CSCI 470: Divide-and-Conquer, Exponentiation, Merge Sort

Vijay Chaudhary

August 28, 2023

Department of Electrical Engineering and Computer Science
Howard University

Overview

1. LaTeX
2. Divide-and-Conquer
3. Exponentiation
4. Merge Sort
5. Classwork

LaTeX

Writing in LaTeX

LaTeX:

- Widely used in academia
- A template will be shared if you are interested in writing in LaTeX

Pros:

- Handy to write mathematical notations
- Homework solution (or anything you write in LaTeX) looks super neat

Cons:

- Can be overwhelming to start with
- Will have to lookup the documentation/cheat sheets a lot in the beginning

Divide-and-Conquer

Divide-and-Conquer

- **Divide** the problem into a number of sub-problems that are smaller instances of the same problem.
- **Conquer** the sub-problems by solving them recursively. If the sub-problem sizes are small enough, however, just solve the sub-problems in a straightforward manner.
- **Combine** the solutions to the sub-problems into the solution for the original problem.

Divide-and-Conquer: An Example

```
SUM(A, n)
if n = 1
    return A[n]
else
    return A[n] + SUM(A, n − 1)
```

Exponentiation

Naive Method

Task: Calculating 2^n.

EXPONENTIATOR(n)

if $ans = 1$
for $i = 1$ **to** n
    $ans = ans * 2$
return ans

Exponentiation

- Can we do it better than $O(n)$?

Exponentiation

- Can we do it better than $O(n)$?
- Can we make use of smaller “sub-products” to get the original one?

Exponentiation: Subproblem

- What can be the subproblem for this?

Exponentiation: Subproblem

- What can be the subproblem for this?
- How do we calculate 2^{16}?

Exponentiation: Subproblem

- What can be the subproblem for this?
- How do we calculate 2^{16}?
- $2^{16} = 2 * 2 * ...2 * 2$; 16 of 2s getting multiplied, basically.

Exponentiation: Subproblem

- What can be the subproblem for this?
- How do we calculate 2^{16}?
- $2^{16} = 2 * 2 * ...2 * 2$; 16 of 2s getting multiplied, basically.
- Sub-problem to 2^{16} can be calculating (2^8) and (2^8), right?

Exponentiation: Subproblem

- What can be the subproblem for this?
- How do we calculate 2^{16}?
- $2^{16} = 2 * 2 * ...2 * 2$; 16 of 2s getting multiplied, basically.
- Sub-problem to 2^{16} can be calculating (2^8) and (2^8), right?
- If I already know 2^8, I can just use this to calculate $(2^8) * (2^8) = 2^{16}$

Exponentiation: Divide-and-Conquer

```
EXPONENTIATOR(n)
if n = 0
    return 1
else if n mod 2 = 0
    x = EXPONENTIATOR(n/2)
    return x * x
else
    x = EXPONENTIATOR((n − 1)/2)
    return 2 * x * x
```

Exponentiation: Proof of Correctness

Base case: When $n = 0$, the algorithm returns 1, which is $2^0 = 1$. Therefore, base case holds. *(This part is missing in the lecture note.)*

Inductive Step: Let's assume that Exponentiator(k) gives us the right output for $k < n$, 2^k. Then, we have to show that Exponentiator(n) gives us the right output, which is 2^n.

Case 1: If $n \bmod 2 = 0$ or n is even, the algorithm returns x^2, where $x = \text{Exponentiator}(n/2)$. By the inductive hypothesis above, $x = 2^{n/2}$. Therefore, $x^2 = (2^{n/2})^2 = 2^n$, which is the expected output.

Case 2: If n is odd, the algorithm returns $2x^2$, where $x = \text{Exponentiator((n-1)/2)}$. Using the inductive hypothesis, for $k = (n-1)/2 < n$, $2 * x^{(n-1)/2} = 2^n$.

Exponentiation: Runtime Analysis

$$T(n) = \begin{cases} c_1 & \text{if } n = 1 \\ T(n/2) + c_2 & \text{if n is even} \\ 2 * T((n-1)/2) + c_3 & \text{if n is odd} \end{cases}$$

Merge Sort

Merge Sort: Divide-and-Conquer

- **Divide**: Divide the n-element sequence to be sorted into two subsequences of $n/2$ elements each.
- **Conquer** Sort the two subsequences recursively using merge sort.
- **Combine** Merge the two sorted subsequences to produce the sorted answer.

