Before you start:

Homework Files

You can download the starter files for coding as well as this *tex* file (you only need to modify *homework1.tex*) on canvas and do your homework with latex. Or you can scan your handwriting, convert to pdf file, and upload it to canvas before the due date. If you choose to write down your answers by hand, you can directly download the pdf file on canvas which provides more blank space for solution box.

Submission Form

For homework 1, there are two parts of submission:

1. A pdf file as your solution named as VE281_HW1_[Your Student ID]_[Your name].pdf uploaded to canvas
2. Code for 3.2 uploaded to joj (there will be hidden cases but the time restriction is similar to the pretest cases).

For the programming question(question 3.2),you must make sure that your code compiles successfully on a Linux operating system with g++ and the options:

```
-std=c++1z -Wconversion -Wall -Werror -Wextra -pedantic
```

Estimated time used for this homework: **4-5 hours.**

Great credits to 2021FA and 2021SU VE281 TA Group!

0 Student Info (0 point)

Your name and student id:

Solution: Gong Zimu 520370910037

1 Complexity Analysis (20 points, after Lec2)

(a) Based on the given code, answer the following questions:(4 points)

```
void question_1a(int n) {
  int count = 0;
  for (int i = 0; i <= n; i++) {
    for (int j = i; j > 0; j--) {
      count += j;
      }
  }
  cout << count << endl;
}
```

i) What is the output? Describe the answer with variable n. (2 points)

ii) What is the time complexity of the following function? (2 points)

Solution:

$$count = \begin{cases} 0, n \leq 0, \\ \frac{1}{6}n(n+1)(n+2), n \geq 1 \end{cases}$$

$O(n^2)$

(b) What is the time complexity of the following function? (4 points)

```
void question_1b(int N, int M, int K) {
  int count = 0;
  for (int i = 0; i < N; i += 2) {
    for (int j = 0; j < M / 2; j++) {
      count++;
    }
  }
  for (int i = 0; i < K; i++) {
    count--;
  }
}
```

Solution: $O(\frac{1}{4}MN + K)$

(c) What is the time complexity of the following function? Select **All** the answers that are correct, and state your reason. (4 points)

```
void question_1c(int n) {
  int count = 0;
  int m = static_cast<int>(floor(sqrt(n)));
  for (int i = n/2; i < n; i++) {
    for (int j = 1; j < n; j = 2*j) {
      for (int k = 0; k < n; k += m) {
        count++;
      }
    }
  }
}
```

A) $\Theta(n^{1/2}\log n)$

B) $\Theta(n\log n)$

C) $O(n\log n)$

D) $\Theta(n^{3/2}\log n)$

E) $\Theta(n^2\log n)$

F) $O(n^{5/2}\log n)$

G) $\Theta(n^{5/2}\log n)$

Solution: D F
$T(n) = \frac{n}{2}\log n\sqrt{n}$
It is obvious that we pick D.
It is O to any function growing faster, so we pick F.

(d) What is the time complexity of the following function (using Θ)? Show your steps. (4 points)

```
int love281(int n) {
  if (n <= 3) return 1;
  return (love281(n - 1) + love281(n - 2) + love281(n - 3);
}
```

Solution:
$T(n) = max\{T(n-1), T(n-2), T(n-3)\} + O(1)$ when $n > 3$
$T(n) = T(n-1) + O(1) = T(n-2) + 2O(1) = ... = nO(1) = O(n)$ when n is large

(e) Consider the following four statements regarding algorithm complexities:

i) an algorithm with a $\Theta(n\log n)$ time complexity will always run faster than an algorithm with a $\Theta(n^2)$ time complexity.

ii) an algorithm with a constant time complexity will always run faster than an algorithm with a $\Theta(\log n)$ time complexity.

iii) an algorithm with a $\Theta(n!)$ time complexity will always run faster than an algorithm with a $\Theta(n^2)$ time complexity.

iv) an algorithm with a $\Theta(n^2)$ time complexity will always run faster than an algorithm with a $\Theta(n!)$ time complexity.

