Practice Problems for Exam 1

Linked Lists

Create a function that accepts the head of a singly linked list and prints the linked list in reverse.

```
struct Node {

  int data;

  Node* next;

};

void print(Node *head){ // write code below



}
```

```
struct Node {
  int data;
  Node* next;
};
void print(Node *head){ // write code below
        if(head == nullptr){
                return;
        }
        print(head->next);
        cout << head->data << " ";
}
```

Given a double-linked list, write a function that prints nodes that are “peak”. A node is considered “peak” if the one before and after are less than the node. Desired output for given examples:
Desired output:
a. [1,2,5,4] -> 5
b. [1,2,6,3,7,5] -> 6,7
c. [1,2,3] -> No output

```
struct Node {

        int data;

         Node* prev;

        Node* next;

};

void peak(Node* head) { // write code below

}
```

```
struct Node {
	int data;
	Node* prev;
	Node* next;
};
void peak(Node* head) { // write code below
	Node* cu = head->next;
	while(cu->next != nullptr) {
		if(cu->prev->data < cu->data && cu->data > cu->next->data)
			cout << cu->data << " ";
		cu = cu->next;
}
```

Write a function addTwoNumbers that accepts 2 doubly linked lists as input(the lists already created). It then adds the values from the nodes. Returns the head of the new list. Each node value will be an integer [0,9].
For example to add 387 + 214 = 601 will be
List 1 : 3 ⇆ 8 ⇆ 7
List 2 : 2 ⇆ 1 ⇆ 4
Result : 6 ⇆ 0 ⇆ 1

```
struct Node {

 int data;

 Node* next;

};

Node* addTwoNumbers(Node* tail_1, Node* tail_2) {



}
```

```
struct Node {
  int data;
  Node* next;
};

Node* addTwoNumbers(Node* tail_1, Node* tail_2) {
            int sum = 0;
           int carry = 0;
            linkedlist result;
           while(tail1 != nullptr || tail2 != nullptr || carry != 0)
           {
                    sum = 0;
                    if(tail1 != nullptr) {
                             sum += tail1->data;
                             tail1 = tail1->prev;
                    }
                    if(tail2 != nullptr) {
                             sum += tail2->data;
                              tail2 = tail2->prev;
                    }
                    sum += carry;
                    carry = sum/10;
                    result.addAtBeg(sum%10);
            }
           return result.getHead();
}
```

Write a function to recursively count the number of nodes in a singly linked list with even values. This function should return the number of nodes in the list with even values.

```
struct Node {

 int val;

 Node* next;

};

int evenCount(Node *head){ // write code below



}
```

```
struct Node {
 int val;
 Node* next;
};
int evenCount(Node *head){
 if(head == nullptr){
  return 0;
 }
 if(head->data % 2 == 0){
  return 1 + evenCount(head->next);
 }
 return evenCount(head->next);
}
```

Implement removeNth which removes the n^{th} node from the end of the list and returns the head of the linked list. You may assume n will be less than the size of the linked list. Ex:
removeNth(1 -> 2 -> 3, 2) => 1 -> 3
removeNth(6 -> 5 -> 4 -> 2 -> 1 -> 0, 1) => 6 -> 5 -> 4 -> 2 -> 1
removeNth(nullptr) => nullptr

```
struct Node {

 int value;

 Node* next;

};

Node *removeNth(Node *head, int n){ // write code below



}
```

```
struct Node {
  int value;
  Node* next;
};

Node *removeNth(Node *head, int n){
  if(head == nullptr){
    return head;
  }
  else{
    int size = 0;
    Node *cu = head;
    while(cu != nullptr){
      size++;
      cu = cu->next;
    }
    int pos = size - n;
    Node *prev = nullptr;
    cu = head;
    for(int i=0; i<pos; i++){
      prev = cu;
      cu = cu->next;
    }
    prev->next = cu->next;
    delete cu;
    return head;
  }
}
```

Given a doubly linked list and the head, implement a function that will print the list backwards. Ex:

{1 -> 5 -> 6 -> 4 -> 7} => 7 4 6 5 1

{1} => 1

{} => {}

```
struct Node {
  int value;
  Node* next;
  Node* prev;
};

void print(Node *head){ // write code below

}
```

```
struct Node {
 int value;
 Node* next;
 Node* prev;
};
void print(Node *head){ // write code below
        if(head != nullptr){
                Node *cu = head;
                while(cu -> next != nullptr){
                        cu = cu->next;
                }
                while(cu -> prev != nullptr){
                        cout << cu-> value << " ";
                        cu = cu->prev;
                }
                cout << cu -> value;
        }
}
```

