COSC 2436
Practice Questions

By Rohan

Note

In General, there are many ways to solve a problem.

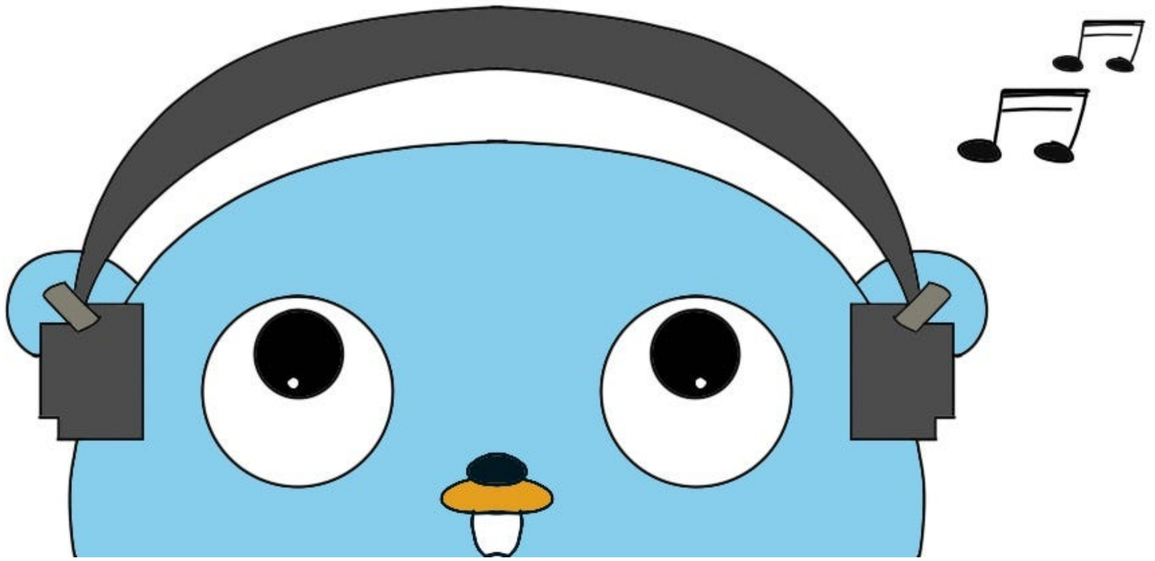
You may be asked to solve in a certain time/space complexity and in a recursive/iterative manner.

Please point out any edge cases or errors you may encounter.

The purpose of this resource is for practice, these type of questions may not be on your exam but in your future interviews.

The more you practice, the more comfortable you will become.

Linked Lists



Leetcode 1290: Convert binary number in a linked list to an integer

You will be given the head of the linked list. The value of each node is either 0 or 1.
Example:
1 -> 1 -> 1 = 7
1 -> 0 -> 1 = 5

```
struct node {
    int val;
    node *next;
};

int convertBinary(node *head) {

}
```

Solution

```
int getDecimalValue(ListNode* head) {
    int sum = 0;
    while (head) {
        sum = sum * 2 + head->val;
        head = head->next;
    }
    return sum;
}
```

There can be more than one correct solution

```
int convertBinary(node *head) {
    int length = 0;
    node *cu = head;

    while (cu != nullptr) {
        length++;
        cu = cu->next;
    }

    length--;
    cu = head;
    int num = 0;

    while (cu != nullptr) {
        if (cu->val == 1)
            num += (pow(2, length));

        length--;
        cu = cu->next;
    }
```

Leetcode 21:
Merge Two Sorted Lists

Merge the two lists into one sorted list. The list should be made by splicing together the nodes of the first two lists. Return the head of the merged linked list.

```
ListNode* addTwoNumbers(ListNode* l1, ListNode* l2)

{

}
```

Example
Input: list1 = [1,2,4], list2 = [1,3,4]
Output: [1,1,2,3,4,4]

Solution

```
ListNode* mergeTwoLists(ListNode* list1, ListNode* list2) {
    ListNode* dummy = new ListNode();
    ListNode* current = dummy;

    while (list1 != nullptr && list2 != nullptr) {
        if (list1->val <= list2->val) {
            current->next = list1;
            list1 = list1->next;
        } else {
            current->next = list2;
            list2 = list2->next;
        }
        current = current->next;
    }

    if (list1 != nullptr) {
        current->next = list1;
    }

    if (list2 != nullptr) {
        current->next = list2;
    }

    return dummy->next;
}
```

Could you do it recursively?

