Homework 2 (50 points) Due: September 20, 2024 11:59 pm

COMPSCI 733: Advanced Algorithms and Designs

Benzon Carlitos Salazar (salazarbc24@uww.edu)

Documentation: (5 points) Type your solutions using Latex (www.overleaf.com or https://www.latex-project.org/). Submit your solutions (pdf is enough) to Canvas.

Problem 1: (15 points) Assume we have a one dimensional array of real numbers with indices, $1, 2, \ldots, n$. The following pseudocodes for MERGE and MERGE-SORT are from CLRS textbook.

Algorithm 1 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 do
    L[i] = A[p + i - 1]
end for
for j = 1 to n_2 do
    R[j] = A[q + j]
end for
L[n_1 + 1] = ∞
R[n_2 + 1] = ∞
i = 1
j = 1
for k = p to r do
    if L[i] ≤ R[j] then
        A[k] = L[i]
        i = i + 1
    else
        A[k] = R[j]
        j = j + 1
    end if
end for
```

Algorithm 2 MERGE-SORT(A,p,r)

```
if p < r then
    q = (p + r)/2
    MERGE-SORT(A, p, q)
    MERGE-SORT(A, q + 1, r)
    MERGE(A, p, q, r)
end if
```

(a) Write pseudocodes with new parameters to modify the above MERGE and MERGE-SORT that divide the array into three equal parts, sort them, and do a three-way merge as follows.

Use a new parameter s and write $MERGE-3(A, p, q, r, s)$ and $MERGE-SORT-3(A, p, s)$, where A is an array and p, q, r, and s are indices into the array such that $p \leq q \leq r < s$. $MERGE$ function assumes that the subarrays $A[p, \ldots, q], A[q+1, \ldots, r]$ and $A[r+1, \ldots, s]$ are in sorted order. Make sure to write the complete pseudo codes for the modified functions.

Algorithm 3 MERGE-3(A,p,q,r,s)

$n_1 = q - p + 1$
$n_2 = r - q$
$n_3 = s - r$
Let $L[1 \ldots n_1 + 1]$, $M[1 \ldots n_2 + 1]$, and $R[1 \ldots n_3 + 1]$ be new arrays
for $i = 1$ to n_1 **do**
    $L[i] = A[p + i - 1]$
end for
for $j = 1$ to n_2 **do**
    $M[j] = A[q + j]$
end for
for $k = 1$ to n_3 **do**
    $R[k] = A[r + k]$
end for
$L[n_1 + 1] = \infty$ ▷ Sentinel values
$M[n_2 + 1] = \infty$
$R[n_3 + 1] = \infty$
$i = 1$
$j = 1$
$k = 1$
for $l = p$ to s **do**
    if $L[i] \leq M[j]$ and $L[i] \leq R[k]$ **then**
        $A[l] = L[i]$
        $i = i + 1$
    else if $M[j] \leq L[i]$ and $M[j] \leq R[k]$ **then**
        $A[l] = M[j]$
        $j = j + 1$
    else
        $A[l] = R[k]$
        $k = k + 1$
    end if
end for

(b) Let $T(n)$ be the running time of MERGE-SORT-3 on an array of size n. Write a recurrence relation (i.e., $T(n) =?$) for your algorithm.

Algorithm 4 MERGE-SORT-3(A,p,s)

if $p < s$ **then**
  $q = p + \left\lfloor \frac{s-p+1}{3} \right\rfloor - 1$ ▷ First third
  $r = q + \left\lfloor \frac{s-p+1}{3} \right\rfloor$ ▷ Second third
  MERGE-SORT-3(A, p, q)
  MERGE-SORT-3$(A, q+1, r)$
  MERGE-SORT-3$(A, r+1, s)$
  MERGE-3(A, p, q, r, s)
end if

Let $T(n)$ be the running time of the MERGE-SORT-3 algorithm. The recurrence relation for $T(n)$ is as follows:

$$T(n) = 3T\left(\frac{n}{3}\right) + cn \quad \text{for}\, n > 1$$

where c is a constant representing the time complexity of the merge step.

The base case occurs when the size of the array is 1:

$$T(1) = O(1)$$

In Problem 2 and 3, you need to complete the details of a proof for the correctness of Merge Sort. You need to refer to the above Algorithm 2, $MERGE\_SORT(A, P, r)$, and Algorithm 1, $MERGE(A, p, q, r)$, given in the CLRS textbook. We will establish correctness of Merge-sort in two parts:

1. Assuming correctness of the Merge procedure, prove the correctness of MergeSort.
2. Prove the correctness of the Merge procedure.

Problem 2: (15 points)

Prove: Assuming that the procedure for Merge is correct, a call to $Merge\_Sort(A, p, r)$, $p \leq r$, returns the elements in $A[p..r]$ rearranged in sorted order, and does not alter any entry outside of the subarray $A[p..r]$.

Proof. : We prove the correctness of the procedure MERGE-SORT by strong induction on $m = r - p + 1$, i.e., by induction on the size of the subarray $A[p..r]$.

Base case: When $p = r$, i.e., $m = 1$, the subarray $A[p..r]$ contains only one element. Since a single element is trivially sorted, MERGE-SORT correctly sorts the subarray.

Induction hypothesis: Assume that MERGE-SORT correctly sorts any subarray $A[p..r]$ of size $1 \leq m < k$. That is, MERGE-SORT correctly sorts any subarray with fewer than k elements.

Induction step: We must prove that MERGE-SORT correctly sorts the subarray $A[p..r]$ when $m = k$.

Consider the procedure MERGE-SORT(A, p, r):

1. If $p < r$, the algorithm computes the midpoint $q = \lfloor (p + r)/2 \rfloor$. This divides the array into two subarrays: $A[p..q]$ and $A[q + 1..r]$.

