Homework Assignment 1

1. **Exercise 2.2−2**
   The loop invariant is that for any iteration i, the smallest $i-1$ elements will be sorted in ascending order. Thus, we only need to run the algorithm on the first $n-1$ elements because the smallest $n-1$ elements will be sorted at that point. In other words, the nth remaining number must be the greatest in our array. In both best and worst cases, the running time of the algorithm is $\theta(n^2)$.

Algorithm 1 Selection Sort Pseudocode

Input: Unsorted Array $\mathbf{x}$
Output: Sorted Ascending Order Array $\mathbf{x}$

```
1: function SelectionSort(x)
2:     for i ← 0 to length(x)−1 do
3:         smallest ← x[i]
4:         for j ← i + 1 to length(x) do
5:             if x[j] < smallest then
6:                 smallest ← x[j]
7:             end if
8:         end for
9:         x[j] ← x[i]
10:        x[i] ← smallest
11:    end for
12: end function
```

2. **Exercise 2.3−3**
   Base case: For $n = 2$, we have $nlogn = 2\log 2 = 2 \cdot 1 = 2$. Therefore, $T(n)$ is true for $n = 2$.

   Inductive Step: Assume that $T(n/2)$ holds for $T(n/2) = (n/2)\log(n/2)$ then, we must show that this holds for $T(n)$.

$$\begin{aligned} T(n) &= 2T(n/2) + n \\ &= 2(n/2)\log(n/2) + n \\ &= n(\log(n) - 1) + n \\ &= n\log(n) \end{aligned}$$

   Therefore, $T(n)$ is true for all $n \geq 2$ which are exact powers of 2.

3. **Problem 2−3**
   The following code fragment implements Horner's rule for evaluating a polynomial

$$\begin{aligned} P(x) &= \sum_{k=0}^{n} a_k x^k \\ &= a_0 + x\left(a_1 + x\left(a_2 + \cdots + x\left(a_{n-1} + xa_n\right)\cdots\right)\right) \end{aligned}$$

given the coefficients $a_0, a_1, \ldots, a_n$ and a value for x :

$$y = 0$$
$$\text{for } i = n \text{ downto } 0$$
$$y = a_i + x \cdot y$$

a. The running time of this code fragment is $\Theta(n)$.
b. Write pseudocode to implement the naive polynomial-evaluation algorithm that computes each term of the polynomial from scratch. What is the running time of this algorithm? How does it compare to Horner's rule?
c. Consider the following loop invariant: At the start of each iteration of the for loop of lines $2-3$,

$$y = \sum_{k=0}^{n-(i+1)} a_{k+i+1} x^k$$

Interpret a summation with no terms as equaling 0. Following the structure of the loop invariant proof presented in this chapter, use this loop invariant to show that, at termination, $y = \sum_{k=0}^{n} a_k x^k$.
d. Conclude by arguing that the given code fragment correctly evaluates a polynomial characterized by the coefficients $a_0, a_1, \ldots, a_n$.

4. **Prove or disprove** $f(n) + g(n) = \Theta(\textbf{max}(f(n), g(n)))$
With the assumption that $f(n)$ and $g(n)$ are non-negative functions, $max(f(n), g(n))$ produces either $f(n)$ or $g(n)$ for all $n \geq 0$. This means that $f(n)+g(n) \leq max(f(n), g(n))$ for all $n \geq 0$. Therefore, $f(n) + g(n) = \Theta(\text{max}(f(n), g(n)))$.

5. **Problem 3.3a**
The expressions can be ranked into the following order:

6. **Exercise 4.1−5**
Use the following ideas to develop a nonrecursive, linear-time algorithm for the maximum-subarray problem. Start at the left end of the array, and progress toward the right, keeping track of the maximum subarray seen so far. Knowing a maximum subarray of $A[1 \ldots j]$, extend the answer to find a maximum subarray ending at index $j+1$ by using the following observation: a maximum subarray of $A[1 \ldots j+1]$ is either a maximum subarray of $A[1 \ldots j]$ or a subarray $A[i \ldots j+1]$, for some $1 \leq i \leq j+1$. Determine a maximum subarray of the form $A[i.j+1]$ in constant time based on knowing a maximum subarray ending at index j.

Algorithm 2 Maximum Subarray Problem

```
1: function MaxSubArray(x)
       return ()
2: end function
```

7. **Exercise 4.2−4**
What is the largest k such that if you can multiply 3×3 matrices using k multiplications (not assuming commutativity of multiplication), then you can multiply $n \times n$ matrices in time $o\left(n^{\lg 7}\right)$? What would the running time of this algorithm be?

8. **Exercise 4.3−7**
   Using the master method in Section 4.5, you can show that the solution to the recurrence $T(n) = 4T(n/3) + n$ is $T(n) = \Theta\left(n^{\log_3 4}\right)$. Show that a substitution proof with the assumption $T(n) \leq cn^{\log_3 4}$ fails. Then show how to subtract off a lower-order term to make a substitution proof work.

9. **Exercise 4.4−9** Use a recursion tree to give an asymptotically tight solution to the recurrence $T(n) = T(\alpha n) + T((1-\alpha)n) + cn$, where α is a constant in the range $0 < \alpha < 1$ and $c > 0$ is also a constant.

10. **Problem 4.3bfhj**