Questions

0. Predict the output of this snippet:

```c
int main() {
   int main = 56;
   printf("%d", main);
   return 0;
}
```

   (a) Compiler Error

   (b) Depends on the compiler

   (c) 56 ←

   (d) none of the above

1. Predict the output of this snippet:

```c
#include <stdio.h>
int main() {
   char ch;
   if (ch = printf("")) {
      printf("It matters\n");
   }
   else {
      printf("It doesn't matter\n");
   }
   return 0;
}
```

   (a) It matters

   (b) It doesn't matter ←

   (c) Runtime error

   (d) Nothing

2. How many times is Hello world printed by the code snippet below?

```c
int main() {
   fork();
   fork();
   printf("Hello world\n");
}
```

   (a) 1

   (b) 2

   (c) 4 ←

   (d) 8

3. What is the output of this C code?

```c
#include <stdio.h>
int main() {
    int x = 1, y = 0, z = 5;
    int a = x && y || ++z;
    printf("%d", z++);
}
```

(a) 1

(b) 5

(c) 6 ←

(d) 7

4. What is the output of this C code?

```c
#include <stdio.h>
int main() {
    int y = 2;
    int z = y +(y = 10);
    printf("%d\n", z);
}
```

(a) 2

(b) 4

(c) 20 ←

(d) Compile time error

5. What is the output of this C code?

```c
#define max(a) a
int main() {
    int x = 1;
    switch (x)
    {
        case max(2):
            printf("yes\n");
        case max(1):
            printf("no\n");
        break;
    }
}
```

(a) yes

(b) no ←

(c) Runtime error

(d) Compile time error

6. What is the output of this C code?

```c
#include <stdio.h>
int main() {
   int x = 35;
   printf("%d %d %d", x == 35, x = 50, x > 40);
   return 0;
}
```

(a) 1 50 1

(b) 0 50 0 ←

(c) Runtime error

(d) Compile time error

7. You have two numbers, A and B.

   Using bitwise operations and loops, write code to determine how many bits must be flipped in order to turn A in to B.

   e.g.

   A: 101001

   B: 10**0**1**0**1

   Answer: 2 bits

```c
int flipBits (int a, int b) {
   unsigned int count = 0;
   unsigned int i = a ^ b;
   while (i != 0) {
      int j = i & 1; // 0b00000000000000000000000000000001
      i = i >> 1;
      if (j != 0) {
         count++;
      }
   }
   return count;
}
```

8. Presume you have an implementation of Quick-sort that picks the first element in a list as its pivot.

   Construct a list of 13 numbers that would make this algorithm run in $O(n^2)$ time.

   Solution: $1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13$

9. Write code to solve the below in $O(n)$ time with one pass over the array. If you can not, provide your best code and state the asymptotic running time.

   In this challenge, given an array of integers, the goal is to efficiently find the subarray that has the greatest positive value when all of its elements are summed together.

   Write some code that, if given an array of integers, will find the subarray that has the greatest sum. A subarray is a contiguous subset of the original array.

   Keep in mind the maximal subarray is not necessary a proper subset, and might contain the whole original array.

   Be careful. Some array elements may be negative, so less can be more.

   If all the elements in an array are negative, the maximal sum should be null as we are interested in the greatest positive value, not the greatest arithmetic value.

   For example, given the array: $\{1, 2, -5, 4, -3, 2\}$

   - The maximum sum of a subarray is 4 and it contains only the element 4.

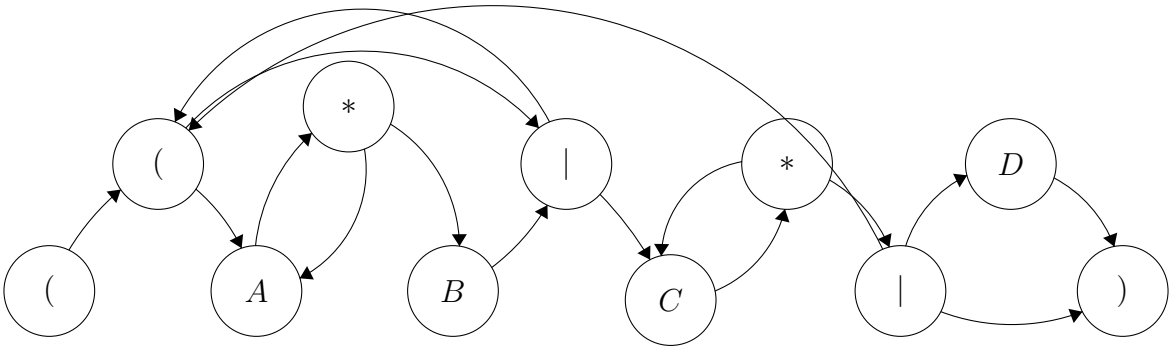
   Before you write the code, take some time to think of the most efficient solution possible; it may surprise you. The major goal of this question is to test your algorithmic skills rather than merely your ability to write code quickly.

```
int maxSubarray(int* array, int len, int** pArrayOut, int* pLenOut) {
   // Max subarray so far
   int   sumMax      = 0;
   int   startIdxMax = 0;
   int   lenMax      = 0;
   // Max of subarray end at Index
   int   sumMaxEndHere     = 0;
   int   startIdxMaxEndHere  = 0;
   int   lenMaxEndHere     = 0;

   int i;
   for (i=0; i < len; i++) {
      // Find sumMaxEnd at this Index
      if (sumMaxEndHere > 0) {
         sumMaxEndHere     = sumMaxEndHere + *(array+i);
         //startIdxMaxEndHere = startIdxMaxEndHere;
         lenMaxEndHere        = lenMaxEndHere + 1;
      }
      else {
         sumMaxEndHere     = *(array+i);
         startIdxMaxEndHere   = i;
         lenMaxEndHere     = 1;
      }
      if (sumMaxEndHere > sumMax) {
         sumMax      = sumMaxEndHere;
         startIdxMax = startIdxMaxEndHere;
```

```
27            lenMax      = lenMaxEndHere;
28         }
29      }
30      if (pArrayOut) {
31         *pArrayOut = array + startIdxMax;
32      }
33      if (pLenOut) {
34         *pLenOut = lenMax;
35      }
36      return sumMax;
37   }
```

10. RE pattern matching.
Draw an NFA (nondeterministic finite state automaton) that recognizes the same language that the regular expression $((A^*B|C)^*|D)$



NFA for $((A^*B|C)^*|D)$