Homework 2
CMSC 341 - Section 2 - Spring 2021
Due Date Feb. 21, 2021, 11:59 PM

Objectives for this homework:

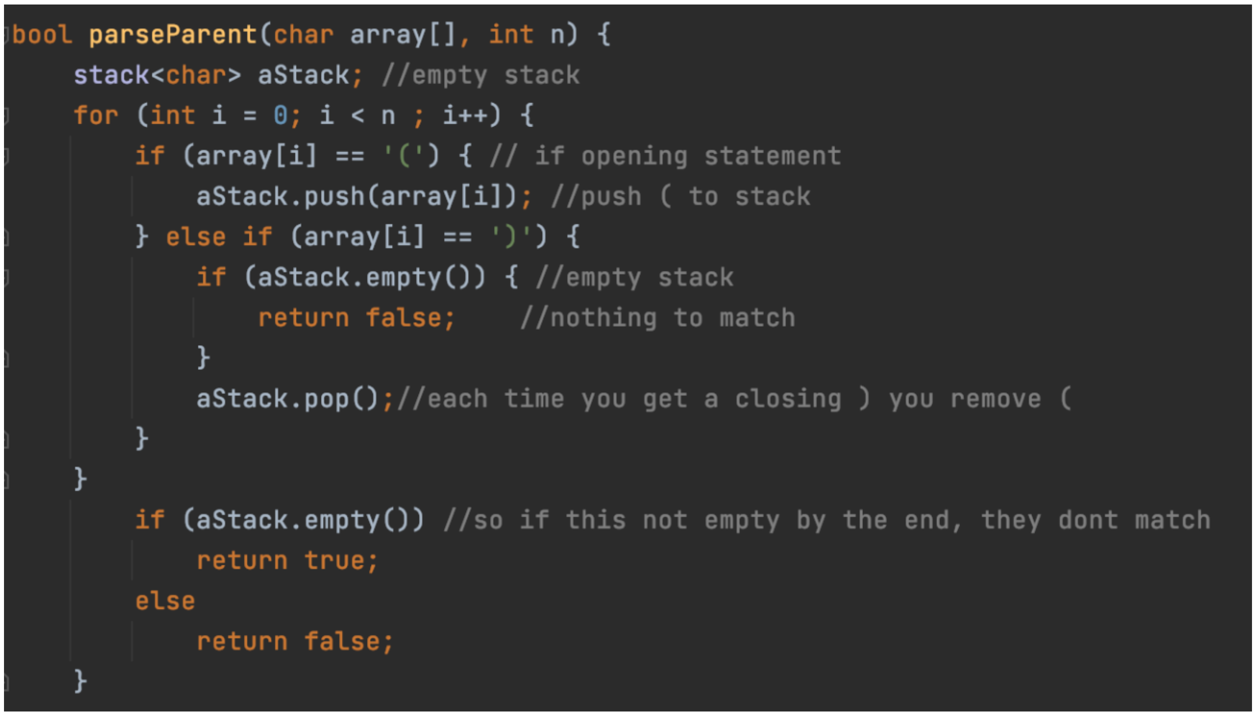
- Performing asymptotic analysis on an algorithm
- Exploring the relationship between Big-O, Big-Ω, and big-Θ
- Exploring the applications of stack ADT
- Practice STL template data structures
- Practice reading and understanding pseudocode

1 - A stack can be used in finding matching items in a sequence. For example, when we parse a mathematical expression, we may use a stack to find matching parentheses. Implement a C++ function that can check on an array of parentheses, and returns true if they all match, otherwise it returns false. The algorithm presented in Code Fragment 5.11 on page 205 (section 5.1.7) of the textbook can be used as a guideline. Use the STL stack class for stack ADT. To use the template class stack, you need to include <stack>. (20 points)

```
#include <stack>
bool parseParent(char array[ ], int n){
        stack<char> aStack;
        //To be implemented
}
```

The following list shows examples of array:
Correct: ()(()())(()), the function returns true
Incorrect: (()(()) , the function returns false

