________________________

  ____
- so place ____
- At each step, we guarantee that: ____________________________________
- So continuing until both subarrays are empty, ____________________________

5 Continued Analysis of Merge Sort

Merge Sort. Given n numbers to sort

- Divide the sequence of length n into arrays of length $\lceil n/2 \rceil$ and $\lfloor n/2 \rfloor$
- Recursively sort the two halves
- (Merge Procedure) Combine the two halves by sorting them.

Question 3. What is the runtime of the *merge procedure* in Merge Sort?

A $\Theta(1)$

B $\Theta(n \lg n)$

C $\Theta(n)$

D $\Theta(n^2)$

6 Recurrence Analysis for Divide and Conquer

Runtimes for divide and conquer algorithms can be described in terms of a ____________________

relation, which ____

Let $T(n)$ denote the runtime for a problem of size n.

Example: merge-sort Assume for this analysis that $n = 2^p$ where $p \in \mathbb{N}$.

7 Three methods for solving recurrences

Given a recurrence relation, there are three approaches to finding the overall runtime.

- **Recursion tree**:
- **Substitution method**:
- **Master theorem**:

8 The Master Theorem for Recurrence Relations

Theorem 8.1. *Let $a \geq 1$ and $b > 1$ be constants, let $f(n)$ be a function, and let $T(n)$ be defined on the nonnegative integers by the relation:*

____

1. *If $f(n) = O(n^{\log_b a-\epsilon})$ for some constant $\epsilon > 0$, then* ______________________________

2. *If $f(n) = \Theta(n^{\log_b a})$, then* ______________________________________

3. *If $f(n) = \Omega(n^{\log_b a+\epsilon})$ for some constant $\epsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then* ____________________________

8.1 Example: Merge-Sort

Recall that Merge-Sort satisfies the following recurrence:

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ 2T(n/2) + \Theta(n) & \text{if } n > 1 \end{cases} \tag{1}$$

We can apply the master theorem with:

8.2 What to know about the master method?

Proof idea:

A full proof can be found in Section 4.6 of the textbook.

What's more important:

8.3 Examples

Apply the master theorem to the following recurrences:

$$T(n) = 9T(n/3) + n \tag{2}$$

$$T(n) = 3T(n/4) + n \log n \tag{3}$$

$$T(n) = 7T(n/2) + \Theta(n^2) \tag{4}$$

9 Strassen's Algorithm for Matrix Multiplication

Let A and B be $n \times n$ matrices, and $C = AB$. The (i, j) entry of C is defined by

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$$

Algorithm 1 Simple Square Matrix Multiply

Input: $A, B \in \mathbb{R}^{n\times n}$
Output: $C = AB \in \mathbb{R}^{n\times n}$
Let $C = zeros(n, n)$
for $i = 1$ to n **do**
    for $j = 1$ to n **do**
        $c_{ij} = 0$
        for $i = 1$ to n **do**
            $c_{ij} = c_{ij} + a_{ik} b_{kj}$
        end for
    end for
end for
Return C

9.1 An attempt at divide-and conquer

Assume that $n = 2^p$ for some positive integer $p > 1$.

Algorithm 2 Simple Recursive Square Matrix Multiply (SSMM)

Input: $A, B \in \mathbb{R}^{n \times n}$
Output: $C = AB \in \mathbb{R}^{n \times n}$
if $n == 1$ **then**
  $c_{11} = a_{11} b_{11}$
else
  $C_{11} = SSMM(A_{11}, B_{11}) + SSMM(A_{12}, B_{21})$
  $C_{12} = SSMM(A_{11}, B_{12}) + SSMM(A_{12}, B_{22})$
  $C_{21} = SSMM(A_{21}, B_{11}) + SSMM(A_{22}, B_{21})$
  $C_{22} = SSMM(A_{21}, B_{12}) + SSMM(A_{22}, B_{22})$
end if
Return C

Question 4. What recursion applies to the above algorithm when $n > 1$?

A $T(n) = 4T(n/2) + O(n^2)$

B $T(n) = 8T(n/4) + O(n^2)$

C $T(n) = 8T(n/2) + O(n)$

D $T(n) = 8T(n/2) + O(n^2)$

9.2 Strassen's Algorithm

Strassen's algorithm introduces a new way to combine matrix multiplications and additions to obtain the matrix C.

Step 1: Partition A and B as before.

Step 2: Compute S matrices

$$\begin{aligned}
S_1 &= B_{12} - B_{22} \\
S_2 &= A_{11} + A_{12} \\
S_3 &= A_{21} + A_{22} \\
S_4 &= B_{21} - B_{11} \\
S_5 &= A_{11} + A_{22} \\
S_6 &= B_{11} + B_{22} \\
S_7 &= A_{12} - A_{22} \\
S_8 &= B_{21} + B_{22} \\
S_9 &= A_{11} - A_{21} \\
S_{10} &= B_{11} + B_{12}
\end{aligned}$$

Runtime: we add (or subtract) 2 matrices of size $n/2 \times n/2$, 10 times.

Step 3: Compute P matrices

$$\begin{aligned}
&P_1 = A_{11} \cdot S_1 = A_{11} \cdot B_{12} - A_{11} \cdot B_{22} \\
&P_2 = S_2 B_{22} \\
&P_3 = S_3 B_{11} \\
&P_4 = A_{22} S_4 \\
&P_5 = S_5 S_6 \\
&P_6 = S_7 S_8 \\
&P_7 = S_9 S_{10}
\end{aligned}$$

Runtime: we recursively call the matrix-matrix multiplication function for 7 matrices of size $n/2 \times n/2$.

Step 4: Combine Using the P matrices, we can show that

$$C_{11} = P_5 + P_4 - P_2 + P_6$$
$$C_{12} = P_1 + P_2$$
$$C_{21} = P_3 + P_4$$
$$C_{22} = P_5 + P_1 - P_3 - P_7$$

9.3 Analysis of Strassen's Method

Question 5. Strassen's algorithm satisfies which recurrence relation for $n > 1$?

A $T(n) = 8T(n/2) + O(n^2)$

B $T(n) = 23T(n/2)$

C $T(n) = 7T(n/2) + O(n^2)$

D $T(n) = 10T(n/2) + O(n^2)$

E $T(n) = 17T(n/2) + O(1)$