Which of these statements is/are true? Show your reasons. (4 points)

Solution:
i) False. An algorithm with $T_1(n) = 1000n \log n$ may run slower than an algorithm with $T_2(n) = 0.001n^2$ if n is small.
ii) False. An algorithm with $T_1(n) = 1000$ may run slower than an algorithm with $T_2(n) = \log n$ if n is small.
iii) False. An algorithm with $T_1(n) = n!$ is likely to be slower than an algorithm with $T_2(n) = n^2$ if n is large.
iv) False. An algorithm with $T_1(n) = 1000n^2$ may run slower than an algorithm with $T_2(n) = 0.001n!$ if n is small.

2 Master Theorem (15 points, after Lec3)

2.1 Recurrence Relation (9 points)

What is the complexity of the following recurrence relation? (if not mentioned, please state it with big-theta notation.)

(a) $T(n) = \begin{cases} c_0, & n = 1 \\ 6T\left(\frac{n}{3}\right) + (3n+2)(4n+1), & n > 1 \end{cases}$

Solution:
$T(n) = 6T(\frac{n}{3}) + O(n^2), a = 6, b = 3, d = 2, a < b^d$
Thus, $T(n) = O(n^2)$

(b) $T(n) = \begin{cases} c_0, & n = 1 \\ 3T\left(\frac{n}{9}\right) + 2\log n + 3\sqrt{n} + c, & n > 1 \end{cases}$

Solution:
$T(n) = 3T(\frac{n}{9}) + O(\sqrt{n}), a = 3, b = 9, d = \frac{1}{2}, a = b^d$
Thus, $T(n) = O(\sqrt{n} \log n)$

(c) $T(n) = \begin{cases} c_0, & n = 1 \\ 3T(\frac{n}{4}) + n\log n + c, & n > 1 \end{cases}$

Solution:
$T(n) = 3T(\frac{n}{4}) + O(n^2), a = 3, b = 4, d = 2, a < b^d$
Thus, $T(n) = O(n \log n)$

2.2 Master Theorem on code (6 points)

Based on the function below, answer the following question. **Assume that** $cake(n)$ **runs in** $\log n$ **time.**

```
void pie(int n) {
  if (n == 1) {
    return;
  }
  pie(n / 4);
  int cookie = n * n;
  for (int i = 0; i < cookie; ++i) {
    for (int j = 0; j < n; ++j) {
      cake(n);
    }
  }
  for (int k = 0; k < cookie; ++k) {
    pie(n / 5);
  }
  cake(cookie * cookie);
}
```

Calculate the recurrence relation of this function.

Solution:
$T(n) = T(\frac{n}{4}) + n^3 \log n + T(\frac{n}{5})n^2 + 4 \log n + O(1)$

3 Sorting Algorithms (45 points, after Lec4)

3.1 Sorting Basics (9 points)

In this part, you only need to write down your choice. No explanation is required.

3.1.1 Sorting algorithms' working scenarios (3 points)

What is the most efficient sorting algorithm for each of the following situations?

(a) Given a small array of integers.

A) insertion sort
B) selection sort
C) quick sort
D) bucket sort

Solution: C

(b) Given a large array of integers that is already almost sorted.

A) insertion sort
B) selection sort
C) quick sort
D) bucket sort

Solution: A

(c) Given a large collection of integers that are drawn from a very small range.

A) insertion sort
B) selection sort
C) quick sort
D) counting sort

Solution: D

3.1.2 Sorting snapshots (6 points)

(a) Suppose you had the following unsorted array:

$$\{22, 9, 13, 52, 66, 74, 28, 59, 71, 35, 11, 47\}$$

A snapshot is taken during execution of a sorting algorithm. If the snapshot of the array is:

$$\{9, 13, 22, 52, 66, 74, 28, 59, 71, 11, 35, 47\}$$

which of the following sorts is currently being run on this array?