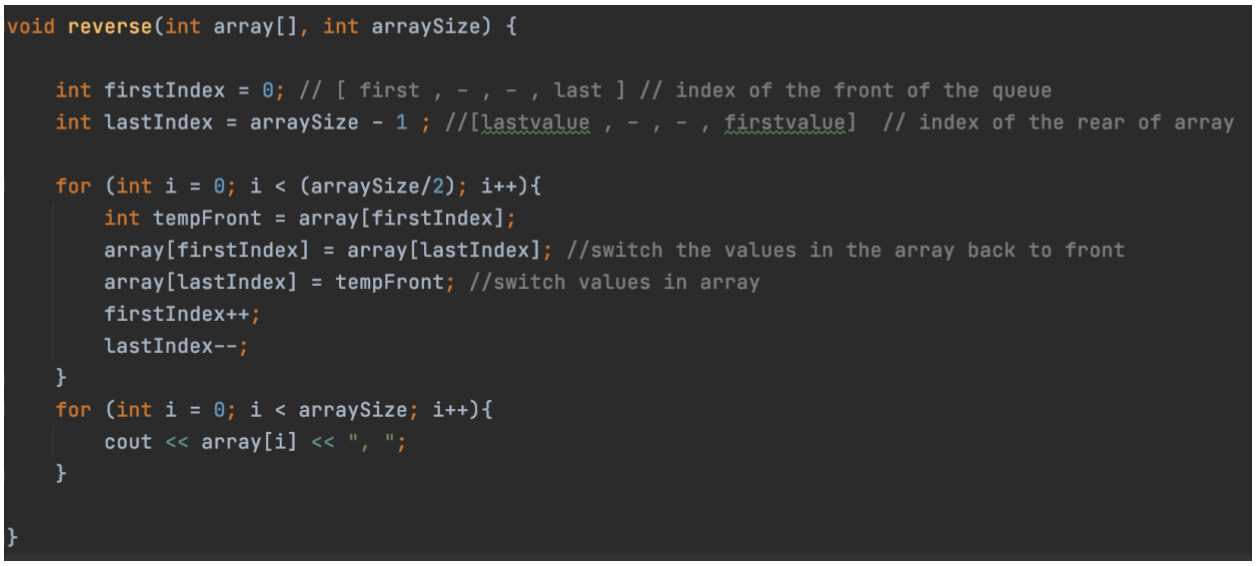
The operations for STL stack class are size(), empty(), push(...), pop(), and top(). The reference to the STL stack class is available at: http://www.cplusplus.com/reference/stack/stack/



```cpp
bool parseParent(char array[], int n) {
    stack<char> aStack; //empty stack
    for (int i = 0; i < n ; i++) {
        if (array[i] == '(') { // if opening statement
            aStack.push(array[i]); //push ( to stack
        } else if (array[i] == ')') {
            if (aStack.empty()) { //empty stack
                return false;    //nothing to match
            }
            aStack.pop();//each time you get a closing ) you remove (
        }
    }
        if (aStack.empty()) //so if this not empty by the end, they dont match
            return true;
        else
            return false;
    }
```

2 - Give a C++ code fragment for reversing an array. Your code should run in O(n) in regard to growth rate of running time and space. (exercise R-6.1 from textbook). Hint: we cannot use two arrays. We need to swap the elements of the original array. (20 points)

void reverse(int array[ ], int arraySize) {
        //to be implemented
}



```cpp
void reverse(int array[], int arraySize) {

    int firstIndex = 0; // [ first , - , - , last ] // index of the front of the queue
    int lastIndex = arraySize - 1 ; //[lastvalue , - , - , firstvalue]  // index of the rear of array

    for (int i = 0; i < (arraySize/2); i++){
        int tempFront = array[firstIndex];
        array[firstIndex] = array[lastIndex]; //switch the values in the array back to front
        array[lastIndex] = tempFront; //switch values in array
        firstIndex++;
        lastIndex--;
    }
    for (int i = 0; i < arraySize; i++){
        cout << array[i] << ", ";
    }

}
```

3 - The following function determines if all the numbers in the vector are different from each other, i.e. they are all distinct. Show that the function runs in O(n^2). Count the number of operations and determine the Big-O function. (20 points)
My work flow:

Size = 10 //use 10 as example

.i runs a.size()−1 times so 9 times // N times. there are three operation in the for statement but generalizing it to N times.

On each run of i

    j runs i+1… a.size() times so 9 times
    on the first run, 8 times on the next and so on till j = 10 i = 9
    in each run of j loop
    If statement runs j loop amount of times

return true is ran once

bool allDistinct(const vector<int>& a) {	# of primitive operations
for (int i = 0; i < a.size()−1; i++) {	N
for (int j = i+1; j < a.size(); j++) {	1 + 2 + 3 + … + 8 + (n – 1) summation
if (a[i] == a[j]) return false;	1 + 2 + 3 + … + 8 + (n – 1) summation
}	
}	
return true;	1
}	

N(n+1)/2 + 1

N + 2(n(n-1)/2) + 1

The highest power operation in the statement is (n(n-1)/2), and n*n = n^2. This makes n^2 the highest power. So this algorithm runs in O(n^2) where C = 1 , n_0 = 1

4 - What is the big-Θ characterization, in terms of n, for the running time of the following function? Please provide your complete work, i.e counting the primitive operations, and finding Big-O and Big-Ω functions. (20 points)

void aFunction(int n){	# of primitive operations
for (int i=0;i<n;i++)//10	N times
for (int j=0;j<n;j++) //10*10	N * N
cout << i*j << " ";	N * N
cout << endl;	1
}	

F (N + 2N^2 + 1)
= N + 2N^2 + 1
(2) N^2
Big Ω (N^2) The least amount of times a program will run. C = 2, n_0 = 0 //smallest power lowest boundary
The function must at least run a nested loop both i and j are < n which means this is a N^2 function
Big O (N^2) where C = 2, n_0 0 = 0 //highest power highest boundary
When big o and big omega are equal it means it is in big theta.
big-Θ is also (N^2) where c = 2 and n_0 = 0

5 - Find the values for n_0 and the constant factor c such that $f(n) = n \log n$ is $\Omega(n)$. Reminder, by definition we want to show ($n \log n \geq c\, n$) for ($n \geq n_0$). (20 points)

$n \log n \geq c\, n$ find: $n_{0 \text{ and } C}$

$n \log n$ is $\Omega(n)$. where c = 1, and $n_0 = 2$