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

Northeastern University

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

Student NAME:

Student ID:

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

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

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

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

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

(a) $a = 2, b = 3, d = 0; d < log_3 2 \Rightarrow Case 1,$ *so* $T(n) = \Theta(n^{log_3 2}$.

(b) $a = 5, b = 4, d = 1; d < log_4 5 \Rightarrow Case 1,$ *so* $T(n) = \Theta(n^{log_4 5}$.

(c) $a = 9, b = 3, d = 2; d = log_3 9 \Rightarrow$ *Case 2 with* $k = 0,$ *so* $T(n) = \Theta(n^2 \log n)$.

(d) $a = 8, b = 2, d = 3; d = \log_2 8 \Rightarrow$ *Case 2 with* $k = 0,$ *so* $T(n) = \Theta(n^3 \log n)$.

(e) $a = 49, b = 25, d = 3/2; d > \log_{25} 49 \Rightarrow,$ *Case 3, so* $T(n) = \Theta(n^{3/2} \log n)$.

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) Size of each node at level i is $n/2^i$. At the last level the size will be 1, so $n/2^i = 1 \Rightarrow i = log_2 n =$ height of the tree.

(b) At level i there are 2^i nodes each of size $n/2^i$. The cost is $c(n/2^i)^2$ per node, so the total cost is $2^i \cdot cn^2/4^i = cn^2/2^i$.

(c) At $i = log_2 n$ we have $2^{log_2 n} = n$ leaf nodes. Total cost at leaf-level is $n \cdot c(1)^2 = cn$

(d) $T(n) = \sum_0^{log_2 n}(1/2)^i \cdot cn^2 + cn < \sum_0^{\infty}(1/2)^i \cdot cn^2 + cn = 1/(1-(1/2))cn^2 + cn = 2cn^2 + cn < O(n^2)$.

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.*

*The pseudocode for the **Selection Sort** algorithm is below.*

Selection-Sort(A, n)

```
for i = 1 to (n − 1) do
    smallest ← i
    for j = 1 + 1 to n do
        if A[j] < A[smallest] then
            smallest ← j
        end if
    end for
    swap(A[i], A[smallest])
end for
```

The running time of the algorithm is $\Theta(n^2)$ for all cases.

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.*

Main observation: The smallest element is the only element whose previous is greater than itself. If there is no previous element, then there is no rotation (i.e., the first element is the smallest). Using this, our algorithm will be as follows.

(i) Check the above condition for the middle element by comparing it with the $(mid-1)$ and $(mid+1)$ elements.

(ii) If the smallest element is not at the middle (neither mid nor $mid+1$),

(iii) If the middle element is smaller than the last element, then the smallest element lies in the left half

(iv) Else it lies in the right half.

(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.*