Homework 2 (50 points) Due: September 20, 2024 11:59 pm
COMPSCI 733: Advanced Algorithms and Designs

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 the CLRS textbook.

Algorithm 1 MERGE(A,p,q,r)

$n_1 = q - p + 1$
$n_2 = r - q$
let $L[1 \ldots n_1 + 1]$ and $R[1 \ldots n_2 + 1]$ 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
$L[n_1 + 1] = \infty$
$R[n_2 + 1] = \infty$
$i = 1$
$j = 1$
for $k = p$ to r **do**
    if $L[i] \leq R[j]$ **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, sorts 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 $p \le q \le r < s$. $MERGE$ function assumes that the subarrays $A[p,\dots,q], A[q+1,\dots,r]$ and $A[r+1,\dots,s]$ 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)

---

let $n_1 = q-p+1$, $n_2 = r-q$, and $n_3 = s-r$
Let $L[1\dots n_1+1]$, $M[1\dots n_2+1]$, and $R[1\dots n_3+1]$ 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 **do**
    $R[k] = A[r+k]$
end for
Set $L[n_1+1] = \infty$, $M[n_2+1] = \infty$, and $R[n_3+1] = \infty$ (to act as sentinels)
Initialize $i = 1$, $j = 1$, $k = 1$
for $t = p$ to s **do**
    if $L[i] \le M[j]$ and $L[i] \le R[k]$ **then**
        $A[t] = L[i]$
        $i = i+1$
    else if $M[j] \le L[i]$ and $M[j] \le R[k]$ **then**
        $A[t] = M[j]$
        $j = j+1$
    else
        $A[t] = R[k]$
        $k = k+1$
    end if
end for

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

if $p < s$ **then**
    Let $q = p + (s - p)/3$ (first division point)
    Let $r = p + 2 * (s - p)/3$ (second division point)
    MERGE-SORT-3(A, p, q) (recursively sort first third)
    MERGE-SORT-3($A, q + 1, r$)(recursively sort second third)
    MERGE-SORT-3($A, r + 1, s$) (recursively sort third third)
    MERGE-3(A, p, q, r, s)(merge the three sorted parts)
end if

(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.

$$T(n) = \begin{cases} \Theta(1) & \text{if } n \leq 1 \\ 3T\left(\frac{n}{3}\right) + \Theta(n) & \text{if } n > 1 \end{cases}$$

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 $A[p..r]$. By strong induction on $m = r - p + 1$ (i.e., by induction on the size of the sub-array $A[p..r]$).

Proof. : **Base Case**: Need to show correctness for the smallest input size. This occurs when $p = r$, i.e., when $m = 1$. Since the sub-array $A[p..r]$ has only one element, it is trivially sorted.

Inductive Hypothesis: Assume that Merge-Sort(A, p, r) works correctly for all subarrays of size $m \leq k$ for some $k \geq 1$. That is, we assume the algorithm correctly sorts all subarrays of size $m \leq k$ and leaves elements outside of $A[p..r]$ unchanged.

Induction Step: We now need to prove that the algorithm works correctly for a subarray of size $m = k + 1$. Specifically, we need to show that Merge-Sort(A, p, r) correctly sorts the subarray $A[p..r]$ when $m = r - p + 1 = k + 1$

Divide:The Merge-Sort algorithm works by dividing the subarray $A[p..r]$ into two smaller subarrays:

1. First half: A[p..q] where $q = [\frac{p+r}{2}]$.
2. Second half: A[q+1..r].

Conquer: By the inductive hypothesis, since the sizes of these two subarrays are strictly less than $k + 1$, the recursive calls to Merge-Sort(A, p, q) and Merge-Sort$(A, q + 1, r)$ correctly sort both subarrays.

Combine: Since the Merge procedure is assumed to be correct, it will correctly merge the two sorted subarrays $A[p..q]$ and $A[q + 1..r]$ into $A[p..r]$

MERGE SORT is correct for all subarray sizes. $\square$

Problem 3: (15 points) Correctness of Merge. Merge procedure copies subarray $A[p..q]$ into $L[1..n_1]$ and $A[q+1..n_2]$ into $R[1..n_2]$, with $L[n_1+1]$ and $R[n_2+1]$ 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. **Initialization:** At the first iteration of the loop, we have $k = p$, therefore the subarray $A[p..k-1]$ is empty. The empty subaray contains the $(k-p=0)$ smallest elements in L and R. Since $i = j = 1$, both $L[i]$ and $R[j]$ are the smallest elements of their arrays that have not been copied back into A.

2. **Maintenance:** When $L[i] \leq R[j]$, $L[i]$ 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. Incrementing j and i (in line 15) reestablishes the loop invariant for the next iteration. When $L[i] \geq R[j]$, the lines 16–17 perform the appropriate action to maintain the loop invariant.

3. **Termination:** When the loop terminates, $k = r+1$. By the loop invariant, $A[p..r]$ contains the $(r-p)+1$ smallest elements from L and R, all in sorted order. The two remaining elements, $L[n_1+1]$ and $R[n_2+1]$, are the sentinel values set to ∞, which ensures they are never copied into A. Therefore, the array $A[p..r]$ is fully sorted, and the elements outside $A[p..r]$ remain unchanged.