Merge Sort: Illustration

1. $\langle 5, 2, 4, 7, 1, 3, 2, 6 \rangle$
2. $\langle 5, 2, 4, 7 \rangle$, $\langle 1, 3, 2, 6 \rangle$
3. $\langle 5, 2 \rangle$, $\langle 4, 7 \rangle$, $\langle 1, 3 \rangle$, $\langle 2, 6 \rangle$
4. $\langle 5 \rangle$, $\langle 2 \rangle$, $\langle 4 \rangle$, $\langle 7 \rangle$, $\langle 1 \rangle$, $\langle 3 \rangle$, $\langle 2 \rangle$, $\langle 6 \rangle$

Merge Sort: Illustration

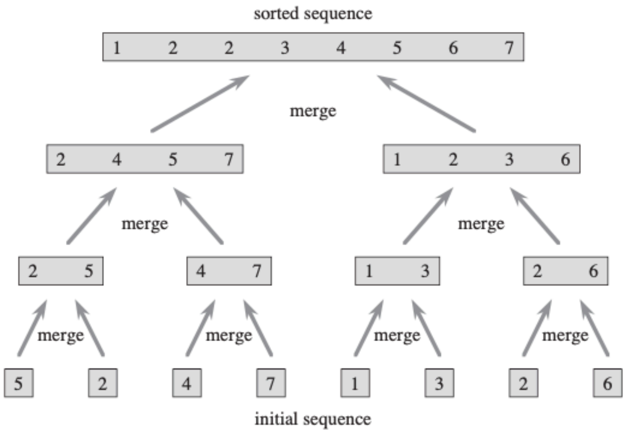

Figure 2.4 The operation of merge sort on the array $A = \langle 5, 2, 4, 7, 1, 3, 2, 6 \rangle$. The lengths of the sorted sequences being merged increase as the algorithm progresses from bottom to top.

Merge Procedure

MERGE(A, p, q, r)

$n_1 = q - p + 1$
$n_2 = r - q$
Let $L[1, ..., n_1 + 1]$ and $R[1, ..., n_2 + 1]$ be new arrays.
for $i = 1$ **to** n_1
    $L[i] = A[p + i - 1]$
for $j = 1$ **to** n_2
    $R[i] = A[q + j]$
$L[n_1 + 1] = \infty$
$R[n_2 + 1] = \infty$
$i = 1$
$j = 1$
for $k = p$ **to** r
    if $L[i] \leq R[j]$
        $A[k] = L[i]$
        $i = i + 1$
    else
        $A[k] = R[j]$
        $j = j + 1$

Merge Procedure

Why the sentinel value, ∞, is useful in the merge procedure?

Merge: Proof of Correctness

Loop invariant: **At the start of each iteration of the for loop of lines 12-18, the subarray $A[p, ...k-1]$ contains the $k-p$ smallest elements of $L[1, ..., n_1+1]$ and $R[1, ..., n_2+1]$ in sorted order.** Moreover, $L[i]$ and $R[j]$ are the smallest elements of their arrays that have not been copied back into A.

Merge: Proof of Correctness

Initialization (Base case): Before the first iteration, we have not copied anything from L and R to A, which shows $A[p, ...k-1]$ is empty, i.e. trivially sorted. We should note that $i = j = 1$, and both $L[i]$, and $R[j]$ are the smallest elements of their arrays.

Maintenance (Inductive step): Let's assume that $A[p, ..., k-1]$ contains the $k-p$ smallest elements of L and R in sorted order.

- Case 1: When $L[i] \leq R[j]$, then $L[i]$ is the smallest element not yet copied back into A. Because $A[p, ..., k-1]$ contains the $k-p$ smallest elements, after line 14 copies $L[i]$ into $A[k]$, the subarray $A[p, ...k]$ will contain the $k-p+1$ smallest elements. With the value of k incremented by the **for** loop, and the value of i to $i+1$ by line 15, reestablishing the loop invariant for the next iteration.
- Case 2: When $L[i] > R[j]$, $R[j]$ is copied to A, which is the next smallest elements to be copied to A such that $A[p, ..., k+1]$ is sorted with $k-p+1$ smallest elements. Loop invariant is maintained in this case as well.

Merge: Proof of Correctness

Termination: The Merge procedure stops, when $k = r + 1$. By the loop invariant, the subarray $A[p, ..k - 1]$, which is $A[p..r]$, contains the $k - p = r - p + 1$ smallest elements of L and R in sorted order. The arrays L and R together contain $n_1 + n_2 + 2 = r - p + 3$ elements. All but the two largest have been copied back into A, and these two largest elements are the sentinels.

Merge-Sort

MERGE-SORT(A, p, r)

if $p < r$

    $q = \lfloor (p + r) \rfloor / 2$

    MERGE-SORT(A, p, q)

    MERGE-SORT$(A, q + 1, r)$

    MERGE(A, p, q, r)

Merge Sort: Proof by Induction

Base Case: When $n = 1$, A is sorted trivially.

Inductive Step: Let's assume that Merge-Sort procedure sorts the subarray of size less than n. The first half of A will be less than n, and so will be the second half. Previously, we showed that Merge procedure sorts two subarrays which are already sorted. If Merge-Sort procedure in line 3 sorts subarray $|A[p...q]| = \lfloor n/2 \rfloor$, and line 4 will sort $|A[q+1, ...r]| = A.length \ - \lfloor n/2 \rfloor$, Merge will sort entire array of size n with the Merge procedure in line 5. This concludes the proof.

Merge-Sort: Runtime analysis

$$T(n) = \begin{cases} \Theta(1) & \text{if } n \leq c \\ aT(n/b) + D(n) + C(n) & \text{otherwise} \end{cases}$$

Merge-Sort: Runtime analysis

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

Classwork

Classwork

1. Using Figure 2.4 as a model, illustrate the operation of merge sort on the array $A = \langle 3, 41, 52, 26, 38, 57, 9, 49 \rangle$.