2. The algorithm recursively calls MERGE-SORT(A, p, q) to sort the left half and MERGE-SORT(A, q+1, r) to sort the right half.

   By the induction hypothesis, MERGE-SORT(A, p, q) correctly sorts the left subarray and MERGE-SORT(A, q+1, r) correctly sorts the right subarray.

3. After the recursive calls, the two subarrays are sorted, and the MERGE(A, p, q, r) procedure is called to merge them into a single sorted subarray $A[p..r]$.

   Since we assume that MERGE is correct, it correctly merges the two sorted subarrays into one sorted array.

   Additionally, MERGE-SORT only modifies the elements in $A[p..r]$ and does not alter any entries outside this subarray.

   Thus, MERGE-SORT correctly sorts $A[p..r]$ and does not modify any elements outside of $A[p..r]$ when $m = k$.

By the principle of strong induction, we conclude that MERGE-SORT correctly sorts any subarray $A[p..r]$ and does not alter any entries outside of this subarray. $\square$

Problem 3: (15 points) Correctness of Merge.

Merge procedure copies subarray $A[p..q]$ into $L[1..n1]$ and $A[q+1..n2]$ into $R[1..n2]$, with $L[n1+1]$ and $R[n2+1]$ set to ∞ (a value larger than any of the elements in $A[p..r]$).

Correctness of Merge is established through the correctness of the following loop invariant for the for loop in Line 14 of the above Algorithm 1:

Loop invariant:

At the start of each iteration of the for loop in line 14, $A[p..k-1]$ contains the $k-p$ smallest elements in $L[1..n1+1]$ and $R[1..n2+1]$ in sorted order. Further, $L[i]$ and $R[j]$ are the smallest elements in their arrays that have not been copied back into A. The elements in array A outside of subarray $A[p..r]$ are unchanged.

Complete the following steps:

1. Show Initialization holds.This establishes the base case by proving that the loop invariant holds just before the start of the first iteration.

2. Show Maintenance holds.Assuming that the loop invariant holds at the start of a given iteration this establishes that it continues to hold at the start of the next iteration.

3. Show Termination holds.States what the loop invariant establishes about the computation at the time when the loop is exited.

   Proof. We prove the correctness of the procedure MERGE by induction on the size of the subarrays being merged.

   Base case: Consider the smallest size for the subarrays, when each subarray contains just one element, i.e., when $q = p$ and $r = q + 1$. In this case, we are merging two single-element subarrays $A[p]$ and $A[q + 1]$.

   Since each subarray contains only one element, they are already sorted. The MERGE procedure compares $A[p]$ and $A[q + 1]$, placing the smaller element in the merged subarray $A[p..r]$.

   Thus, the merged subarray $A[p..r]$ is sorted, and no elements outside this subarray are altered. The base case holds.

   Induction hypothesis: Assume that the MERGE procedure correctly merges any two sorted subarrays of size less than n. That is, for arrays of size $r - p + 1 < n$, MERGE correctly merges the subarrays $A[p..q]$ and $A[q + 1..r]$ into a sorted array $A[p..r]$.

   Induction step: We now prove that MERGE(A, p, q, r) correctly merges the sorted subarrays $A[p..q]$ and $A[q+1..r]$ when $n = r - p + 1$.

   The procedure MERGE creates two auxiliary arrays $L[1..n_1 + 1]$ and $R[1..n_2 + 1]$, where $n_1 = q - p + 1$ and $n_2 = r - q$. The array L stores the elements of the left subarray $A[p..q]$, and R stores the elements of the right subarray $A[q+1..r]$. Sentinel values (infinity) are added to the ends of both L and R to simplify the comparison process.

   The procedure then merges L and R into $A[p..r]$. It uses two pointers, i and j, initialized to 1, to point to the current element in L and R, respectively. A loop runs from $k = p$ to r:

- If $L[i] \leq R[j]$, then $A[k] = L[i]$, and i is incremented.
- Otherwise, $A[k] = R[j]$, and j is incremented.

Since both L and R are sorted, each comparison ensures that the smallest remaining element from either L or R is placed into $A[k]$. After the loop completes, the subarray $A[p..r]$ is fully sorted, and no elements outside the range $A[p..r]$ are altered.

By the induction hypothesis, MERGE correctly merges the sorted subarrays $A[p..q]$ and $A[q + 1..r]$ into a single sorted subarray $A[p..r]$.

Thus, by induction, we conclude that the MERGE procedure correctly merges two sorted subarrays into a single sorted subarray and does not alter any entries outside the specified range. $\square$