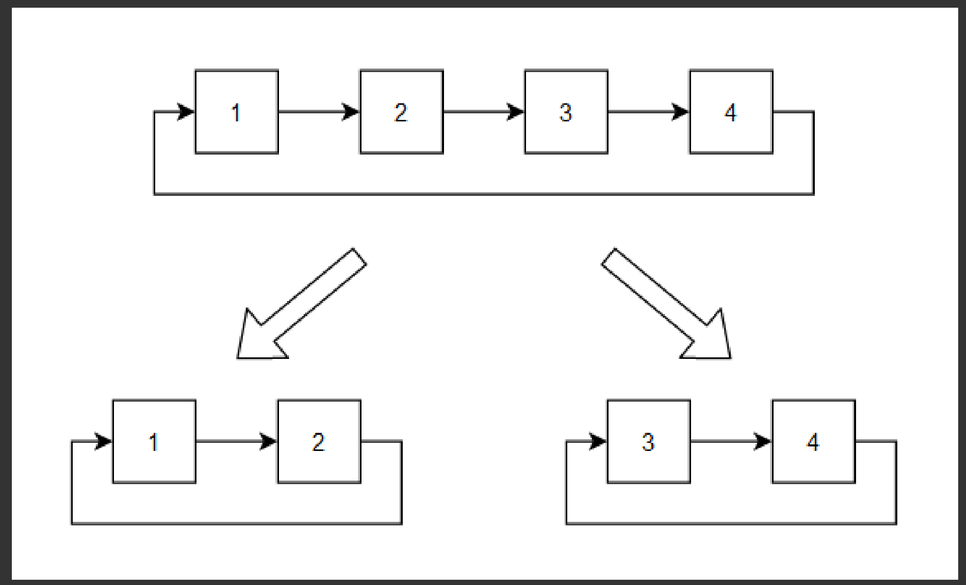
```
ListNode* mergeTwoLists(ListNode* l1, ListNode* l2) {
    if (l1 == NULL) return l2;
    if (l2 == NULL) return l1;

    if (l1->val <= l2->val) {
        l1->next = mergeTwoLists(l1->next, l2);
        return l1;
    } else {
        l2->next = mergeTwoLists(l1, l2->next);
        return l2;
    }
}
```

Geeksforgeeks
Split a Circular Linked List into two halves

If there are odd number of nodes, then first list should contain one extra.

void splitCircularList(Node* head, Node firstHalf, Node** secondHalf) {}**

Example



Solution

```
void splitCircularList(Node* head, Node** firstHalf, Node** secondHalf) {
    if (head == NULL) {
        return;
    }

    Node* slow = head;
    Node* fast = head->next;

    while (fast != head && fast->next != head) {
        fast = fast->next->next;
        slow = slow->next;
    }

    *firstHalf = head;
    *secondHalf = slow->next;
    slow->next = head;

    Node* cur = *secondHalf;

    while (cur->next != head) {
        cur = cur->next;
    }

    cur->next = *secondHalf;
}
```

Hackerrank
Delete duplicate-value nodes from a sorted linked list

SinglyLinkedListNode* removeDuplicates
(SinglyLinkedListNode* head) {}

Example
Input: 1->2->2->3->3
Output: 1->2->3

Solution

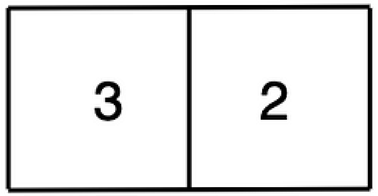
```
SinglyLinkedListNode*
removeDuplicates(SinglyLinkedListNode* head) {
    if (head == NULL || head->next == NULL) {
        return head;
    }

    SinglyLinkedListNode* current = head;

    while (current->next != NULL) {
        if (current->data == current->next->data) {
            SinglyLinkedListNode* temp = current->next;
            current->next = current->next->next;
            delete temp; // Deleting the duplicate node
        } else {
            current = current->next;
        }
    }

    return head;
}
```

What if the list was unsorted?

