Instructor Name: Siddhartha Nath
Email: s.nath@northeastern.edu

Northeastern University

Program Structure and Algorithms (INFO 6205)
Homework #2 – 100 points

Student NAME: Xijing Zhang

Student ID: 002205911

Notes:

- Please submit two files.
- The first file MUST be a PDF that contains your solutions to all questions except the coding question.
- The second file is your solution to the coding question with either .py or .cpp or .java extension.

Question 1 (20 points)**.** *Solve the following recurrence relations using the Master method and give a Θ bound for each of them. Please clearly indicate values of a, b and d.*

(a) $T(n) = 2T(n/3) + 1$

$a = 2$

$b = 3$

$f(n) = 1 = Theta(n^0)$

Calculate:

$$\log_3 2 \approx 0.631$$

Since $f(n) = O(n^{\log_b a-\epsilon})$ *for some* $\epsilon > 0$*, we apply Case 1 of the Master Theorem:*

$$T(n) = \Theta(n^{\log_b a}) = \Theta(n^{0.631})$$

(b) $T(n) = 5T(n/4) + n$

$a = 5$

$b = 4$

$f(n) = n = \Theta(n^1)$

Calculate:

$$\log_4 5 \approx 1.161$$

Since $f(n) = O(n^{\log_b a - \epsilon})$ *for some* $\epsilon > 0$*, we apply Case 1 of the Master Theorem:*

$$T(n) = \Theta(n^{\log_b a}) = \Theta(n^{1.161})$$

(c) $T(n) = 9T(n/3) + n^2$

- $a = 9$
- $b = 3$
- $f(n) = n^2 = \Theta(n^2)$

Calculate:

$$\log_3 9 = 2$$

Since $f(n) = \Theta(n^{\log_b a})$*, we apply Case 2 of the Master Theorem, where* $f(n) = \Theta(n^{\log_b a} \log^k n)$ *with* $k = 0$*:*

$$T(n) = \Theta(n^2 \log n)$$

(d) $T(n) = 8T(n/2) + n^3$

- $a = 8$
- $b = 2$
- $f(n) = n^3 = \Theta(n^3)$

Calculate:

$$\log_2 8 = 3$$

Since $f(n) = \Theta(n^{\log_b a})$*, we apply Case 2 of the Master Theorem, where* $f(n) = \Theta(n^{\log_b a} \log^k n)$ *with* $k = 0$*:*

$$T(n) = \Theta(n^3 \log n)$$

(e) $T(n) = 49T(n/25) + n^{3/2} \log n$

- $a = 49$
- $b = 25$
- $f(n) = n^{3/2} \log n = \Theta(n^{3/2} \log n)$

Calculate:

$$\log_{25} 49 \approx 1.18$$

Comparing $f(n)$ *with* $n^{\log_b a}$*:*

- $f(n) = \Theta(n^{3/2} \log n)$ - $n^{\log_b a} = n^{1.18}$

Since $f(n) = \Omega(n^{\log_b a + \epsilon})$ *for some* $\epsilon > 0$*, and the regularity condition holds, we apply Case 3 of the Master Theorem:*

$$T(n) = \Theta(f(n)) = \Theta(n^{3/2} \log n)$$

Question 2 (25 points)**.** *Consider the recurrence, $T(n) = 2T(n/2) + cn^2$. Please use a recursion tree to answer the following questions.*

(a) (5 points) What is the height (or, depth) of the tree?

(b) (5 points) What is the total cost at any depth i, that is not the leaf-level?

(c) (5 points) How many leaves does the tree have? What is the total cost at the leaf-level?

(d) (10 points) Derive a guess for an asymptotic upper bound (i.e, $O(\cdot)$) for $T(n)$.

(a) Each recursive step reduces the problem size from n to $n/2$, meaning the depth of the recursion tree is determined by how many times we can divide n by 2 until we reach the base case ($T(1)$).

