CSC3100 Solution 1

Preface

This reference solution is compiled by CSC3100 Teaching Team in 2022-2023 Fall semester. If you find any mistakes or typos, please contact us by email 120090350@link.cuhk.edu.cn.

Problem 1

2.3-4

> We can express insertion sort as a recursive procedure as follows. In order to sort $A[1..n]$, we recursively sort $A[1..n-1]$ and then insert $A[n]$ into the sorted array $A[1..n-1]$. Write a recurrence for the running time of this recursive version of insertion sort.

It takes $\Theta(n)$ time in the worst case to insert $A[n]$ into the sorted array $A[1..n-1]$. Therefore, the recurrence

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1, \\ T(n-1) + \Theta(n) & \text{if } n > 1. \end{cases}$$

The solution of the recurrence is $\Theta(n^2)$.

Problem 2

2-4

> Let $A[1..n]$ be an array of n distinct numbers. If $i < j$ and $A[i] > A[j]$, then the pair (i, j) is called an ***inversion*** of A.
>
> **a.** List the five inversions in the array $\langle 2, 3, 8, 6, 1 \rangle$.
>
> **b.** What array with elements from the set $\{1, 2, \ldots, n\}$ has the most inversions? How many does it have?
>
> **c.** What is the relationship between the running time of insertion sort and the number of inversions in the input array? Justify your answer.

> **d.** Give an algorithm that determines the number of inversions in any permutation of n elements in $\Theta(n \lg n)$ worst-case time. (*Hint:* Modify merge sort).

a. $(1, 5)$, $(2, 5)$, $(3, 4)$, $(3, 5)$, $(4, 5)$.

b. The array $\langle n, n-1, \dots, 1 \rangle$ has the most inversions $(n-1) + (n-2) + \cdots + 1 = n(n-1)/2$.

c. The running time of insertion sort is a constant times the number of inversions. Let $I(i)$ denote the number of $j < i$ such that $A[j] > A[i]$. Then $\sum_{i=1}^{n} I(i)$ equals the number of inversions in A.

Now consider the **while** loop on lines 5-7 of the insertion sort algorithm. The loop will execute once for each element of A which has index less than j is larger than $A[j]$. Thus, it will execute $I(j)$ times. We reach this **while** loop once for each iteration of the **for** loop, so the number of constant time steps of insertion sort is $\sum_{j=1}^{n} I(j)$ which is exactly the inversion number of A.

d. We'll call our algorithm COUNT-INVERSIONS for modified merge sort. In addition to sorting A, it will also keep track of the number of inversions.

$\text{COUNT-INVERSIONS}(A, p, r)$ sorts $A[p..r]$ and returns the number of inversions in the elements of $A[p..r]$, so $left$ and $right$ track the number of inversions of the form (i, j) where i and j are both in the same half of A.

$\text{MERGE-INVERSIONS}(A, p, q, r)$ returns the number of inversions of the form (i, j) where i is in the first half of the array and j is in the second half. Summing these up gives the total number of inversions in A. The runtime of the modified algorithm is $\Theta(n \lg n)$, which is same as merge sort since we only add an additional constant-time operation to some of the iterations in some of the loops.

```
COUNT-INVERSIONS(A, p, r)
    if p < r
        q = floor((p + r) / 2)
        left = COUNT-INVERSIONS(A, p, q)
        right = COUNT-INVERSIONS(A, q + 1, r)
        inversions = MERGE-INVERSIONS(A, p, q, r) + left + right
        return inversions
```

```
MERGE-INVERSIONS(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[j] = A[q + j]
    L[n1 + 1] = ∞
    R[n2 + 1] = ∞
    i = 1
```

```
    j = 1
    inversions = 0
    for k = p to r
        if L[i] <= R[j]
            A[k] = L[i]
            i = i + 1
        else
            inversions = inversions + n1 - i + 1
            A[k] = R[j]
            j = j + 1
    return inversions
```

Problem 3

3.1-5

> Prove Theorem 3.1.

The theorem states:

> For any two functions $f(n)$ and $g(n)$, we have $f(n) = \Theta(g(n))$ if and only if $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$.

From $f = \Theta(g(n))$, we have that

$0 \le c_1 g(n) \le f(n) \le c_2 g(n)$ for $n > n_0$.

We can pick the constants from here and use them in the definitions of O and Ω to show that both hold.

From $f(n) = \Omega(g(n))$ and $f(n) = O(g(n))$, we have that

$$\begin{aligned} & 0 \le c_3 g(n) \le f(n) & \text{for all } n \ge n_1 \\ \text{and } & 0 \le f(n) \le c_4 g(n) & \text{for all } n \ge n_2. \end{aligned}$$

If we let $n_3 = \max(n_1, n_2)$ and merge the inequalities, we get

$0 \le c_3 g(n) \le f(n) \le c_4 g(n)$ for all $n > n_3$.

That is the definition of Θ.

