

Prob1.

1) The data structure I used is just an integer type variable initialized to 0. Since a proper matched parenthesis should always have more '(' than ')' when looked at the front, I incremented the variable whenever the input character was '(' and decremented it when input was ')'. If the variable gets negative, which means there are more ')' than '(', it is wrong. If the value at the end of input is not 0, it is also wrong. If neither of the case, the input is correct.

2) I'm sure this algorithm is correct because the logic for the algorithm seems right. And I tested checker with the outputs of the generator. And they were all correct.

3) If the input line is size of n , the for loop runs n times so I think the time complexity of the algorithm is $O(n)$. In order to test this, I used the time function to check the time as input grows. When input length was small, it was hard to find the difference so I put a long 1 line input to the checker.

```

junbum@ubuntu: ~/Desktop/20180155pj2
File Edit View Search Terminal Help
junbum@ubuntu:~/Desktop/20180155pj2$ time ./checker < out.txt
correct 1, wrong 0

real    0m0.004s
user    0m0.004s
sys     0m0.000s
junbum@ubuntu:~/Desktop/20180155pj2$ time ./checker < out.txt
correct 1, wrong 0

real    0m0.013s
user    0m0.008s
sys     0m0.004s
junbum@ubuntu:~/Desktop/20180155pj2$ time ./checker < out.txt
correct 1, wrong 0

real    0m0.048s
user    0m0.021s
sys     0m0.000s
junbum@ubuntu:~/Desktop/20180155pj2$

```

>input length :
100,000 - 0.004s
500,000 - 0.013s
1,000,000 - 0.048s

It is not exact, but it seems that as input grows the execution time grows linearly. So I assumed that $O(n)$ is correct. For, space complexity since I used only an Integer variable, it will be $O(n)$.

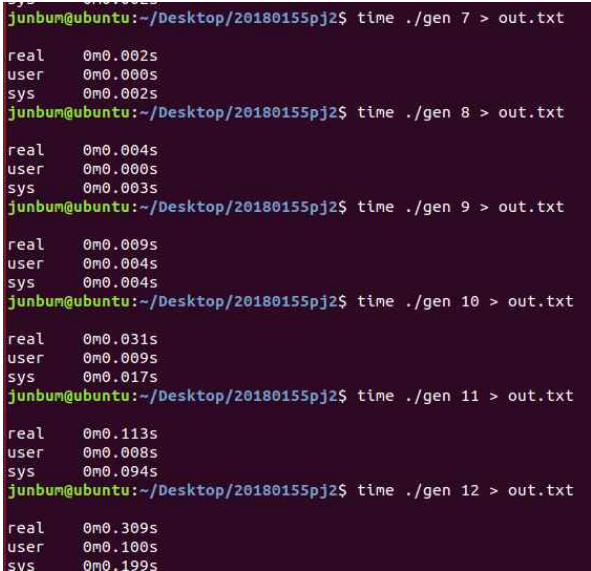
Prob 2.

1) For this problem I used `std::string` data structure to create the strings. In creating the correct matched parenthesis, I thought of a way of adding '(' or ')' to the string recursively. But I found that if there are same number of '(' and ')', you can only add ')'. Other than that, you can add '(' or ')' any time until the number of '(' and ')' reaches n . In recursive program, the more you go in to recursive stage, the longer it takes. so I tried to minimize the recursive stages as possible. So I added a condition when the number of open parenthesis is equal to n , then we don't go to recursive any more and just add close parenthesis. Also, when '(' and ')' are same number, then just add ')' and go into next recursive stage.

2) I think the algorithm is correct because the number of the created output was equal to the catalan number. And when I put it in the checker, created string were all correct. I also think the logic for algorithm for this is right.