A) bubble sort
B) insertion sort
C) selection sort
D) quick sort
E) merge sort
F) none of above

Solution: E

(b) Suppose you had the following unsorted array:

$$\{22, 9, 13, 52, 66, 74, 28, 59, 71, 35, 11, 47\}$$

A snapshot is taken during execution of a sorting algorithm. If the snapshot of the array is:

$$\{9, 11, 13, 22, 28, 74, 66, 59, 71, 35, 52, 47\}$$

which of the following sorts is currently being run on this array?

A) bubble sort
B) insertion sort
C) selection sort
D) quick sort
E) merge sort
F) none of above

Solution: C

(c) Suppose you had the following unsorted array:

$$\{22, 9, 13, 52, 66, 74, 28, 59, 71, 35, 11, 47\}$$

A snapshot is taken during execution of a sorting algorithm. If the snapshot of the array is:

$$\{22, 9, 13, 11, 35, 28, 47, 59, 71, 66, 52, 74\}$$

which of the following sorts is currently being run on this array?

A) bubble sort
B) insertion sort
C) selection sort
D) quick sort
E) merge sort
F) none of above

Solution: D

3.2 Squares of a Sorted Array (17 points)

You are given an integer array A sorted in **non-decreasing order (non-positive numbers included)**, and you are required to return an array of the **squares** of each number sorted in non-decreasing order. It is guaranteed that there is **always one zero**(0) in the given array. Here is the sample code:

```cpp
#include <iostream>
using namespace std;

// REQUIRES: an array A and its size n
// EFFECTS: sort array A
// MODIFIES: array A
void insertion_sort(int *A, size_t size) {
    for (size_t i = 1; i < size; i++) {
        size_t j = 0;
        while (j < i && A[i] >= A[j]) {
            j++; // Find the location to insert the value
        }
        int tmp = A[i]; // Store the value we need to insert
        for (size_t k = i; k > j; k--) {
            A[k] = A[k-1];
        }
        A[j] = tmp;
    }
}

int main(){
    int A[5] = {-4, -1, 0, 3, 10};
    size_t sizeA = 5;
    for (size_t i = 0; i < sizeA; i++) { //Calculate the square of input array
        A[i] = A[i] * A[i];
    }
    insertion_sort(A, sizeA);
    for (auto item: A) {
        cout << item << ' ';
    }
    cout << endl;
    return 0;
}
```

However, the code runs quite slowly when it encounters array with great length. It is found that some operations maybe useless in his code because of the **special property of the input array**. Hence, you are required to find out how the code can be improved to have $\Theta(n)$ **time complexity**. **You should follow the provided starter file.**

In the starter file, we have finished the IO part for you, and here is the IO rule:
Input format: an integer array A sorted in non-decreasing order with always one zero (non-positive numbers included)
Output format: an array of the squares of each number sorted in non-decreasing order

Here are some demos:

- **Sample 1**
  Input: `-1 0 1`
  Output: `0 1 1`

- **Sample 2**
  Input: `-4 -2 0 1 3`
  Output: `0 1 4 9 16`

*You can find the starter code in the given starter file *square.cpp*. Please finish the `square_sort()` function in *square.cpp* file, and upload the file to joj(DO NOT CHANGE THE FILE NAME!!).
*Since the cases on joj are quite small, they are just used for you to briefly check the correctness of your code. We will manually check the time complexity of your code.

3.3 Wiggle Sort (6 + 6 points)

If we are given an unordered array arr, we would like to reorder it **in-place** such that $arr[0] \leq arr[1] \geq arr[2] \leq arr[3] \ldots$ Write the **pseudocode** clearly.
The intuitive idea you can think of is to first sort the original array, and then manipulate some swapping to achieve required reordering. The average time complexity is $\Theta(n \log n)$.

Solution:

Algorithm 1 WiggleSort(arr[.])
Input: an array arr of n elements
Output: the reordering array of arr

```
QuickSort(arr[.]);
for (int i = 0; i < size(arr[.])-1; i+=2) {
    int tmp = arr[i];
    arr[i] = arr[i+1];
    arr[i+1] = tmp;
}

```

Another more efficient way is to think of the property of each position. For example, if it is of odd index, then it should be larger or equal to the left element. The method's time complexity is $\Theta(n)$.