Problem 4-5

3-4.b-c

> Let $f(n)$ and $g(n)$ by asymptotically positive functions. Prove or disprove each of the following conjectures.
>
> **b.** $f(n) + g(n) = \Theta(\min(f(n), g(n)))$.
>
> **c.** $f(n) = O(g(n))$ implies $\lg(f(n)) = O(\lg(g(n)))$, where $\lg(g(n)) \ge 1$ and $f(n) \ge 1$ for all sufficiently large n.

b. Disprove, $n^2 + n \ne \Theta(\min(n^2, n)) = \Theta(n)$.

c. Prove, because $f(n) \ge 1$ after a certain $n \ge n_0$.

$$\begin{aligned} \exists c, n_0 : \forall n \ge n_0, 0 \le f(n) \le cg(n) \\ \Rightarrow 0 \le \lg f(n) \le \lg(cg(n)) = \lg c + \lg g(n). \end{aligned}$$

We need to prove that

$\lg f(n) \le d \lg g(n)$.

We can find d,

$d = \frac{\lg c + \lg g(n)}{\lg g(n)} = \frac{\lg c}{\lg g(n)} + 1 \le \lg c + 1$,

where the last step is valid, because $\lg g(n) \ge 1$.

Problem 6

4.3-3

> We saw that the solution of $T(n) = 2T(\lfloor n/2 \rfloor) + n$ is $O(n \lg n)$. Show that the solution of this recurrence is also $\Omega(n \lg n)$. Conclude that the solution is $\Theta(n \lg n)$.

First, we guess $T(n) \le cn \lg n$,

$$\begin{aligned} T(n) & \le 2c\lfloor n/2 \rfloor \lg \lfloor n/2 \rfloor + n \\ & \le cn \lg(n/2) + n \\ & = cn \lg n - cn \lg 2 + n \\ & = cn \lg n + (1 - c)n \\ & \le cn \lg n, \end{aligned}$$

where the last step holds for $c \ge 1$.

Next, we guess $T(n) \ge c(n + a) \lg(n + a)$,

$$
\begin{aligned}
T(n) &\ge 2c(\lfloor n/2 \rfloor + a)(\lg(\lfloor n/2 \rfloor + a) + n \\
&\ge 2c((n-1)/2 + a)(\lg((n-1)/2 + a)) + n \\
&= 2c\frac{n-1+2a}{2}\lg\frac{n-1+2a}{2} + n \\
&= c(n-1+2a)\lg(n-1+2a) - c(n-1+2a)\lg 2 + n \\
&= c(n-1+2a)\lg(n-1+2a) + (1-c)n - (2a-1)c & (0 \le c < 1, n \ge \frac{(2a-1)c}{1-c}) \\
&\ge c(n-1+2a)\lg(n-1+2a) & (a \ge 1) \\
&\ge c(n+a)\lg(n+a).
\end{aligned}
$$

Problem 7

4.5-3

> Use the master method to show that the solution to the binary-search recurrence $T(n) = T(n/2) + \Theta(1)$ is $T(n) = \Theta(\lg n)$. (See exercise 2.3-5 for a description of binary search.)

$$
\begin{aligned}
a &= 1, b = 2, \\
f(n) &= \Theta(n^{\lg 1}) = \Theta(1), \\
T(n) &= \Theta(\lg n).
\end{aligned}
$$

Problem 8

4-1

> Give asymptotic upper and lower bound for $T(n)$ in each of the following recurrences. Assume that $T(n)$ is constant for $n \le 2$. Make your bounds as tight as possible, and justify your answers.
>
> **a.** $T(n) = 2T(n/2) + n^4$.
>
> **b.** $T(n) = T(7n/10) + n$.
>
> **c.** $T(n) = 16T(n/4) + n^2$.
>
> **d.** $T(n) = 7T(n/3) + n^2$.
>
> **e.** $T(n) = 7T(n/2) + n^2$.
>
> **f.** $T(n) = 2T(n/4) + \sqrt{n}$.
>
> **g.** $T(n) = T(n-2) + n^2$.

a. By master theorem, $T(n) = \Theta(n^4)$.

b. By master theorem, $T(n) = \Theta(n)$.

c. By master theorem, $T(n) = \Theta(n^2 \lg n)$.

d. By master theorem, $T(n) = \Theta(n^2)$.

e. By master theorem, $T(n) = \Theta(n^{\lg 7})$.

f. By master theorem, $T(n) = \Theta(\sqrt{n} \lg n)$.

g. Let $d = m \mod 2$,

$$
\begin{aligned}
T(n) &= \sum_{j=1}^{j=n/2} (2j+d)^2 \\
&= \sum_{j=1}^{n/2} 4j^2 + 4jd + d^2 \\
&= \frac{n(n+2)(n+1)}{6} + \frac{n(n+2)d}{2} + \frac{d^2 n}{2} \\
&= \Theta(n^3).
\end{aligned}
$$