Before you start:

Homework Files

You can download the starter files for coding as well as this *tex* file (you only need to modify *homework0.tex*) on canvas and do your homework with latex (recommended). 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 0, you need to upload a **tar ball** with three files in the following format:

- VE281_HW0_[Your Student ID]_[Your name].tar
  - ⌊ VE281_HW0_[Your Student ID]_[Your name].pdf
  - ⌊ square.cpp
  - ⌊ sortList.cpp

For all programming questions (question 3.2 and 3.3), you need to successfully compile your code, or otherwise you will at most get half of the whole score(we will give you partial points if you implement some of the functionalities). We will have some simple test cases to test whether your code can correctly work.
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.**

0 Student Info (1 point)

Your name and student id:

Solution: Lan Wang, 519370910084

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) The output is:

$$\frac{n(n-1)}{2}$$

ii) The number of iterations is:

$$\begin{aligned} N &= \sum_{i=0}^{n-1}\sum_{j=0}^{i-1} 1 \\ &= \sum_{i=0}^{n-1} i \\ &= \frac{(0+n-1)n}{2} \\ &= \frac{1}{2}n^2 - \frac{1}{2}n \end{aligned}$$

Hence the time complexity of this code is $O(n^2)$.
Compared with the answer in part i), we can find that N equals to the output we get in i). This is not strange since we count the number of iteration in part ii), which is just what the given code is counting.

(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: The number of iterations is:

$$\begin{aligned} n &= \sum_{i=0,i\%2=0}^{N-1} \sum_{j=0}^{M/2-1} 1 + \sum_{i=0}^{K-1} 1 \\ &= \sum_{i=0,i\%2=0}^{N-1} \frac{M}{2} + K \\ &= \frac{N}{2}\frac{M}{2} + K \\ &= \frac{MN}{4} + K \end{aligned}$$

Hence the time complexity of this code is $O(MN + K)$.

(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: iv) and vi) are correct. We can first calculate the number of iterations:

$$\begin{aligned} N &= \sum_{i=n/2}^{n-1} \sum_{j=1,j=2^p}^{n-1} \sum_{k=0,k\%\sqrt{n}=0}^{n-1} 1 \\ &= \sum_{i=n/2}^{n-1} \sum_{j=1,j=2^p}^{n-1} \sqrt{n} \\ &= \sum_{i=n/2}^{n-1} (\log n)\sqrt{n} \\ &= \frac{n\sqrt{n}\log n}{2} \end{aligned}$$

Hence the time complexity of this code is $\Theta(n^{\frac{3}{2}} \log n)$.
So if the time complexity of this function is $O(T(n))$, $T(n)$ should increse faster than $n^{\frac{3}{2}} \log n$; if the time complexity of this function is $\Omega(T(n))$, $T(n)$ should increse slower than $n^{\frac{3}{2}} \log n$. So 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: Since in each recurrence there is just one comparison, the number of iterations is:

$$\begin{aligned} N &= \sum_{i=1}^{n} 1 \\ &= n \end{aligned}$$

Hence the time complexity of this code 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:

i) False. The time cost depends on the input. When the input size becomes larger, since the latter algorithm scales better, it will run faster.

ii) False. If the input size is small, the coefficients and constants can make the latter algorithm actually take less operations and run faster.

iii) False. The time cost depends on the input. When the input size becomes larger, since the latter algorithm scales better, it will run faster.

iv) False. If the input size is small, the coefficients and constants can make the latter algorithm actually take less operations and run faster.

2 Master Theorem (20 points, after Lec3)

2.1 Recurrence Relation (12 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:
When $n > 1$, $T(n) = 4T\left(\frac{n}{2}\right) + \Theta(n^2)$.
According to the Master Theorem, since $4 = 2^2$, $T(n) = \Theta(n^2 \log n)$

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

Solution:
When $n > 1$, $T(n) = 5T\left(\frac{n}{25}\right) + \Theta(\sqrt{n})$.
According to the Master Theorem, since $5 = 25^{1/2}$, $T(n) = \Theta(n^{1/2} \log n)$

(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:
When $n > 1$, $T(n) = 3T(n-1) + c$.

$$\Rightarrow T(n) + \frac{c}{2} = 3(T(n-1) + \frac{c}{2})$$
$$\Rightarrow \frac{T(n) + \frac{c}{2}}{T(1) + \frac{c}{2}} = 3^{n-1}$$
$$\Rightarrow T(n) = 3^{n-1}(c_0 + \frac{c}{2}) - \frac{c}{2} = \Theta(3^n)$$

