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 2020FA VE281 TA Group and enormous thanks to 2021SU VE281 TA Roihn!

0 Student Info (0 point)

Your name and student id:

Solution: 陶思郡 Sijun Tao 519021910906

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 += 1;
    }
   }
  cout << count << endl;
}
```

i) What is the output? Describe the answer with variable n. (1 points)

ii) What is the time complexity of the following function? What do you find when comparing it with your answer of the previous part i)? (3 points)

Solution:

(i) Output = $1+2+\cdots+(n-1)=\frac{n^2}{2}-\frac{n}{2}$

(ii) iteration times = $\frac{n^2}{2}-\frac{n}{2}=O(n^2)$

Therefore, the time complexity of the function is $O(n^2)$

We find that the answer in (i) equals to $O(n^2)$.

(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:

iteration times = $\frac{N}{2} \cdot \frac{M}{2} + k = \frac{1}{4}MN + k$

time complexity = $O(MN+k)$

(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++;
      }
    }
  }
}
```

i) $\Theta(n^{1/2} \log n)$
ii) $\Theta(n \log n)$
iii) $O(n \log n)$
iv) $\Theta(n^{3/2} \log n)$
v) $\Theta(n^2 \log n)$
vi) $O(n^{5/2} \log n)$
vii) $\Theta(n^{5/2} \log n)$

Solution:

iteration times = $\frac{n}{2} \cdot \log n \cdot \text{ceil}(\sqrt{n})$

thus, the whole iteration times $k \geq \frac{1}{2} n^{\frac{3}{2}} \cdot \log n$

time complexity = $\Omega(n^{\frac{3}{2}} \cdot \log n)$
= $\Theta(n^{\frac{3}{2}} \cdot \log n)$
= $O(n^{\frac{5}{2}} \cdot \log n)$

Therefore, iv) and vi) are correct.

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

```
int unknown_function(int n) {
  if (n <= 1) return 1;
  return n * (unknown_function(n-1));
}
```

Solution:

the time complexity for one recursion is $O(1)$.
Since we call the recursion for $n+1$ times,
the total iteration times is $O(1)\cdot(n+1) = O(1)\cdot O(n) = O(n)$
Hence, the time complexity is $O(n)$.

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

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

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

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

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

How many of these statements are true? Show your reasons. (4 points)

Solution:

Statement (iii) is true.

(i) No. $\Theta(n^2)$ is larger, so in most cases it will run slower.

(ii) No. In some special cases, the algorithm with $\Theta(n^2)$ may run faster.

(iii) Yes.

(iv) No. $\Theta(n!)$ is really large, so it will run slower.

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) $T(n) = \begin{cases} c_0, & n = 1 \\ 4T\left(\frac{n}{2}\right) + 16n + n^2 + c, & n > 1 \end{cases}$

Solution:

$a = 4,\ b = 2,\ d = 2$

Since $a = b^d$, we know that

$$T(n) = \Theta(n^d \log n) = \Theta(n^2 \log n)$$

(b) $T(n) = \begin{cases} c_0, & n = 1 \\ 5T\left(\frac{n}{25}\right) + \sqrt{n} + c, & n > 1 \end{cases}$

Solution:

$a = 5,\ b = 25,\ d = \frac{1}{2}$

Since $a = b^d$, we know that

$$T(n) = \Theta(n^d \log n) = \Theta(\sqrt{n} \cdot \log n)$$

(c) $T(n) = \begin{cases} c_0, & n = 1 \\ 3T(n-1) + c, & n > 1 \end{cases}$ (Hint: Can you still use master theorem here?)

Solution:

$$T(n) = 3T(n-1) + c = 3[3T(n-2) + c] + c = \cdots = 3^{n-1}T(1) + (3^{n-2} + 3^{n-1} + \cdots + 3^0)c$$
$$= 3^{n-1}c_0 + \frac{1}{2} \cdot 3^{n-1} \cdot c - \frac{1}{2}c$$

$$T(n) = 3^n\left(\frac{1}{3}c_0 + \frac{1}{6}c\right) - \frac{1}{2}c \leq 3^n\left(\frac{1}{3}c_0 + \frac{1}{6}c\right) = O(3^n)$$

$$T(n) = O(3^n)$$

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 / 7);
  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 < n; ++k) {
    pie(n / 3);
  }
  cake(cookie * cookie);
}
```

Calculate the recurrence relation of this function.

Solution:

Let $T(n)$ be the time needed to run pie(n)

Recurrence relation:

$$T(n) = T(\tfrac{n}{7}) + n^2 \cdot n \cdot \log n + n \cdot T(\tfrac{n}{3}) + 4\log n$$

$$T(n) = T(\tfrac{n}{7}) + n \cdot T(\tfrac{n}{3}) + (n^3+4)\log n$$

$$T(n) = T(\tfrac{n}{7}) + n \cdot T(\tfrac{n}{3}) + O(n^3 \log n)$$

$$\Rightarrow T(n) = \begin{cases} C_0, & n = 1 \\ T(\tfrac{n}{7}) + n \cdot T(\tfrac{n}{3}) + (n^3+4)\log n + C, & n > 1 \end{cases}$$

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) A small array of integers.

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

Solution: A

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

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

Solution: B

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

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

3.2 Squares of a Sorted Array (17 points)

Sxt is now doing his homework1 for VE281. He is given an integer array A sorted in **non-decreasing order (non-positive numbers included)**, and is 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 Sxt's code:

```
#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 << ' ';
    }
```

```
31      cout << endl;
32      return 0;
33  }
```

However, after discussing with Jhf, he finds his code runs slowly when it encounters array with great length. Also, he guesses some operations maybe useless in his code because of the **special property of the input array**. Hence, he hopes that you can help him find out where he can improve his code to have $O(n)$ **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`

1. square
2. find index of '0'
3. two pointers. fill in the new array.

merge

*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 Tim Sort (12 points)

Suppose that We want to find the 6^{th} largest element, which is 6, in the following array, Insertion sort is a simple and fast algorithm when the length of array n is short. However, when n goes large, insertion sort may not be the best choice, as the worst case time complexity is $O(n^2)$. We can speed up insertion sort by combining it with merge in mergeSort we learnt in the lectures.

Algorithm 1 timSort(a[.],x)

Input: an array a of n elements, and integer x $>$ 0 (you can assume that $x \ll n$)
Output: the sorted array of a

```
for i = 0; i < n; i += x do
    insertionSort(a,i,min(i+x-1, n-1));
end for
for step = x; step < n; step *= 2 do
    for left = 0; left < n; left += 2 * step do
        mid = left + step - 1;
        right = min(left + 2*step - 1, n - 1);
        merge(a, left, mid, right);
    end for
end for
```

The algorithm is used as the default sorting algorithm in Java and Python. Herem we assume that $x \ll n$ is a known constant.

(a) Suppose n = 1000 and x = 32. How many times will insertionSort be performed?

(b) Suppose x = 32. Express in terms of n how many comparisons in the worst case will be performed in insetionSort.

(c) Express the worst case running time of the whole algorithm in terms of big-Oh notation.

(d) Is this algorithm in-place? If not, express the additional space needed in terms of the big-Oh notation.

Solution:

(a) Suppose perform m times. then $32m \geq 1000$ and $31m < 10$

$m \in [31.25, 32.26) \Rightarrow m = 32$

insertionSort will be performed 32 times.

(b) In this case, insertionSort will be performed $\frac{n}{32}$ times.

for the first $\frac{n}{32}-1$ times, the sort length will be $X = 32$,

for the last time, the sort length will be $n - i = n - (\frac{n}{32}-1)32$

total times $= \frac{31}{2} \cdot 32 \cdot \lfloor \frac{n}{32} \rfloor + \frac{(n-32\lfloor \frac{n}{32} \rfloor - 1)(n-32\lfloor \frac{n}{32} \rfloor)}{2} \approx \frac{31}{2} n$

(C) worst case $= \frac{n}{X} \cdot (\frac{X^2}{2} - \frac{X}{2}) + \frac{n}{2X} \cdot 2X\log 2X + \frac{n}{4X} \cdot 4X \log 4X + \cdots + \frac{n}{\frac{n}{2}} \cdot \frac{n}{2} \log \frac{n}{2}$

$= n \cdot (\frac{X}{2} - 1) + n(\log 2X + \log 4X + \cdots + \log n - \log 2)$

$= O(n \log n)$

(d) No. Since in merge sort, it needs an additional array, it is not in-place. When left = 0, right = n-1, it needs space of n.

additional space = $O(n)$

3.4 Quicker sort simulation (7 points)

To fully understand 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:

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

Sxt shares his answer for this array:

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:

$$A = \{6, 2, 8, 5, 11, 10, 4, 1, 9, 7, 3\}$$

You should follow the format that Sxt has shared.

Solution:

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

3.4.2 Merge Sort (8 points)

For the following array:

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

Sxt shares part of his answer for applying merge sort to the given array:

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:

$$A = \{6, 2, 8, 5, 11, 10, 4, 1, 9, 7, 3\}$$

Please show all the details of each division or merge.

Solution:

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

Last. Merge: {1,2,3,4,5,6,7,8,9,10,11}

4 Selection Algorithm (20 points, after Lec5)

(a) Coned 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:
  $$37, 7, 31, 27, 22, 13, 46$$
  Solution format for the pivot sequence: stable movement
  (Certainly the pivot selection is not correct for the requirement :) It is only for clarifying the format.)

  Solution:
  Pick 22: 7,13,22,37,31,27,46
  Pick 46: 7,13,22,37,31,27,46
  Pick 37: 7,13,22,31,27,37,46
  ...

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

(b) William is a master of algorithm. He thinks that Coned'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)

(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:
(a) (i) Pick 7 : 7, 37, 31, 27, 22, 13, 46
Pick 13 : 7, 13, 37, 31, 27, 22, 46
Pick 22 : 7, 13, 22, 37, 31, 27, 46
Pick 46 : 7, 13, 22, 37, 31, 27, 46
Pick 27 : 7, 13, 22, 27, 37, 31, 46
Pick 37 : 7, 13, 22, 27, 31, 37, 46

(ii) the time complexity for the worst-case scenario is $O(n^2)$

Solution:

(b) Yes. time complexity will still be $O(n)$.

Proof: if partition the number by 7, then $\begin{cases} T(1) \le c \\ T(n) \le cn + T(\frac{n}{7}) + T(\frac{5}{7}n) \end{cases}$

Suppose $T(n) \le 7cn$. then

for $T(1)$, $T(1) \le 7c$

induction, $T(n) \le cn + cn + 5cn = 7cn$.

Hence $T(n) \le 7cn = O(n) \Rightarrow T(n) = O(n)$

(c) I think deterministic algorithm will be better in large cases.

① As for the time, the worst case for random algorithm is $O(n^2)$, which means that if we have the worst case, the time will be greatly larger than deterministic.

② As for the average performance, since the deterministic algorithm can guarantee the selection in linear time even in the worst case, the average performance will be better.