CSCI 470: Merge Sort (contd), Solving Recurrence Relations

Vijay Chaudhary

September 5, 2023

Department of Electrical Engineering and Computer Science
Howard University

Overview

1. Merge Sort

2. Solving Recurrence Using Substitution

Merge Sort (continued)

Merge Procedure

```
MERGE(A, p, q, r)
n1 = q − p + 1
n2 = r − q
Let L[1, ..., n1 + 1] and R[1, ..., n2 + 1] be new arrays.
for i = 1 to n1
    L[i] = A[p + i − 1]
for j = 1 to n2
    R[i] = A[q + j]
L[n1 + 1] = ∞
R[n2 + 1] = ∞
i = 1
j = 1
for k = p to r
    if L[i] ≤ R[j]
        A[k] = L[i]
        i = i + 1
    else
        A[k] = R[j]
        j = j + 1
```

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.

Initialization (Base case): Before the first iteration, we have not copied anything from L and R to A, which shows $A[p, \ldots k-1]$ 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.

Merge: Proof of Correctness

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

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

Please note that $\Theta(1)$ is to indicate a constant runtime. We can always express sum of constants, $c_1 + c_2 + ... + c_k$ as $\Theta(1)$, where these constants are independent of the input size n.

Merge-Sort: Runtime

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

Solving Recurrence Using Substitution

Substitution Method

1. Guess the form of the solution.
2. Use mathematical induction to find the constants and show that the solution works.

Substitution Method: Example 1

What will be the upper bound of $T(n) = T(n-1) + 1$?

- Let's guess that $T(n) = O(n)$. In order to show this we need to show $T(n) < cn$ for some $c > 0$, and $n \geq n_0$.

Substitution Method: Example 1

What will be the upper bound of $T(n) = T(n-1) + 1$?

- Let's guess that $T(n) = O(n)$. In order to show this we need to show $T(n) < cn$ for some $c > 0$, and $n \geq n_0$.
- We assume that the bound holds for all positive $m > n$, in particular $m = n - 1$, yielding $T(n-1) \leq c(n-1)$.

Substitution Method: Example 1

What will be the upper bound of $T(n) = T(n-1) + 1$?

- Let's guess that $T(n) = O(n)$. In order to show this we need to show $T(n) < cn$ for some $c > 0$, and $n \geq n_0$.
- We assume that the bound holds for all positive $m > n$, in particular $m = n - 1$, yielding $T(n-1) \leq c(n-1)$.
- $$\begin{aligned} T(n) &\leq c(n-1) + c_1 \\ &= cn - c + c_1 \\ &\leq cn \end{aligned}$$

Substitution Method: Example 1

What will be the upper bound of $T(n) = T(n-1) + 1$?

- Let's guess that $T(n) = O(n)$. In order to show this we need to show $T(n) < cn$ for some $c > 0$, and $n \geq n_0$.
- We assume that the bound holds for all positive $m > n$, in particular $m = n - 1$, yielding $T(n-1) \leq c(n-1)$.
- $$\begin{aligned} T(n) &\leq c(n-1) + c_1 \\ &= cn - c + c_1 \\ &\leq cn \end{aligned}$$
- $T(1) = 1$, therefore, for $c \geq 1$, and $n \geq 1$, $T(n) \leq cn$ holds. Therefore, $T(n) = O(n)$.

Substitution Method: Example 2

Let's find the upper bound on the recurrence: $T(n) = 2T(\lfloor n/2 \rfloor) + n$.

- We guess that the solution is $T(n) = O(n \lg n)$.

Substitution Method: Example 2

Let's find the upper bound on the recurrence: $T(n) = 2T(\lfloor n/2 \rfloor) + n$.

- We guess that the solution is $T(n) = O(n \lg n)$.
- The substitution method requires us to prove that $T(n) \leq cn \lg n$ for an appropriate choice of the constant $c > 0$ and $n \geq n_0$.

Substitution Method: Example 2

Let's find the upper bound on the recurrence: $T(n) = 2T(\lfloor n/2 \rfloor) + n$.

- We guess that the solution is $T(n) = O(n \lg n)$.
- The substitution method requires us to prove that $T(n) \leq cn \lg n$ for an appropriate choice of the constant $c > 0$ and $n \geq n_0$.
- We start assuming that this bound holds for all positive $m < n$, in particular for $m = \lfloor n/2 \rfloor$, yielding $T(\lfloor n/2 \rfloor) \leq c\lfloor n/2 \rfloor \lg(\lfloor n/2 \rfloor) + n$.

Example 2 Contd

$$
\begin{aligned}
T(n) &\leq c\lfloor n/2 \rfloor \lg(\lfloor n/2 \rfloor) + n \\
&\leq cn \lg(n/2) + n \\
&= cn \lg n - cn \lg 2 + n \\
&= cn \lg n - cn + n \\
&\leq cn \lg n,
\end{aligned}
$$

where the last step holds as long as $c \geq 1$.

Example 2 contd

However, we also need to *choose* $n \geq n_0$. When $n = 1$, $T(n) \leq cn \lg n = c1 \lg 1 = 0$, which is not true, as $T(1) = 1$. Since we get to choose n_0 that holds $T(n) \leq cn \lg n$ for $c \geq 1$, we can choose $n > 3$. With $T(1) = 1$, we can derive $T(2) = 4$, and $T(3) = 5$.
When $n = 2$, $T(2) \leq c2 \lg 2$, and when $n = 3$, $T(3) \leq c3 \lg 3$. For $c = 3$, $T(2) \leq 3 * 2 \lg 2 = 6$, $T(3) \leq 3 * 3 \lg 3 = 14.26...$. This fixes our base case irregularity. Therefore, we can make $n = 2$ and $n = 3$ as the base cases for this recurrence.

Classwork

Show that $T(n) = T(n-1) + n^2$ is $T(n) = O(n^3)$.

Making a good guess

Unfortunately, there is no general way to make a good guess. However, we can use recursion trees to make a good guess. If a recurrence is similar to one you have seen before, that can be a really good guess.

Possible pitfalls

A wrong guess will give a very wrong answer. For instance, in case of $T(n) = 2T(\lfloor n/2 \rfloor) + n$, if we guess that $T(n) \leq cn$, we will end up with a different asymptotic notation.

$$\begin{aligned} T(n) &\leq 2(c\lfloor n/2 \rfloor) + n \\ &\leq 2 * c * (n/2) + n \\ &= cn + n \\ &= O(n) \Leftarrow \textcolor{red}{\text{wrong!!}} \end{aligned}$$