2.2 Master Theorem on code (8 points)

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

```
1 void pie(int n) {
2   if (n == 1) {
3     return;
4   }
5   pie(n / 7);
6   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 the time required by this function be $T(n)$ when input is n.
The number of iterations in the function can be calculated as:

$$\begin{aligned} T(n) &= \sum_{i=0}^{n^2-1} \sum_{j=0}^{n-1} \log n + \log n^4 + T(n/7) + \sum_{k=0}^{n-1} T(n/3) \\ &= \sum_{i=0}^{n^2-1} n \log n + 4 \log n + T(n/7) + nT(n/3) \\ &= (n^3 + 4) \log n + T(n/7) + nT(n/3) \\ &= T(n/7) + nT(n/3) + O(n^3 \log n) \end{aligned}$$

3 Sorting Algorithms (49 points, after Lec4)

3.1 Sorting Basics (3 points)

What is the most efficient sorting algorithm for each of the following situations and briefly explain:

(a) A small array of integers.

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

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

Try to state your explanation from the following perspectives: The number of swaps / copies / compares; Time complexity; special cases depend on input; ...

Solution:

(a) Selection sort. Since the input size is small, the coefficients and constants affect the time more, so we consider the simple sort algorithms whose time complexities are $O(n^2)$. Since insertion sort needs some copies when insertion, and bubble sort needs some swaps to move the element to the front, they cost more time than selection sort on average cases, which just takes one swap to change the position of target element.

(b) Insertion sort. If the input array is almost sorted, both insertion sort and bubble sort have time complexity of $O(n)$. However, when we do bubble sort, we do swaps to move the element to the front, while for insertion sort, we do copies, which will be faster then using swaps.

(c) Count sort. We know that the input array contains just integers from a small range, so it's easy for us to form another list and use it to do count sort. Since the range is very small compared to input size n, the size of the other list is small compared to n, then the time complexity is nearly $O(n)$.

3.2 Squares of a Sorted Array (10 points)

Roihn is now doing his homework0 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.
For example, given an array: $\{-4, -2, 0, 1, 3, 7\}$, Roihn needs to return $\{0, 1, 4, 9, 16, 49\}$.
Here is Roihn'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 << ' ';
    }
    cout << endl;
    return 0;
}
```

However, 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 can modify this code however you like and briefly state your reason for why you modify it in that way and can satisfy the required time complexity.
You can find the code above in the given starter file *square.cpp*. Please modify the code above in *square.cpp* file, and upload the file together to canvas(DO NOT CHANGE THE FILE NAME!!). You can find more details for homework submission at the beginning of this homework.

Solution: (State your reason here)
Consider about the property of the input. Since there must be a zero, it's easy for us to devide the input to two parts: negative and positive. For each part, calculate the square, and then merge the two parts into a sorted array.
Since the division and calculation of squares are done in a linear scan, and the process of merging is also a linear scan, the time complexity is $O(n)$.

3.3 List Sort (20 points)

Given the *head* of a single linked list, return the list after sorting it in ascending order.
Input format: The first line of input is one variable n ($0 \leq n < 500$), indicating the size(length) of the array; The second line is n integers separated by space, which is the input link list *nums*. The first element in this line is the head value. It is guaranteed that all the integers are less than MAXINT.
Output format: One line with sorted array.

- **Sample 1:**

  Input:

```
5
4 5 3 2 1
```

Output:

```
1 2 3 4 5
```

- **Sample 2:**

  Input:

```
9
1 2 3 4 5 6 8 7 8
```

  Output:

```
1 2 3 4 5 6 7 8 8
```

We have provided you the starter code for *listNode* and related functions. You must use them for the whole question. The extra space complexity is required to be $O(1)$, which means you cannot use any other data structure (array, vector, etc.) to store your values.
You can use either type of sorting algorithm, but it is recommended to state your reason for why you choose that type of sorting for this question.

Solution: (State your reason here (Optional, not graded))

3.4 Quicker sort simulation (16 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 (8 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:

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

Roihn shares his answer for this array:

Solution:

Iter	Current Subarray	Pivot	Swapped Subarray	Current Array
1	$\{6, 2, 8, 10, 3, 1, 7\}$	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	$\{\}$	$\{1, 2, 3, 6, 8, 9, 10\}$

*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 Roihn 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:

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

Roihn 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: $\{2, 5, 6, 8, 10, 11\}$ $\{1, 3, 4, 7, 9\}$
20. Merge: $\{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11\}$

4 Selection Algorithm (10 points, after Lec5)

(a) As we have learned in the lecture, partition takes $O(n)$ time complexity. In terms of random selection algorithm, when will we encounter the worst-case scenario? What is the worst-case runtime for random selection algorithm? Is this related to input sequence? (3 points)

(b) Do we have a sequence of input which guarantees that it can have the best-case runtime for random selection algorithm? What is the best-case runtime for random selection algorithm? (2 points)

(c) Deterministic selection algorithm guarantees its runtime to be $O(n)$ for every input array of length n. Given the assumption that there exists a positive constant c that satisfies:

i) $T(1) \leq c$

ii) $T(n) \leq cn + T\left(\frac{n}{5}\right) + T\left(\frac{7n}{10}\right)$

Then prove that $T(n) = O(n)$. (Please clearly show your steps in detail) (5 points)

Solution:

(a) i) If the pivot is chosen to be the smallest (or the largest) element every time, and choose the target element at last. This means that we do $(n-1)$ times partition and n layer of recursion.

ii) $O(n^2)$

iii) No. No matter how the input is, each time we have to deal with a subarray which has all the element except the pivot in the original array, so always do $(n-1)$ times partition and n layer of recursion.

(b) i) An array with single element or with elements of the same value.

ii) $O(1)$. When the first chosen pivot is just what we want to find.

(c) If $T(n) = O(n)$, then there exists $a =$ constant such that $T(n) \leq an$ for all $n \geqslant 1$.
Choose $a = 10c$.
When $n = 1$, $\quad T(1) \leqslant c \leqslant 10c \cdot 1 = an$.
Assume that $\forall k \in \mathbb{N}^*, k < n$, we have $T(k) \leqslant 10ck$.
Then for n, we have:

$$\begin{aligned} T(n) &\leqslant a + T\left(\frac{n}{5}\right) + T\left(\frac{7n}{10}\right) \\ &\leqslant cn + 10c \cdot \frac{n}{5} + 10c \cdot \frac{7n}{10} \\ &= cn + 2cn + 7cn \\ &= 10cn \\ \Rightarrow T(n) &\leqslant 10cn \end{aligned}$$

$\Rightarrow T(n) \leqslant$ an for all $n \geqslant 1$, where $a = 10c$.
$\Rightarrow T(n) = O(n)$