Before you get started:

1. Your code in Q2 and Q3 will be graded by both an autograder and the TA. Therefore, please make sure that:
    a. Your code satisfies the requirements and is robust to corner cases.
    b. Do your best to optimize your algorithms (if no specific algorithm is required) rather than use brute force.
    c. For each required function, no header or main function is needed. There will be a complete script in which your function will be embedded for automatic testing, and your score will be given immediately. So, to avoid any unexpected 0 points, please use the exact function signature provided in each question and complete the function body, and make sure that the syntax is correct. Just in case, please also comment in the first line of your code to claim what libraries you've used.
    d. Keep your code clean, such as using clear variable names, whitespaces, comments, etc.
2. Be honest with yourself. The cost of plagiarism is always much more than expected as it consists of not only the external outcomes (e.g., failed assignments and courses), but also the moral burden on your conscience, which you will have to bear. Grading will be generous as long as you have tried your best.
3. If you have any questions about homework or in-class content, feel free to put your questions in the Discussions module on Canvas or show up in the instructor’s or TA’s office hours. Try everything as best you can instead of trying to find a shortcut.

Good luck! :D

1. (12 pts) **Mathematically** explain if each statement below is true or false.
    a) $4n^4 = O(4n)$
    b) $100n^3 + 99n^2 + 98n = O(100n^3)$
    c) $6n! = O(\log n)$
    d) $3/n = O(1/n)$
    e) $10 \log n = O(n)$
    f) $60 + 30 + 10 = O(100)$

2. (16 pts) You are given a random array consisting of integers (including positive, negative, and 0), and each item may be repeated.

   Note: for all the subquestions below, **please use the *iostream* library only.**

   a) (4 pts) Write a function ***bubbleSort*** to implement Bubble Sort to sort the array in **descending** order.
      - Function signature:
        ```
        void bubbleSort(int nums[], int n)
        ```
      - Example:
        - Input: *nums = [5, 1, 9, 6, 5, 1, 2], n = 7*
        - Expected realization: *nums = [9, 6, 5, 5, 2, 1, 1]*

   b) (6 pts) Write another function ***quickSort*** to implement Quick Sort to sort the array in **descending** order.
      - Function signature:
        ```
        void quickSort(int nums[], int n)
        ```
      - Example: see above.

   c) (3 pts) Write a function ***deduplicate*** to remove the repeated integers from your sorted array and return the length of the updated array. Do not use extra memory space.
      - Function signature:
        ```
        int deduplicate(int nums[], int n)
        ```
      - Example:
        - Input: *nums = [9, 6, 5, 5, 2, 1, 1], n = 7*
        - Expected realization: *nums = [9, 6, 5, 2, 1]*
        - Output: *5*

   d) (3 pts) What are the time and space complexity of each function that you realized above? Explain why.

3. (22 pts) Your little sibling needs your help with checking his/her solutions of a ton of math expressions, which are made up of **single** digits (0, 1, …, 9), arithmetic operators (“+”, “-”, “*”, “/” (integer division), “%” (mod)), and parentheses, but you prefer to save much time to have fun. For your cute and helpless sibling’s sake, please use what you learned to help him/her out!

   Note: for all the subquestions below, **you may leverage the code given in class or some other STLs.** A string expression terminates with “\0”.

   a) (7 pts) First, you may want to make sure that the handwritten expressions that your sibling copied from the board in class are valid (you’d not like to get lost in his/her silly writing mistakes).

      ★ An expression is valid if:

      - All the parentheses are balanced. That is, each left/right parenthesis can be paired with a right/left one, and there is no parenthesis left alone.
      - Within each pair of parentheses, there is exactly one operator between two operands (e.g., *(3+5)*), and each operand may also be an expression in a pair of parentheses (e.g., *(1+(2*3))*).

      How about writing a function ***checkValid*** to do it? It can return “True” if the expression is valid, or “False” otherwise (“0/1” Boolean expression is good as well).

      - Function signature:

        ```
        bool checkValid(char exp[])
        ```

      - Example 1:
        - Input: *exp = “(1+((2+3)*(4*5)))”*
        - Output: *True*
      - Example 2:
        - Input: *exp = “(1+((2+3)*))”*
        - Output: *False*

   b) (7 pts) Now you can make sure that all the expressions are valid. To help the computer parse and process expressions for you, it is necessary to transform an expression from infix to postfix. Let’s write a function ***inToPost*** to realize that! Put the resulting string

into the array *postExp* (assume sufficient memory space has been allocated), which terminates with "\0".

- Function signature:

```
void inToPost(char inExp[], char postExp[])
```

- Example:
  - Input: *inExp = "(1+((2+3)*(4*5)))"*
  - Expected realization: *postExp = "123+45**+"*

c) (5 pts) Almost there! Let's write the last function ***calculate*** to get the show on the road! It will take any postfix expression and get the final result.

- Function signature:

```
int calculate(char postExp[])
```

- Example:
  - Input: *postExp = "123+45**+"*
  - Output: *101*

d) (3 pts) What are the time and space complexity of each of the three functions above? Explain why.