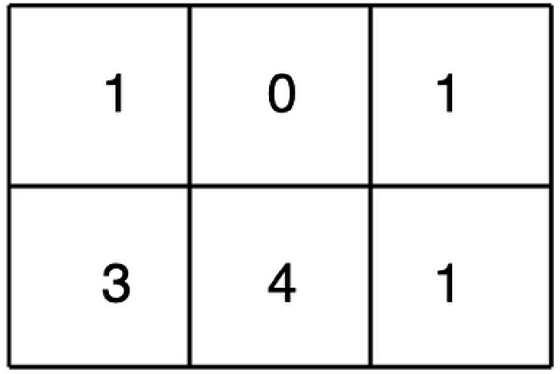
```
SinglyLinkedListNode* removeDuplicates(SinglyLinkedListNode*
head) {
    if (head == nullptr || head->next == nullptr) {
        return head;
    }

    SinglyLinkedListNode* current = head;

    while (current->next != nullptr) {
        if (current->data == current->next->data) {
            SinglyLinkedListNode* temp = current->next;
            current->next = current->next->next;
            delete temp;
        } else {
            current = current->next;
        }
    }

    return head;
}
```

Arrays

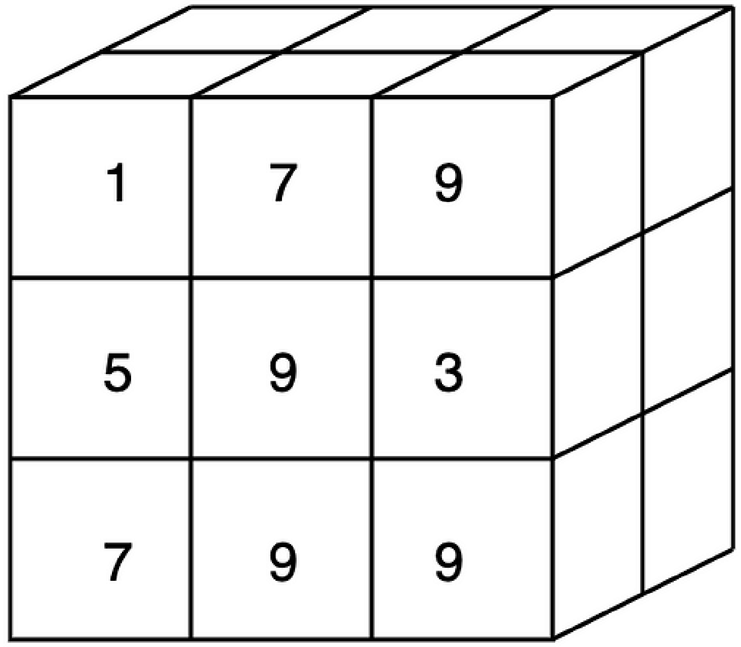
1D Array



2D Array



3D Array



Create a dynamic array of size m of the following types:

- string
- linkedList (assume linkedList is a class)
- node* (assume node is a struct)
- two dimensional array with n rows and m columns

Solution

string:

```
string *arr = new string[m];
```

linkedList:

```
linkedList *arr = new linkedList[m];
```

node*:

```
node **arr = new node*[m];
```

2d array:

```
int **matrix = new int*[n];
for(int i = 0; i < n; i++){
      matrix[i] = new int[m];
}
```

Find common elements in three sorted arrays

Given three Sorted arrays in non-decreasing order, print all common elements in these arrays.

