1. We consider the Horner's method for evaluating the polynomial $a_n x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0$ at $x = c$.

```
1 procedure Horner(c, a_0, a_1, ..., a_n)
2 y ← a_n
3 for i = 1, 2, ..., n
4     y ← y * c + a_{n-i}
```

   (a) (10 points) Evaluate $3x^2 + 2x + 1$ at $x = 3$ by working through each step of the algorithm showing the values assigned at each assignment step.

   Solution:

   1. $y \leftarrow 3$
   2. $y \leftarrow 3 \times 3 + 2 = 11$
   3. $y \leftarrow 11 \times 3 + 1 = 34$

   (b) (10 points) Exactly how many multiplications and additions are used by this algorithm to evalue a polynomial of degree n at $x = c$? (Do not count additions used to increment the loop variable.)

   Solution: At line 4, we do 1 multiplication and 1 addition. The loop at line 3 goes from 1 to n. Therefore, we do n multiplications and n additions.

2. Use the Master Theorem to solve the following recurrence relations.

   (a) (5 points) $T(n) = 8T(n/2) + n$

   Solution: $a = 8$, $b = 2$, and $f(n) = n$. $\log_b a = 3$, $f(n) = O(n^{3-\epsilon})$, where $\epsilon = 2$, and thus $T(n) = \Theta(n^3)$.

   (b) (5 points) $T(n) = 8T(n/2) + n^3$

   Solution: $f(n) = \Theta(n^3)$, which suggests $T(n) = \Theta(n^3 \log n)$.

   (c) (5 points) $T(n) = 3T(n/2) + n$

   Solution: $a = 3$, $b = 2$, and $f(n) = n$. $\log_b a = \log_2 3 \in (1, 2)$, $f(n) = O(n^{3-\epsilon})$, and thus $T(n) = \Theta(n^{\log_2 3})$.

   (d) (5 points) $T(n) = T(n/4) + 1$

Solution: $a = 1$, $b = 4$, $f(n) = 1$. $\log_b a = 0$, $f(n) = \Theta(n^0)$, and thus $T(n) = \Theta(\log n)$.

(e) (5 points) $T(n) = 3T(n/3) + n^2$

Solution: $a = 3$, $b = 3$, and $f(n) = n^2$. $\log_b a = 1$, $f(n) = O(n^{1+\epsilon})$. Furthermore, $3f(n/3) \leq cf(n)$ is true for $c \in (1/3, 1)$ and all values of n, then $T(n) = \Theta(f(n)) = \Theta(n^2)$.

3. (15 points) Consider the input sequence $A =< 5, 3, 17, 10, 84, 19, 6, 22, 9 >$. Illustrate the operations of build-heap algorithm on A by specifying the content of A after each step of the algorithm.

Solution: heap-size $\leftarrow 9$

1. $i = 4$
   (a) call heapify$(A, 4)$, $A =< 5, 3, 17, \underline{22}, 84, 19, 6, \underline{10}, 9 >$.
   (b) call heapify$(A, 8)$, A doesn't change.
2. $i = 3$
   (a) call heapify$(A, 3)$, $A =< 5, 3, \underline{84}, 22, \underline{17}, 19, 6, 10, 9 >$.
   (b) call heapify$(A, 5)$, A doesn't change.
3. $i = 2$
   (a) call heapify$(A, 2)$, $A =< 5, \underline{22}, 84, \underline{3}, 17, 19, 6, 10, 9 >$.
   (b) call heapify$(A, 4)$, $A =< 5, 22, 84, \underline{10}, 17, 19, 6, \underline{3}, 9 >$.
   (c) call heapify$(A, 8)$, A doesn't change.
4. $i = 1$
   (a) call heapify$(A, 1)$, $A =< \underline{84}, 22, \underline{5}, 10, 17, 19, 6, 3, 9 >$.
   (b) call heapify$(A, 3)$, $A =< 84, 22, \underline{19}, 10, 17, \underline{5}, 6, 3, 9 >$.
   (c) heapify$(A, 6)$, A doesn't change.

4. (15 points) Write an efficient *heapify* algorithm that uses iterative control instead of recursion.

Solution: Let n be the heap-size of A.

```
heapify(A, i, n)
while (2i ≤ n)
    largest ← i
    if A[2i] > A[i]
        largest ← 2i
    if 2i + 1 ≤ n & A[2i + 1] > A[largest]
        largest ← 2i + 1
    if largest ≠ i
        exchange A[i] ↔ A[largest]
        i ← largest
    else
        i ← n + 1
```

5. (25 points) Instead of a complete sort, the binning problem assigns each element of an input sequence A into bins with specified range. For instance, let $A = \{0.4, 0.7, 0.1, 0.24, 0.3, 0.6, 0.9\}$, and there are four bins as shown below, the binning process will assign $0.1, 0.24$ to the first bin, $0.4, 0.3$ to the second bin, $0.7, 0.6$ to the third bin, and 0.9 to the forth bin.

0.1,0.24	0.4,0.3	0.7,0.6	0.9
$[0, 0.25)$	$[0.25, 0.5)$	$[0.5, 0.75)$	$[0.75, 1]$

Suppose that we are given an input sequence A of n elements $a_1, a_2, \ldots, a_n$ within the range $[0, 1]$ and we need to do the binning process using the above four bins. Write an algorithm with linear complexity that does the work.

[*Hint: A decimal number within* $[0, 1]$ *is presented as a* 0–1 *string on the computer. For instance,* $0.25 = (0.010000000\ldots)_2$. *If you just look at the substring* $(01)_2$, *what is the result in base 10?*]

Solution: We need three auxiliary arrays $B[1 : n]$, $C[1 : n]$, and $D[1 : 4]$.

```
for i = 1, 2, ..., n
    B[i] ← max(ceil(A[i] * 4), 1)
for i = 1, 2, 3, 4
    D[i] ← 0
```

```
for i = 1, 2, ..., n
    D[B[i]] ← D[B[i]] + 1
for i = 2, 3, 4
    D[i] ← D[i] + D[i − 1]
for i = n, n − 1, ..., 1
    C[D[B[i]]] ← A[i]
    D[B[i]] ← D[B[i]] − 1
```

6. **(Bonus)** Consider the adaptive version of the binning problem. In this case, each bin can have no more than s elements, where s is a prescribed parameter. Should that happen, the bin is partitioned into two equal parts. For instance, assume that $s = 3$, and the bin $[0.25, 0.5)$ have four elements 0.27,0.33,0.35, and 0.49, the adaptive binning process will partition it into two bins $[0.25, 0.375)$ with 0.27, 0.33, and 0.35 and $[0.375, 0.5)$ with 0.49. Design an algorithm that does the work.

   Note: the problem is worth 25 points if your result has $O(n \log n)$ complexity, and is worth 50 points if the complexity is $O(n)$. Please submit the solution to this problem separately.