Due: 1 pm on Friday, March 1

Assignment 1

1 Homework Guidelines

- Welcome to Algorithm Design and Analysis (Honor Track)!
- According to the unified arrangement of the course (with regular track), the release and deadline of each assignment are on Monday and Friday of the week, respectively. Late assignments will NOT be accepted.
- You are encouraged to discuss assignment problems with your peers, with the TAs, and with the course instructor. However, your solutions should be based on your own understanding and should be written **independently**. For each assignment, if you discussed the problems with other students in the class, you should include a list of the students' names.
- You can write your solutions in either English or Chinese with equal treatment. Just take it easy :)

2 Textbook Exercises

1.18, 1.19(1)(3)(5)(7), 1.21

3 Complexity Bounds

For each blank, indicate whether Ai is in O, Ω, or Θ of Bi. More than one space per row can be valid.

A	B	O	Ω	Θ
$10n$	n	✓	✓	✓
10	n	✓		
n^2	$2n$			
n^{2021}	2^n			
$n^{\log 9}$	$9^{\log n}$			
$\log(n!)$	$\log(n^n)$			
$(3/2)^n$	$(2/3)^n$			
3^n	2^n			
$n^{1/\log n}$	1			
$\log^5 n$	$n^{0.5}$			
n^2	$4^{\log n}$			
$n^{0.2}$	$(0.2)^n$			
$\log\log n$	$\sqrt{\log n}$			
$\log(\sqrt{n})$	$\sqrt{\log n}$			

4 Gauging Complexity from Code

Refer to the algorithms below for solving these problems.

Give the worst-case run-time complexity in Big-Oh notation for each of the algorithms above.

Algorithm 1: Search an element in an array of length n

```
Input: An array A of size n and an element e
Output: Index of element e
i ← 0
while i < n do
    if A[i] = e then
        return i
    i ← i + 1
return −1                                   // If element e is not found
```

Algorithm 2: Replace an element in an array of length n with a given element

```
Input: An array A of size n, an element e present in the array and the new element E
Output: Updated array
i ← 0
while i < n do
    currElement ← A[i]
    if currElement = e then
        A[i] ← E
        return A
    else
        A[i] ← currElement
    i ← i + 1
return A        // If element e is not found, return the original array
```

Algorithm 3: Sort an array of length n

```
Input: An array A of size n
Output: Sorted array
for i ∈ [0, n − 1) do
    /* Declare i as the index having the current smallest element   */
    smallest ← i
    for j ∈ [i + 1, n) do
        /* Find the smallest element in the range [i, n)            */
        if A[j] ≤ A[smallest] then
            smallest ← j
    swap(A[i], A[smallest]) // Swap numbers present at index i and smallest
return A
```

5 Partial Sum of a 1D Array

Given an array `A` consisting of `n` integers `A[1],A[2],...,A[n]`. You want to obtain a 2D $n \times n$ array `B` where `B[i][j]` (for `i<j`) contains the sum of array entries `A[i]` through `A[j]`, i.e., `A[i]+A[i+1]+...+A[j]`. The value of array entry `B[i][j]` is left unspecified whenever `i`$\geqslant$`j`.

Here's a basic algorithm to solve this problem:

```
For i = 1, 2, ..., n
  For j = i + 1, i + 2, ..., n
    Add up array entries A[i] through A[j]
    Store the result in B[i, j]
  Endfor
Endfor
```

(a) For some function f you may choose, give a bound of $O(f(n))$ on the runtime of this algorithm.

(b) For this same function f, prove that the runtime of the algorithm is also $\Omega(f(n))$ (This shows an asymptotically tight bound of $\Theta(f(n))$ on the runtime).

(c) Although the algorithm you analysis in (a)(b) is the most intuitive way to solve the problem, after all, it contains some highly unnecessary sources of inefficiency. Give a different algorithm to solve the problem, with an asymptotically better runtime. In other words, you should develop an algorithm with runtime $O(g(n))$, where $\lim_{n\to\infty} g(n)/f(n) = 0$.