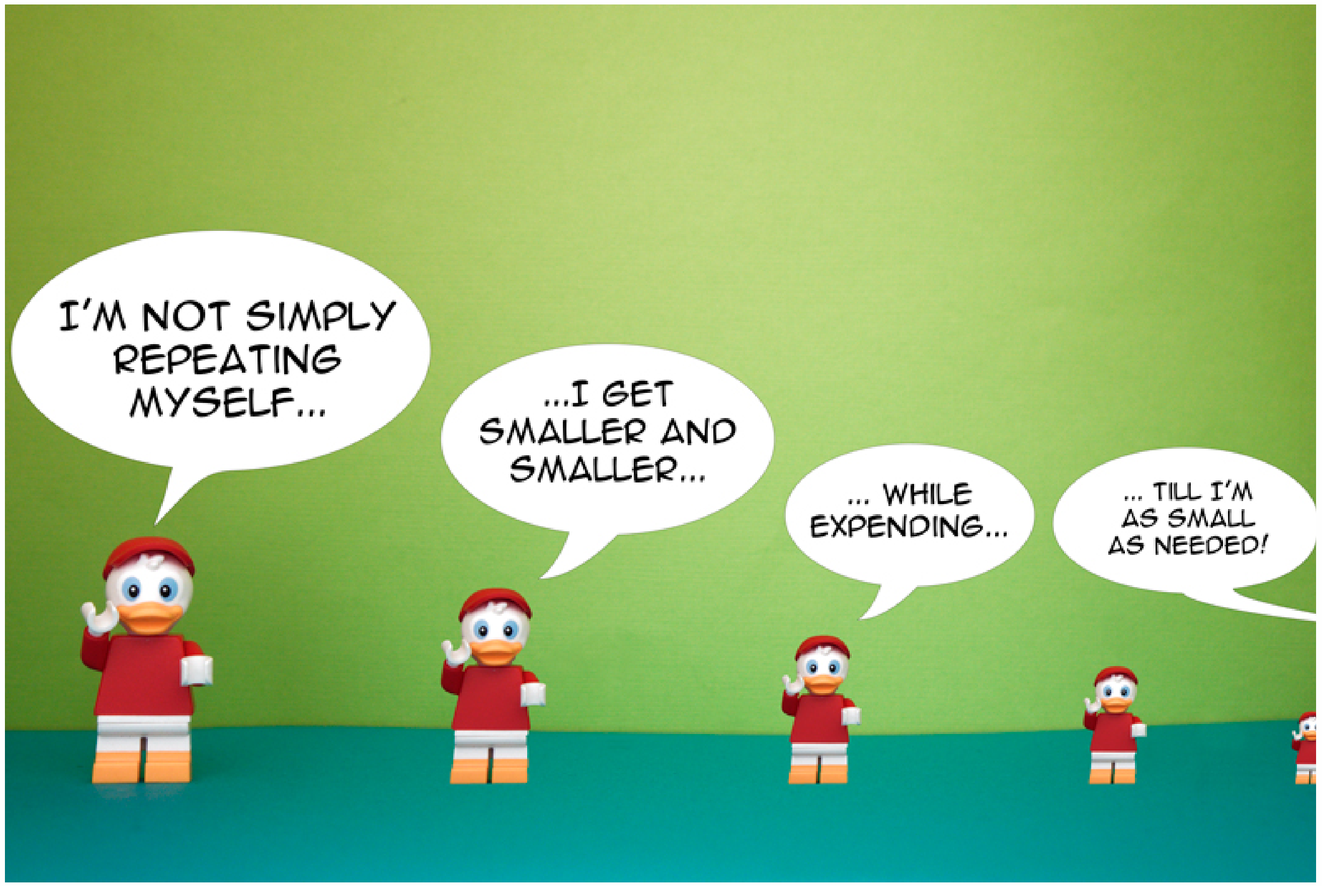
```
void findCommon(int ar1[], int ar2[], int ar3[], int n1,
                int n2, int n3){}
```

n1, n2, and n3 are size of array 1,2, and 3 respectively

Solution

```
void findCommon(int ar1[], int ar2[], int ar3[], int n1, int n2, int n3) {
   int i = 0, j = 0, k = 0;

   while (i < n1 && j < n2 && k < n3) {
      if (ar1[i] == ar2[j] && ar2[j] == ar3[k]) {
         cout << ar1[i] << " ";
         i++;
         j++;
         k++;
      } else if (ar1[i] < ar2[j]) {
         i++;
      } else if (ar2[j] < ar3[k]) {
         j++;
      } else {
         k++;
      }
   }
}
```

Recursion


I'M NOT SIMPLY REPEATING MYSELF...
...I GET SMALLER AND SMALLER...
... WHILE EXPENDING...
... TILL I'M AS SMALL AS NEEDED!