Solution:

Algorithm 2 WiggleSort(arr[.])
Input: an array arr of n elements
Output: the reordering array of arr

```
for (int i = 0; i < size(arr[.]); i+=2) {
    if (arr[i] < arr[i-1]) {
        int tmp = arr[i];
        arr[i]=arr[i-1];
        arr[i-1]=tmp;
    }
    else {
        arr[i]=arr[n-i];
        arr[n-i-1]=tmp;
    }
}
```

3.4 Quicker sort simulation (7 points)

To get a better understanding of the mechanism of sorting algorithm, please simulate the given array for each iteration of required algorithm.

3.4.1 Quick sort (3 points)

Assume that we always choose the **first entry** as the pivot to do the partition, and we want to sort the array in **ascending order**. Then, for the following array:

$$A = \{6, 2, 8, 10, 3, 1, 9\}$$

Please give the solution in the following format

Solution:

Iter	Current Subarray	Pivot	Swapped Subarray	Current Array
1	$\{6, 2, 8, 10, 3, 1, 9\}$	6	$\{3, 2, 1, 6, 10, 8, 9\}$	$\{3, 2, 1, 6, 10, 8, 9\}$
2	$\{3, 2, 1\}$	3	$\{1, 2, 3\}$	$\{1, 2, 3, 6, 10, 8, 9\}$
3	$\{1, 2\}$	1	$\{1, 2\}$	$\{1, 2, 3, 6, 10, 8, 9\}$
4	$\{\}$	None	$\{\}$	$\{1, 2, 3, 6, 10, 8, 9\}$
5	$\{2\}$	None	$\{2\}$	$\{1, 2, 3, 6, 10, 8, 9\}$
6	$\{\}$	None	$\{\}$	$\{1, 2, 3, 6, 10, 8, 9\}$
7	$\{10, 8, 9\}$	10	$\{9, 8, 10\}$	$\{1, 2, 3, 6, 9, 8, 10\}$
8	$\{9, 8\}$	9	$\{8, 9\}$	$\{1, 2, 3, 6, 8, 9, 10\}$
9	$\{8\}$	None	$\{8\}$	$\{1, 2, 3, 6, 8, 9, 10\}$
10	$\{\}$	None	$\{\}$	$\{1, 2, 3, 6, 8, 9, 10\}$
11	$\{\}$	None	$\{\}$	$\{1, 2, 3, 6, 8, 9, 10\}$

*Brief explanation: You need to strictly follow the algorithm we learned in class as the following (since there are so many different kinds of quick sort with slight changes).

```
void quicksort(int *a, int left, int right) {
  int pivotat; // index of the pivot
  if(left >= right) return;
  pivotat = partition(a, left, right);
  quicksort(a, left, pivotat-1);
  quicksort(a, pivotat+1, right);
}
```

The steps above strictly follows the recursion order. For example, for iter 4, this is the left half part of iter 3, while iter 5 is the right half part of iter 3.

Now please simulate quick sort for the following array:

$$A = \{6, 3, 15, 2, 7, 11, 8, 1, 10, 9, 4, 13\}$$

You should follow the format above.

Solution:

Iter	Current Subarray	Pivot	Swapped Subarray	Current Array
1	$\{6,3,15,2,7,11,8,1,10,9,4,13\}$	6	$\{1,3,4,2,6,11,8,7,10,9,15,13\}$	$\{1,3,4,2,6,11,8,7,10,9,15,13\}$
2	$\{1,3,4,2\}$	1	$\{1,3,4,2\}$	$\{1,3,4,2,6,11,8,7,10,9,15,13\}$
3	$\{\}$	None	$\{\}$	$\{1,3,4,2,6,11,8,7,10,9,15,13\}$
4	$\{3,4,2\}$	3	$\{2,3,4\}$	$\{1,2,3,4,6,11,8,7,10,9,15,13\}$
5	$\{2\}$	None	$\{2\}$	$\{1,2,3,4,6,11,8,7,10,9,15,13\}$
6	$\{4\}$	None	$\{4\}$	$\{1,2,3,4,6,11,8,7,10,9,15,13\}$
7	$\{11,8,7,10,9,15,13\}$	11	$\{9,8,7,10,11,15,13\}$	$\{1,2,3,4,6,9,8,7,10,11,15,13\}$
8	$\{9,8,7,10\}$	9	$\{7,8,9,10\}$	$\{1,2,3,4,6,7,8,9,10,11,15,13\}$
9	$\{7,8\}$	7	$\{7,8\}$	$\{1,2,3,4,6,7,8,9,10,11,15,13\}$
10	$\{\}$	None	$\{\}$	$\{1,2,3,4,6,7,8,9,10,11,15,13\}$
11	$\{8\}$	None	$\{8\}$	$\{1,2,3,4,6,7,8,9,10,11,15,13\}$
12	$\{10\}$	None	$\{10\}$	$\{1,2,3,4,6,7,8,9,10,11,15,13\}$
13	$\{15,13\}$	15	$\{13,15\}$	$\{1,2,3,4,6,7,8,9,10,11,13,15\}$
14	$\{13\}$	None	$\{13\}$	$\{1,2,3,4,6,7,8,9,10,11,13,15\}$
15	$\{\}$	None	$\{\}$	$\{1,2,3,4,6,7,8,9,10,11,13,15\}$

3.4.2 Merge Sort (4 points)

For the following array:

$$A = \{6,2,8,10,3,1,7\}$$

Part of the answer is shown for applying merge sort to the given array ($mid = (left + right)/2$):

Solution:

1. Division: $\{6,2,8,10\}$ $\{3,1,7\}$
2. Division: $\{6,2\}$ $\{8,10\}$ $\{3,1,7\}$
3. Division: $\{6\}$ $\{2\}$ $\{8,10\}$ $\{3,1,7\}$
4. Merge: $\{2,6\}$ $\{8,10\}$ $\{3,1,7\}$
5. Division / Merge: ...

   ...

   Last. Merge: $\{1,2,3,6,7,8,10\}$

Now please simulate merge sort for the following array:

$$A = \{6,3,15,2,7,11,8,1,10,9,4,13\}$$

Please show all the details of each division or merge.

Solution:

1. Division: $\{6,3,15,2,7,11\}$ $\{8,1,10,9,4,13\}$
2. Division: $\{6,3,15\}$ $\{2,7,11\}$ $\{8,1,10,9,4,13\}$

3. Division: $\{6,3\}$ $\{15\}$ $\{2,7,11\}$ $\{8,1,10,9,4,13\}$
4. Division: $\{6\}$ $\{3\}$ $\{15\}$ $\{2,7,11\}$ $\{8,1,10,9,4,13\}$
5. Merge: $\{3,6\}$ $\{15\}$ $\{2,7,11\}$ $\{8,1,10,9,4,13\}$
6. Merge: $\{3,6,15\}$ $\{2,7,11\}$ $\{8,1,10,9,4,13\}$
7. Division: $\{3,6,15\}$ $\{2,7\}$ $\{11\}$ $\{8,1,10,9,4,13\}$
8. Division: $\{3,6,15\}$ $\{2\}$ $\{7\}$ $\{11\}$ $\{8,1,10,9,4,13\}$
9. Merge: $\{3,6,15\}$ $\{2,7\}$ $\{11\}$ $\{8,1,10,9,4,13\}$
10. Merge: $\{3,6,15\}$ $\{2,7,11\}$ $\{8,1,10,9,4,13\}$
11. Merge: $\{2,3,6,7,11,15\}$ $\{8,1,10,9,4,13\}$
12. Division: $\{2,3,6,7,11,15\}$ $\{8,1,10\}$ $\{9,4,13\}$
13. Division: $\{2,3,6,7,11,15\}$ $\{8,1\}$ $\{10\}$ $\{9,4,13\}$
14. Division: $\{2,3,6,7,11,15\}$ $\{8\}$ $\{1\}$ $\{10\}$ $\{9,4,13\}$
15. Merge: $\{2,3,6,7,11,15\}$ $\{1,8\}$ $\{10\}$ $\{9,4,13\}$
16. Merge: $\{2,3,6,7,11,15\}$ $\{1,8,10\}$ $\{9,4,13\}$
17. Division $\{2,3,6,7,11,15\}$ $\{1,8,10\}$ $\{9,4\}$ $\{13\}$
18. Division $\{2,3,6,7,11,15\}$ $\{1,8,10\}$ $\{9\}$ $\{4\}$ $\{13\}$
19. Merge: $\{2,3,6,7,11,15\}$ $\{1,8,10\}$ $\{4,9\}$ $\{13\}$
20. Merge: $\{2,3,6,7,11,15\}$ $\{1,8,10\}$ $\{4,9,13\}$
21. Merge: $\{2,3,6,7,11,15\}$ $\{1,4,8,9,10,13\}$
22. Merge: $\{1,2,3,4,6,7,8,9,10,11,13,15\}$