Recursion

Write a RECURSIVE function that gets the factorial of the number n that's passed into the function. The factorial is the product of all numbers leading up to n. For example 4! (4factorial) = 4 * 3 * 2 * 1 = 24?

```
int factorial(int n){



}
```

```
int factorial(int n)
{
 if (n == 0)
        return 1;
 return n * factorial(n - 1);
}
```

Consider a palindrome an array of numbers, such as 76967 or 566665. Write a recursive function palindrome that returns true if the array is a palindrome and false if it is not

```
bool isPalindrome(int* arr, int low, int high) {



}
```

```
bool isPalindrome(int* arr, int low, int high)
{
if (low >= high)
        return true;
if (arr[low] != arr[high])
        return false;
return isPalindrome(arr, low + 1, high - 1);
}
```

What is the output?

```
#include<iostream.h>
void fun(int x)
{
        if(x > 1)
        {
                fun(--x);
                cout << --x <<" ";
                fun(x-1);
        }
}
int main()
{
 int a = 5;
 fun(a);
 return 0;
}
```

01230

Big O

What is the time complexity of the following code?

```
for(int i = 0; i < 10000000; i++) {
        for(int j = 0; j < n; i++) {
                cout << "Test" << endl;
        }
}
```

O(10000000*n) = O(n)

Even though this is a nested loop, the outer for loop has the time complexity of O(10000000) and this is a constant can be rewritten as O(1)

What is the time complexity of the following code?

```
void myFunc(int n, int m) {
        if(n <= 0) return;
        for(int i = 0; i < m; i++) {
                myFunc(n-1,m);
         }
}
```

O(m^n)

For each recursive call, we repeat the for loop of O(m), the recursive will be called for O(n) times

⇨ O(m) * O(m) * O(m) * ... (multiplies n times)

• In recursion we will have exponential time complexity, The base (the number that is being multiplied by itself) will depend on how many times you call the recursive function.

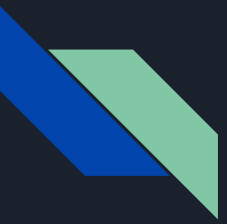
Ex: In recursive fibonacci you call recursion function twice (fib(n-2) + fib(n-1)), hence fibonacci have a time complexity of O(2^n)

If you have a recursive function that call itself 3 times, the time complexity will be O(3^n) and so on

Give the time complexities for the following two functions and explain which one has a shorter time complexity.

```
void func_a(int n) {
int i = 0;
int j = 0;
while(i < n) {
        while(j < n) {
                j = j * j;
        }
        j = 0;
        i += 1;
}
}
```

```
void func_b(int n) {
int i = 0;
int j = n;
while(i < n) {
        while(j > 0) {
                j = j - 2;
        }
        j = 0;
        i += 1;
}
}
```



O(n * sqrt(n))

O(n^2)