Write a recursive function that checks if a number is a pallindrome or not

bool isPalindrome(int num, int originalNum, int reverseNum) {}

Inititally we are passing 0 into reverseNum

A pallindrome is a sequence that reads the same backwards and forwards
Example
121 is a pallindrome
112 is not as it is 211 backwards

Solution

```
bool isPalindrome(int num, int originalNum, int reverseNum) {
    if (num == 0) {
        return originalNum == reverseNum;
    }

    int lastDigit = num % 10;
    reverseNum = reverseNum * 10 + lastDigit;

    return isPalindrome(num / 10, originalNum, reverseNum);
}
```

What if it was a string

Assume we are passing start as 0 and end is last index of string

```
bool isPalindrome(string word, int start, int end) {
    if (start >= end) {
        return true;
    }

    char beginning = word[start];
    char ending = word[end];

    if (beginning == ending) {
        return isPalindrome(word, start + 1, end - 1);
    } else {
        return false;
    }
}
```

Write a recursive function to find the difference of the first n natural numbers.

```
int getDiff(int n){
}
```

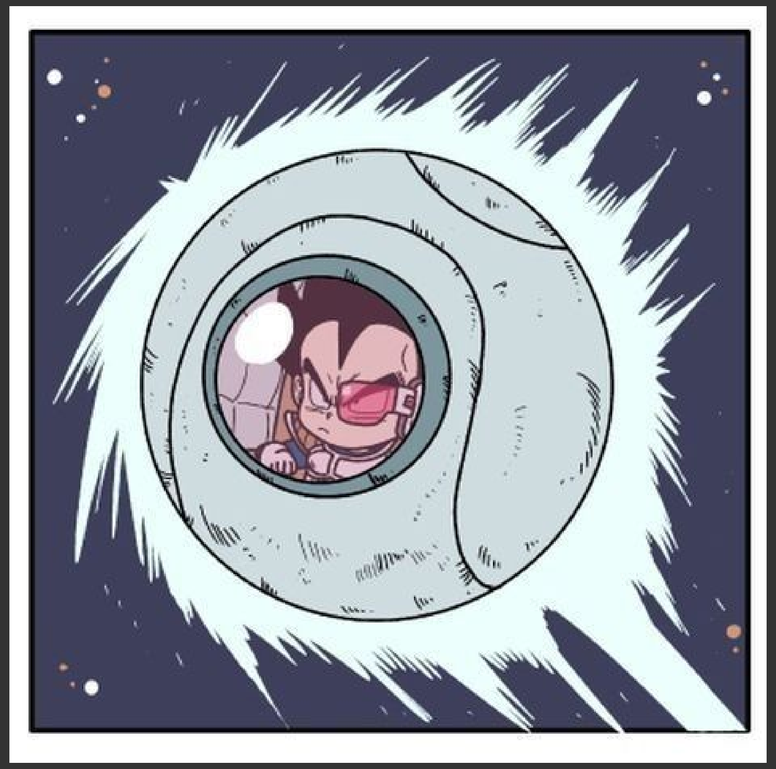
Example:
n = 5
Output: 1−2−3−4−5 = −13

Solution

```
int getDiff(int n) {
    if (n == 1) {
        return 1;
    } else {
        return getDiff(n - 1) + n;  // Change subtra
addition
    }
}
```

What is the output of fun(6)?

```
void fun(int x) {
    if (x >= 2) {
        fun(--x);
        --x;
        fun(x - 3);
        cout << x << endl;
    }
}
```



NOTE : The program is running (--x) not (x--)

Solution

0
1
2
3
4

Don’t forget a new line because of cout << endl;

What if it was (x-1)?

```
void fun(int x) {
    if (x >= 2) {
        fun(x - 1);
        --x;
        fun(x - 3);
        cout << x << endl;
    }
}
```

Solution

1
2
3
4
1
5

What is fun(5)

```
void fun(int x) {
    if (x >= 1) {
        fun(--x);
        x++;
        fun(--x);
        cout << x << endl;
    }
}
```

Solution

0
0
1
0
0
1
2

What is fun(5)