4 Selection Algorithm (20 points, after Lec5)

(a) Tim is an undergraduate student taking VE281. He is interested in **random** selection algorithm and wondering which scenario will be the **worst-case** one.

- For the following input sequence, can you give out the pivot sequence for the worst-case scenario? Suppose that we are finding the 3-rd biggest element. (6 points)
  Input sequence:
  $$23, 7, 30, 24, 12, 17, 9$$
  Solution format for the pivot sequence:
  (Certainly the pivot selection is not correct for the requirement :) It is only for clarifying the format.)

Solution:
Pick 23: 12, 7, 9, 17, 23, 24, 30
...

Write your answer below:

Solution:
Pick 7: 7, 23, 30, 24, 12, 17, 9
Pick 9: 7, 9, 30, 24, 12, 17, 23
Pick 12: 7, 9, 12, 24, 30, 17, 23
Pick 30: 7, 9, 12, 24, 23, 17, 30
Pick 17: 7, 9, 12, 17, 23, 24, 30
Pick 24: 7, 9, 12, 17, 23, 24, 30
Pick 23: 7, 9, 12, 17, 23, 24, 30

- What is the time complexity of the worst-case scenario? (2 points)

Solution: $O(n^2)$

(b) William is a master of algorithm. He thinks that Tim's selection algorithm is too naive and prefers **deterministic** selection algorithm. However, since his favorite number is 7, he **partitions the numbers by groups of 7 rather than 5**. He is confident that such a change will make no difference to the time complexity of the algorithm. Is him right? Namely, will the time complexity of William's new selection algorithm **still be** $O(n)$, where n is the amount of input numbers? Give your proof. (8 points)

Solution:
At least $\frac{4}{7} \times \frac{1}{2} = \frac{2}{7}$ of the elements smaller than $x_{k/2}$, also the same amount of elements larger, resulting in shrinking the range to $\frac{5}{7}$
Claim: suppose there exists a positive constant c such that

(a) $T(1) \leq c$

(b) $T(n) \leq cn + T(\frac{n}{7}) + T(\frac{5}{7}n)$

Then $T(n) \leq 14cn$ Proof by induction:

(a) Base case: $T(1) \leq 14c$

(b) Inductive case: inductive hypothesis $T(k) \leq 14ck$, $\forall k < n$. Then
$T(n) \leq cn + T(\frac{n}{7}) + T(\frac{5}{7}n) \leq cn + 7cn + 5cn < 14cn$

Thus, the time complexity of new selection algorithm is still $O(n)$.

(c) When we consider the time complexity of an algorithm, we usually ignore the constant coefficients since we care about large n. Similarly, if we only consider the large cases, which algorithm do you think may work better, deterministic or random? State your reason in detail. (4 points) (Hint: think about William's change and what professor said in the lecture before you have your answer.)

Solution:
Random is better.
Deterministic method may generate a worse constant compared to random due to the nature of recurrence.
Deterministic method is not in-place since it requires n/7 additional space to store the median array, which also takes more time to access the memory in practice.