3) Since this program is based on recursive program. I think the time complexity of the program is $O(2^n)$. Because when creating the string, we may add '(' or ')' so there are at most two possible for every recursive stage.

input length	execution time
7	0.002s
8	0.004s
9	0.009s
10	0.031s
11	0.113s
12	0.309s



The terminal screenshot shows the following commands and outputs:

```
junbum@ubuntu:~/Desktop/20180155pj2$ time ./gen 7 > out.txt
real    0m0.002s
user    0m0.000s
sys     0m0.002s
junbum@ubuntu:~/Desktop/20180155pj2$ time ./gen 8 > out.txt
real    0m0.004s
user    0m0.000s
sys     0m0.003s
junbum@ubuntu:~/Desktop/20180155pj2$ time ./gen 9 > out.txt
real    0m0.009s
user    0m0.004s
sys     0m0.004s
junbum@ubuntu:~/Desktop/20180155pj2$ time ./gen 10 > out.txt
real    0m0.031s
user    0m0.009s
sys     0m0.017s
junbum@ubuntu:~/Desktop/20180155pj2$ time ./gen 11 > out.txt
real    0m0.113s
user    0m0.008s
sys     0m0.094s
junbum@ubuntu:~/Desktop/20180155pj2$ time ./gen 12 > out.txt
real    0m0.309s
user    0m0.100s
sys     0m0.199s
```

As input grows linearly, the execution time seems to increase exponentially, so I believe that $O(2^n)$ is correct for the time complexity.

For space complexity, I think (at most) length n strings are 2^n times created and destroyed so $O(n \cdot 2^n)$ might be the complexity for space. But I'm not really sure about this because recursive functions will likely be executed simultaneously.

Prob 3.

1) For this problem I created a Node class and Sorted Linked List class.

the node refers to the line segment which has a private integer value for start point(offset), end point, and length of the line. The Sorted Linked List(SLL) will gather these nodes(lines) and merge them. The SLL has 3 public functions addLine, mergeLine1, and mergeLine2. The addLine function gets Node as an input and adds it to the Linked List in a increasing order of "offset". MergeLine1 function then checks if there are line segments that has the same offset, and if so, it only leaves the one with the largest length and deletes all the other. MergeLine 2 then merges line by comparing the "offset + length" value and the next node's offset.

2) I think this algorithm works well because I tried testing this with many different inputs, and same inputs with difference order of line and they all worked. And I'm sure all inputs get be merged by the 3 processes which I divided as functions.

3)

Time complexity.

- addLine : since it has to check whether it is sorted or not every time the input comes in, worst case will be $1+2+3 \dots + n$, which is $O(n^2)$.

- mergeline1 : it checks with 2 for loop, but it deletes the nodes as it goes so I think time complexity for this function is $O(n)$. because if the for loop doesn't delete any node, it means that it doesn't have any line segment with same offset. And the second for loop will be meaningless. and if it delete node every time, the first for loop is meaning less.

- mergeline 2: it checks with 2 for loop, but as with similar logic as mergeline1, I think the time complexity for this function is $O(n)$

so the Total time complexity for this function is $O(n^2)$ overall.

	>input length - execution time
real 0m0.014s user 0m0.013s sys 0m0.000s junbun@ubuntu:~/Desktop/20180155pj2\$ time ./merger < out.txt >test.txt	3000 - 0.014s
real 0m0.060s user 0m0.049s sys 0m0.004s junbun@ubuntu:~/Desktop/20180155pj2\$ time ./merger < out.txt >test.txt	6000 - 0.060s
real 0m0.262s user 0m0.240s sys 0m0.000s junbun@ubuntu:~/Desktop/20180155pj2\$	12000 - 0.262s

As input length gets twice, the execution times grows about 4 times. So I think $O(n^2)$ is correct.

Space complexity is $O(n)$ because it will have n nodes if the input is n .

4) prob 1, 2 didn't take me a lot of time but prob 3 took me about 5,6 hours because at first I didn't use the sorted Linked list and I was having a hard time. Then I deleted all the code and started again with sorted Linked List.

5) I didn't collaborated with anyone. I only looked at the Data Structure lecture, textbook and questions in Classum.

6) I think I can improve the prob 3's time complexity by adjusting the addLine function with is the only $O(N^2)$. I think may be by using quick sort or other sorting algorithm, the time complexity will be better than before, But It will increase the space complexity for sure.