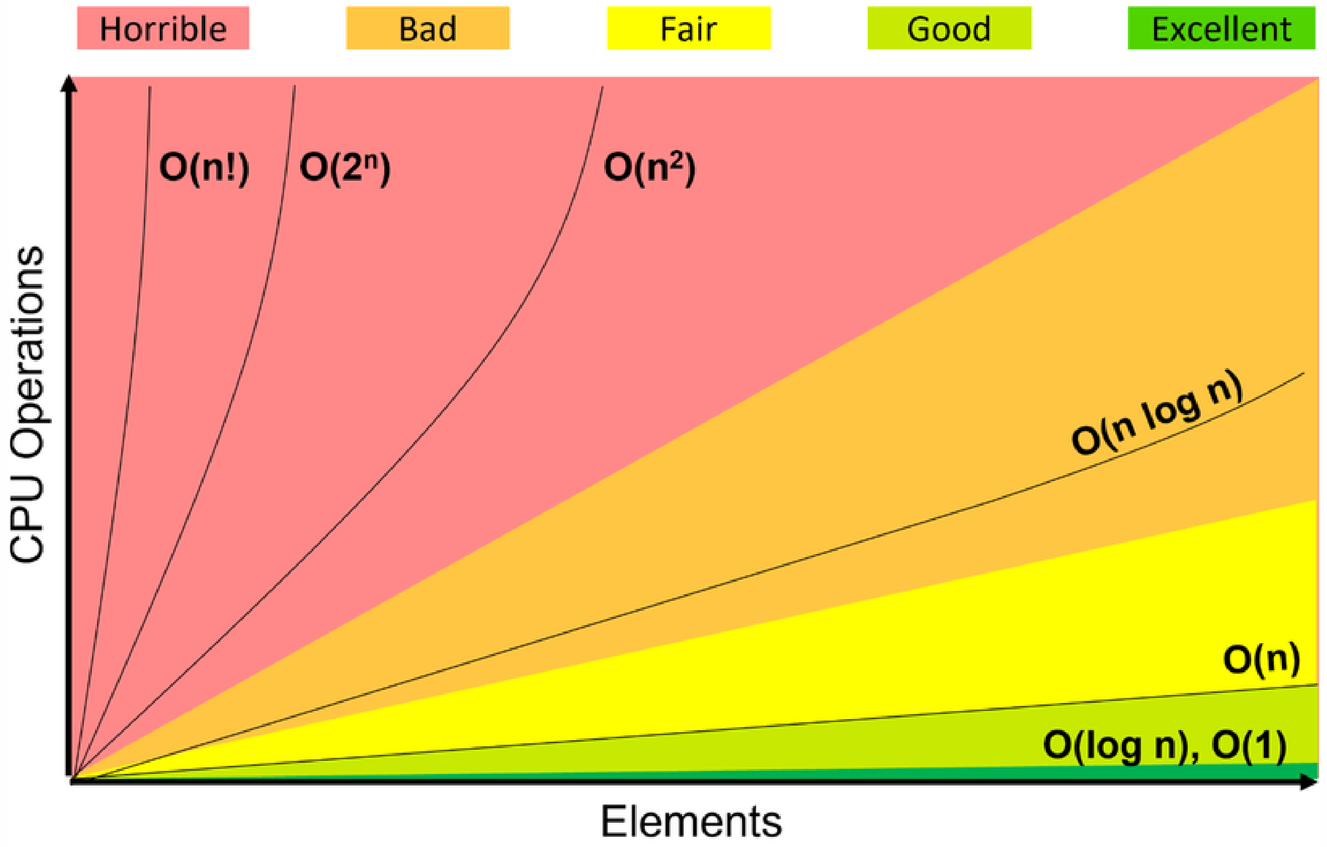
```
void fun(int x) {
    if (x <= 0) {
        cout << x << endl;
    } else if (x % 2 == 1) {
        fun(--x);
        cout << x << " ";
    } else {
        cout << x << " ";
        fun(x - 2);
    }
}
```

Solution

4 2 0
4

Don’t forget about cout << endl and cout << “ “

Time Complexity



What is time complexity of the following function ?

```
void fun(int n) {
    if (n >= 0) {
        fun(n – 2);
        cout << n – 1 << endl;
        fun(n – 2);
    }
}
```

Solution

The time complexity of this function is O(2^n)

