1) Please determine a Big-Oh bound for the following recurrence using the iteration technique:

$$T(n) = 4T\left(\frac{n}{2}\right) + 1$$

Note: $\sum_{i=0}^{k-1} 4^i = \frac{4^k - 1}{3}$.

Solution

Iterate the recurrence three times:

$$T(n) = 4T\left(\frac{n}{2}\right) + 1$$
$$T(n) = 4(4T\left(\frac{n}{4}\right) + 1) + 1$$
$$T(n) = 16T\left(\frac{n}{4}\right) + \sum_{i=0}^{1} 4^i$$
$$T(n) = 16(4T\left(\frac{n}{8}\right) + 1) + \sum_{i=0}^{1} 4^i$$
$$T(n) = 64T\left(\frac{n}{8}\right) + \sum_{i=0}^{2} 4^i$$

Now we guess the general form of the recurrence after k iterations to be:

$$T(n) = 4^k T\left(\frac{n}{2^k}\right) + \sum_{i=0}^{k-1} 4^i$$

To solve the recurrence, plug in the value of k for which $\frac{n}{2^k} = 1$, alternatively, when $n = 2^k$. Note that $4^k = (2^2)^k = (2^k)^2 = n^2$.

$$T(n) = n^2 T(1) + \frac{4^k - 1}{3}$$
$$T(n) = n^2 T(1) + \frac{n^2 - 1}{3}$$
$$T(n) = (T(1) + \frac{1}{3})n^2 - \frac{1}{3}$$
$$T(n) = O(n^2)$$

Note, that T(1) is a constant, so we can take the last step, since any function that is less than or equal to cn^2 for some constant c is $O(n^2)$.

2) Write a ***recursive*** function, `equal`, that takes in pointers to two linked lists and returns 1 if the two lists are equal and 0 otherwise. For two lists to be equal, they have to have the same number of elements, and each corresponding element must be equal. (For example, the lists 3, 9, 5 and 3, 9, 5 are equal, but the lists 3, 4, 7 and 3, 7, 4 are not equal and the lists 2, 9, 1 and 2, 9, 1, 4 are not equal.)

```
typedef struct node {
    int data;
    struct node* next;
} node;

int equal(node* listA, node* listB) {

    if (listA == NULL && listB == NULL) return 1;

    if (listA == NULL || listB == NULL) return 0;

    if (listA->data != listB->data) return 0;

    return equal(listA->next, listB->next);

}
```

3) A algorithm that sorts n items runs in $O(n\sqrt{n})$. When run on an input of 10,000 items, the algorithm takes 200 milliseconds. How long, in seconds, will the algorithm take when run on an input of 90,000 items?

Let T(n) represent the run time of the algorithm. Let $T(n) = cn\sqrt{n}$, for some constant c. Using the given information, we have:

$$T(10000) = c(10000)\sqrt{10000} = .2\ seconds$$
$$c(10^4)(10^2) = .2 seconds$$
$$c = 2 \times 10^{-7} seconds$$

Now, solve for T(90000):

$$T(90000) = c(90000)\sqrt{90000}$$
$$= (2 \times 10^{-7} seconds)(9 \times 10^4)\sqrt{9}\sqrt{10000}$$
$$= 2 \times 10^{-7} \times 9 \times 10^4 \times 3 \times 10^2 seconds$$
$$= 54 \times 10^{-1} seconds$$
$$= 54 \times 10^{-1} seconds$$
$$= 5.4\ seconds$$

4) We define a set of strings s_1, s_2, ... as follows: s_1 = "1" and to form s_{i+1} we stick together two copies of s_i next to each other followed by the character i+1. For example, s_2 = 112 and s_3 = 1121123. Write a function that takes in n (guaranteed to be in between 1 and 9, inclusive) and prints out s_n.

```
void printSequence(int n) {
    if (n > 0) {
        printSequence(n-1);
        printSequence(n-1);
        printf("%d", n);
    }
}
```