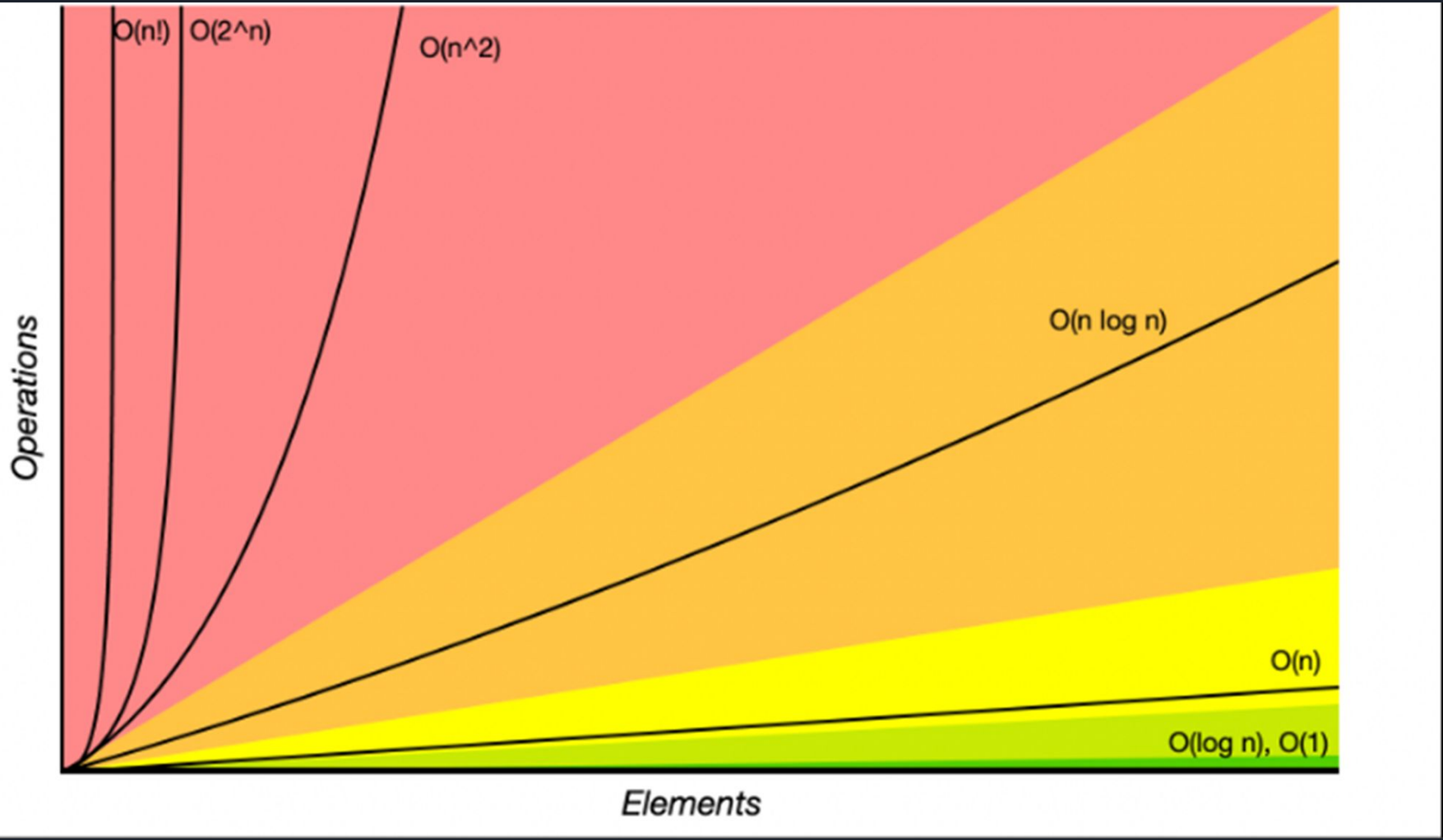
What is the best Time Case time complexity for Selection Sort? Describe what the list must look like for it to have this time complexity.

Answer:
O(n^2)
List must be sorted already to have the best time complexity

What is the time complexity for:

1) Adding an element to an array
2) Inserting an element into an array
3) Adding two arrays
4) Adding at a specific position in a linked list
5) Searching for a specific element in an array
6) Binary search
7) Adding at the head of a linked list
8) Adding at the tail of a linked list with only a pointer to the head
9) Adding at the tail of a linked list with a pointer to the tail
10) Printing out every element in a 2D square array using a double for loop
11) Recursive Fibonacci

1) O(1)
2) O(n)
3) O(2n) -> O(n)
4) O(n)
5) O(n)
6) O(logn)
7) O(1)
8) O(n)
9) O(1)
10) O(n^2)
11) O(2^n)


O(n!)
O(2^n)
O(n^2)
O(n log n)
O(n)
O(log n), O(1)
Operations
Elements

Sorting

Use Bubble Sort to sort the given array in ascending order. Write out the array for every iteration the sort occurs until the sorting is complete. Arr = {8,6,9,1,2,3}. DON'T Write a function just list the steps for every swap MANUALLY

Answer:

8,6,9,1,2,3

6,8,1,2,3,9

6,1,2,3,8,9

1,2,3,6,8,9

Given the array {24,31,12,17,9,15}. Implement a sorting function that would produce these arrays after each swap. State what type of sorting is used

Given array : {24,31,12,17,9,15}

1st swap : {9,31,12,17,24,15}

2nd swap : {9,12,31,17,24,15}

3rd swap : {9,12,15,17,24,31}

This is selection sort.

```
void selectionSort(int arr[], int n) {
 int min_idx;
 for (int i = 0; i < n-1; i++) {
  min_idx = i;
  for (int j = i+1; j < n; j++)
   if (arr[j] < arr[min_idx])
    min_idx = j;
  int temp = arr[i];
  arr[i] = arr[min_idx];
  arr[min_idx] = temp;
 }
}
```

**KEEP IN MIND WE COULD ASK YOU TO WRITE THE CODE FOR SELECTION, BUBBLE, OR INSERTION SORT FOR A LINKED LIST AS WELL

Use Insertion Sort to sort the given array in descending order. Write out the array for every iteration the sort occurs until the sorting is complete. Arr = {1,5,9,17,2,6}. DON'T Write a function just list the steps for every swap MANUALLY

Answer:

1,5,9,17,2,6

17,1,5,9,2,6

17,9,1,5,2,6

17,9,6,1,5,2

17,9,6,5,1,2

17,9,6,5,2,1

THAT'S IT, KEEP STUDYING!