Since the size reduces as:

$$n, \frac{n}{2}, \frac{n}{4}, \ldots, \frac{n}{2^i}$$

The recursion stops when $n/2^i = 1$, solving for i:

$$2^i = n$$

$$i = \log_2 n$$

Thus, the ***height (or depth) of the recursion tree is*** *$\log_2 n$.*

(b) At depth i, there are 2^i subproblems, each of size $n/2^i$. The cost at each node is:

$$c(n/2^i)^2 = c\frac{n^2}{4^i}$$

Since there are 2^i nodes at level i, the total cost at depth i is:

$$2^i \cdot c\frac{n^2}{4^i} = cn^2\frac{2^i}{4^i} = cn^2\left(\frac{1}{2}\right)^i$$

(c) The number of leaves corresponds to the number of subproblems at the deepest level. Since the tree has height $\log_2 n$, and the number of nodes doubles at each level, the number of leaves is:

$$2^{\log_2 n} = n$$

At each leaf, the remaining problem size is 1. Assuming a constant cost per leaf, the total cost at the leaf level is proportional to the number of leaves:

$$O(n)$$

(d) The total cost at each depth i is:

$$cn^2\left(\frac{1}{2}\right)^i$$

Summing over all levels from $i = 0$ to $\log_2 n$:

$$T(n) = \sum_{i=0}^{\log_2 n} cn^2\left(\frac{1}{2}\right)^i$$

This forms a geometric series with sum:

$$T(n) = cn^2 \sum_{i=0}^{\log_2 n} \left(\frac{1}{2}\right)^i$$

The sum of a finite geometric series is:

$$S = \frac{1 - r^{m+1}}{1 - r}$$

where $r = \frac{1}{2}$ *and* $m = \log_2 n$*, so:*

$$S = \frac{1 - (1/2)^{\log_2 n+1}}{1 - 1/2} = 2\left(1 - (1/2)^{\log_2 n+1}\right)$$

Since $(1/2)^{\log_2 n} = 1/n$*, for large* n*:*

$$S \approx 2$$

Thus:

$$T(n) = cn^2 \cdot 2 = O(n^2)$$

So the asymptotic upper bound is $O(n^2)$*.*

Question 3 (25 points)**.** *Consider sorting* n *numbers stored in array* $A[1:n]$ *by first finding the smallest element of* $A[1:n]$ *and exchanging it with the element in* $A[1]$*. Then find the smallest element of* $A[2:n]$*, and exchange it with* $A[2]$*. Then find the smallest element of* $A[3:n]$*, and exchange it with* $A[3]$*. Continue in this manner for the first* $n-1$ *elements of* A*. Write pseudocode for this algorithm, which is known as* **Selection Sort***. Give the worst-case running time of selection sort in* Θ*-notation.*

Pseudocode for Selection Sort

```
SELECTION-SORT(A, n)
for i = 1 to n - 1 do
    min_index = i
    for j = i + 1 to n do
        if A[j] < A[min_index] then
            min_index = j
    Swap A[i] and A[min_index]
```

Step-by-Step Explanation of the Algorithm
- Iterate through the array from index $i = 1$ to $n - 1$.
- Find the smallest element in the unsorted portion of the array $A[i:n]$.
- Swap the smallest element with $A[i]$.
- Repeat the process for all elements except the last one.

Worst-Case Running Time Analysis
- Outer loop runs $n - 1$ times.
- Inner loop runs $(n - i)$ times in the worst case.
- Total comparisons:

$$(n-1) + (n-2) + (n-3) + ... + 2 + 1 = \sum_{i=1}^{n-1} i = \frac{(n-1)n}{2}$$

- At most $n-1$ swaps are performed.
Thus, the worst-case time complexity is:

$$\Theta(n^2)$$

Question 4 (30 points). *You are given an array of distinct integers $A[1:n]$. A was sorted in increasing order but has been right rotated (i.e, the last element is cyclically shifted to the starting position of the array) k times. You task is to find the minimum value of k by designing an efficient divide-and-conquer algorithm.*

Suppose, $A = \{15, 18, 2, 3, 6, 12\}$, then original it would have been $\{2, 3, 6, 12, 15, 18\}$ and rotated $k = 2$ times.
(a) *Please describe your algorithm in English.*

(b) *Please write code for your algorithm in (a) in either Python / Java / C++. To receive full credit, please structure your code, write comments and show the output for the above two examples.*

(a) We use a modified binary search to find the index of the minimum element in $O(\log n)$ time.

1. **Initialize** `low = 0` and `high = n - 1`.
2. **Check if the array is already sorted**: If `A[low] <= A[high]`, return `low` since no rotation has happened.
3. **Find the mid element**: `mid = (low + high) / 2`.
4. **Check if mid is the pivot**:
   - If `A[mid] > A[mid + 1]`, then `A[mid + 1]` is the minimum (pivot).
   - If `A[mid - 1] > A[mid]`, then `A[mid]` is the minimum (pivot).
5. **Move towards the unsorted part**:
   - If `A[mid] >= A[low]`, the pivot must be in the right half → `low = mid + 1`.
   - Otherwise, search in the left half → `high = mid - 1`.
6. **Continue until the pivot is found**.