5) Determine a closed form solution for the following summation, in terms of n:

$$\sum_{i=n}^{2n-1}(4i+7)$$

$$\begin{aligned}
\sum_{i=n}^{2n-1}(4i+7) &= 4\left(\sum_{i=n}^{2n-1} i\right) + 7(2n-1-n+1) \\
&= 4[\left(\sum_{i=1}^{2n-1} i\right) - \left(\sum_{i=1}^{n-1} i\right) + 7n \\
&= 4\left[\frac{(2n-1)(2n)}{2} - \frac{(n-1)n}{2}\right] + 7n \\
&= 2n[2(2n-1)-(n-1)] + 7n \\
&= 2n[4n-2-n+1] + 7n \\
&= 2n[3n-1] + 7n \\
&= 6n^2 - 2n + 7n \\
&= 6n^2 + 5n \\
&= n(6n+5)
\end{aligned}$$

6) Determine the run-time, in terms of the formal parameter n, of the following function. Leave your answer in a Big-Oh bound and justify your answer.

```
int f(int array[], int n) {

    int i, total = 0;
    for (i=0; i<n; i++) {
        int low = 0, high = n-1;
        while (low < high) {
            int mid = (low+high)/2;
            if (2*array[i] < array[mid])
                high = mid-1;
            else
                low = mid+1;
        }
        total += low;
    }
    return total;
}
```

The outer loop runs n times. The inner loop runs roughly log n times, since its code structure is similar to a binary search. Namely, the difference between low and high is roughly n at the beginning and at each while loop iteration, we divide this difference by about 2. This sets up the equation $n/2^k = 1$, where k is the maximum number loop iterations. The solution is $k = \log_2 n$. It follows that the total run time of the segment of code is O(nlgn).

7) Consider the following recursive function:

```
int compute(int array[],int low, int high) {

    if (low == high) return array[low]%3 + 1;

    int mid = (low+high)/2;
    int left = compute(array, low, mid);
    int right = compute(array, mid+1, high);
    return left*right;
}
```

Consider the function call `compute(array, 0, 6)` where array is shown below:

index	0	1	2	3	4	5	6
array	17	4	19	30	47	999	13

Determine the result of this recursive call, as well as each other recursive call that gets made as a result of this original one and its return value. Please fill in the recursive calls in the order that they *start*. (Note: This order is different than the order in which they finish and a significant hint has been given to you below.)

Recursive Call	Return Value
`compute(array, 0, 6)`	**72**
`compute(array, 0,` **3** `)`	**12**
`compute(array, 0,` **1** `)`	**6**
`compute(array, 0,` **0** `)`	**3**
`compute(array, 1,` **1** `)`	**2**
`compute(array, 2,` **3** `)`	**2**
`compute(array, 2,` **2** `)`	**2**
`compute(array, 3,` **3** `)`	**1**
`compute(array, 4,` **6** `)`	**6**
`compute(array, 4,` **5** `)`	**3**
`compute(array, 4,` **4** `)`	**3**
`compute(array, 5,` **5** `)`	**1**
`compute(array, 6,` **6** `)`	**2**

8) Complete the program below so that it prints out all the permutations of 0,1,2, ...,SIZE-1 such that the absolute value of the difference between each pair of adjacent numbers in the permutations is 2 or greater. For example, when SIZE = 4, the code would print out:

```
1 3 0 2
2 0 3 1
```

the only 2 permutations such that the absolute value of the difference between each pair of adjacent terms is 2 or greater.

```c
#include <stdio.h>
#include <math.h>

#define SIZE 4

void printPerms(int perm[], int used[], int k, int n);
void print(int perm[], int n) ;

int main() {
    int perm[SIZE], used[SIZE], i;
    for (i=0; i<SIZE; i++) used[i] = 0;
    printPerms(perm, used, 0, SIZE);
    return 0;
}

void printPerms(int perm[], int used[], int k, int n) {

    if (k == n) print(perm, n);

    int i;
    for (i=0; i<n; i++) {
        if (!used[i]) {
            if ( k == 0 || abs(perm[k-1]-i) >= 2 ) {

                used[i] = 1 ;

                perm[k] = i ;

                printPerms( perm ,used ,k+1 ,n);

                used[i] = 0 ;
            }
        }
    }
}
```