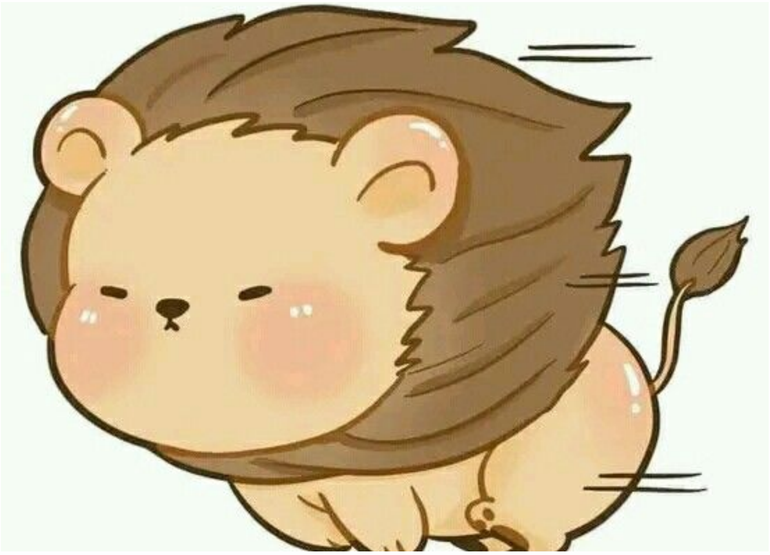
What is time complexity of the following function ?

```
int f(int n) {
    if (n == 0 || n == 1) {
        return 1;
    }
    return f(n - 1) + f(n - 2);
}
```

(Do you know which algorithm this is ?) ^_^

Solution

The time complexity of this function is O(2^n)



What is time complexity of the following function ?

```
int func(int n) {
    if (n <= 1) {
        return n;
    }
    return func(n / 2) + func(n / 2);
}
```

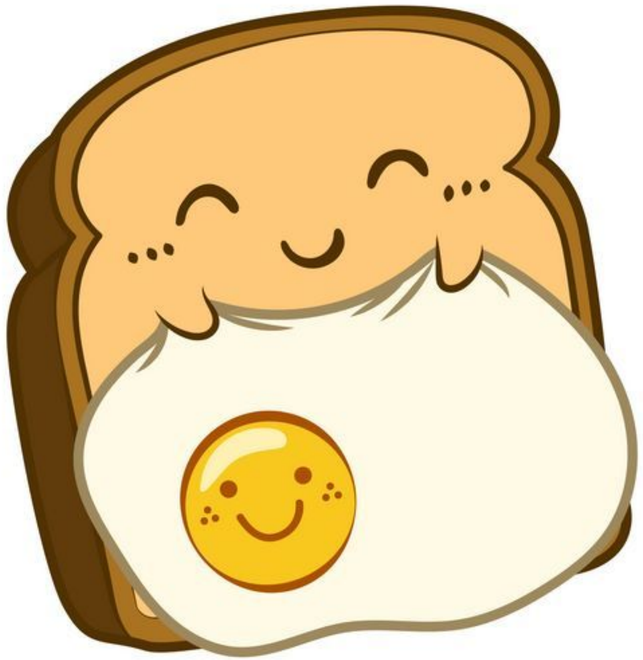
Solution

The time complexity of this function is O(2^logn)

What is time complexity of the following function ?

```
#include <iostream>

int main() {
    for (int i = 0; i < 9999999; i++) {
        for (int j = 0; j < n; j++) {
            cout << i << " " << j << endl;
        }
    }

    return 0;
}
```

Solution



The overall time complexity of the code is O(n * 9999999), which can be simplified to O(n).

Sorting
(harry potter reference below)

What are the best case and worst case time complexities of bubble, selection, and insertion sort?

Solution

Bubble Sort:
Best Case: O(n)
Worst Case: O(n2)

Selection Sort:
Best Case: O(n2)
Worst Case: O(n2)

Insertion Sort:
Best Case: O(n)
Worst Case: O(n2)

Sort the array below using insertion sort. Show the array after every time a swap occurs.

{7,5,9,2,1,3,4}

Solution

{7, 5, 9, 2, 1, 3, 4}

{5, 7, 9, 2, 1, 3, 4}

{5, 7, 9, 2, 1, 3, 4}

{2, 5, 7, 9, 1, 3, 4}

{1, 2, 5, 7, 9, 3, 4}

{1, 2, 3, 5, 7, 9, 4}

{1, 2, 3, 4, 5, 7, 9}

Sort the array below using selection sort. Show the array after
every time a swap occurs.

{7,5,9,2,1,3,4}

Solution

{7, 5, 9, 2, 1, 3, 4}

{1, 5, 9, 2, 7, 3, 4}

{1, 2, 9, 5, 7, 3, 4}

{1, 2, 3, 5, 7, 9, 4}

{1, 2, 3, 4, 7, 9, 5}

{1, 2, 3, 4, 5, 9, 7}

{1, 2, 3, 4, 5, 7, 9}

FIN.