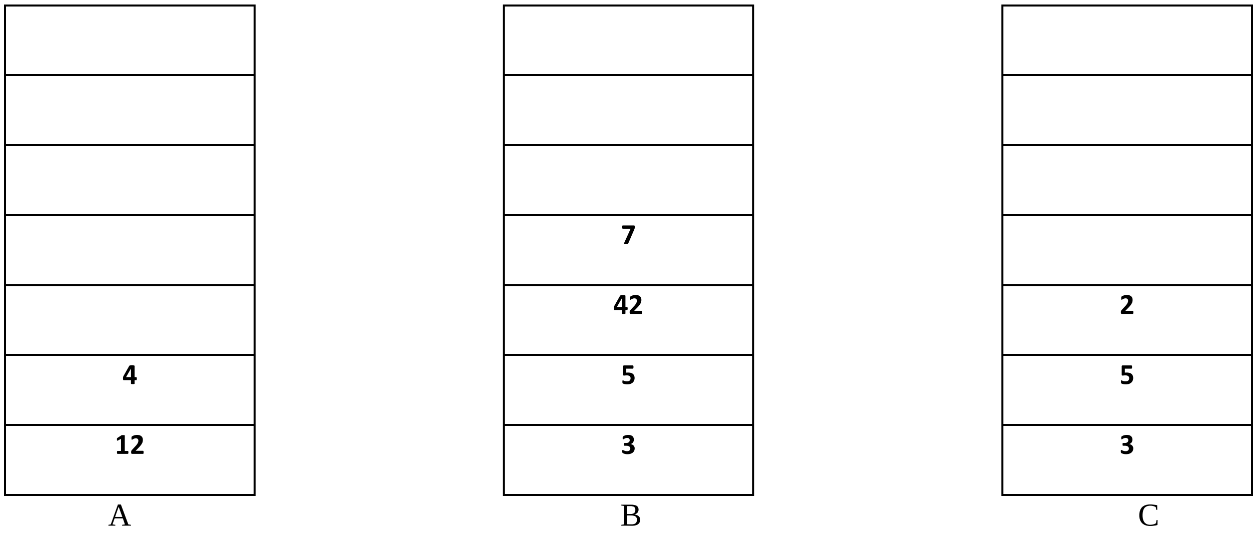
9) Write a function that takes in pointers to two sorted linked lists, combines them by rearranging links into one sorted linked list, effectively merging the two sorted lists into one, and returns a pointer to the new front of the list. Since no new nodes are being created and no old nodes are being deleted, your code should **NOT** have any mallocs or frees. Also, note that this destroys the old lists. If you try to print either listA or listB after calling merge, the lists are likely to print differently. (***Hint: This is probably much easier to do recursively.***) Fill in the function prototype provided below and use the struct provided below:

Solution

```
typedef struct node {
    int data;
    struct node* next;
} node;

node* merge(node* listA, node* listB) {

    if (listA == NULL) return listB;
    if (listB == NULL) return listA;

    if (listA->data < listB->data) {
        listA->next = merge(listA->next, listB);
        return listA;
    }

    listB->next = merge(listA, listB->next);
    return listB;
}
```

10) Evaluate the following postfix expression, showing the state of the operand stack at the three points A, B and C indicated below:

```
                     A                   B              C
12    6    2    -        /    5    42    7    /    4    -    *    +
```



Value of the Expression: **13**

11) Circle either True or False about each of the following assertions about queues.

a) A queue is a Last In, First Out (LIFO) abstract data structure. True **False**

b) If a queue is implemented with a regular linked list, with a pointer to the front of the queue only, the enqueue operation would take Θ(n) time for a list with n elements. (Θ indicates proportional to n.) **True** False

c) If a queue is implemented with a regular linked list, with a pointer to the front of the queue only, the dequeue operation would take Θ(n) time for a list with n elements. True **False**

d) A queue must be implemented with a linked list. True **False**

e) A queue allows for access to any of its elements in O(1) time. True **False**

12) Convert the following infix expression to postfix, showing the state of the operator stack at the three points A, B and C indicated below:

$$(\; (\; 42 \; - \; 9 \;) \; / \; (\; 2 \; + \overset{\mathbf{A}}{\;} 5 \; - \; 2 \; * \; 2 \;) \overset{\mathbf{B}}{\;} - \; 2 \; * \; 3 \;) \overset{\mathbf{C}}{\;} * \; (\; 3 \; + \; 4 \;)$$

+		
(		
/	**/**	
(	**(**	*****
A	B	C

Equivalent Postfix Expression:

42 9 - 2 5 + 2 2 * - / 2 3 * - 3 4 + *