CS 32 Week 10 Worksheet

This worksheet is entirely **optional**, and meant for extra practice. Some problems will be more challenging than others and are designed to have you apply your knowledge beyond the examples presented in lecture, discussion or projects. All exams will be done on paper, so it is in your best interest to practice these problems by hand and not rely on a compiler.

If you have any questions or concerns, please go to any of the office hours.

Concepts

Hash Tables, Heaps

For this worksheet, “using a hash table” means you’re allowed to use a STL implementation of one (unordered_map, unordered_set).

1. Given a string, find the first non-repeating character in it and return its index. If it doesn't exist, return -1. You may assume the string contains only lowercase letters. Use a hash table to solve this problem.

   Examples:

   Input: s = "leetcode"
   Output: 0

   Input: s = "loveleetcode"
   Output: 2

   ```
   int firstUniqueChar(std::string s) {
        // Fill in code
   }
   ```

2. Given an array of integers and a target sum, determine if there exists two integers in the array that can be added together to equal the sum.

   The time complexity of your solution should be O(N), where N is the number of elements in the array. In other words, the brute force method of comparing each element with every other element using nested for loops will not satisfy this requirement.

Examples:

Input: arr[] = [4, 8, 3, 7, 9, 2, 5], target = 15
Output: true
Explanation: 8 and 7 add up to the target sum 15

Input: arr[] = [1, 3, 5, 2, 4], target = 10
Output: false
Explanation: No combination of two numbers in the array sum to 10

```
bool twoSum(int arr[], int n, int target);
```

3. Given a vector of strings, group anagrams together. Two words are anagrams of each other if one word can be formed by rearranging the letters of the other. For example, *cinema* and *iceman* are anagrams because *cinema* can be rearranged to form the word *iceman*.

   Hint: Solve this problem using the following hash function. As stated in the comment, all words with the same characters will have the same hash value; think about why this makes it a less than ideal hash function, and, how that can help you in this particular problem.

```
// Given a string, compute its hash value based on prime numbers
int calculateHash(string word) {
           int primes[26] = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29,
      31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97,
      101};
         int hash_val = 1;
         for(int i = 0; i < word.size(); i++) {
            // Multiplying by prime numbers ensures that all words
            // with the same characters will have the same product
             hash_val *= primes[word[i]-'a'];
         }
         return hash_val;
}
```

   For example, given: ["eat", "tea", "tan", "ate", "nat", "bat"],

   Return:
   [
    ["ate", "eat","tea"],
    ["nat","tan"],
    ["bat"]

]

You may assume only lower case letters will be used.

```
vector<vector<string>> groupAnagrams(vector<string> strs) {
      // Fill in code here
}
```

4. Implement the following function:

```
bool isMaxHeap(int* arr, int len);
```

This function takes in an array *arr* of length *len* and returns whether or not that array represents a binary max heap. In other words, *arr* must follow the max heap property, where a parent is greater than or equal to its children.

5. Implement the following function, given the following data structure:

```
struct Node {
   int val;
   Node* left;
   Node* right;
};

bool isMinHeap(Node* head);
```

This function takes in the head of a binary tree and returns whether or not that binary tree represents a binary min heap. In other words, this tree must follow the min heap property, where a parent is less than or equal to its children. It must also follow the completeness property, where every level of the tree except possibly for the lowest is completely filled, and the lowest level's nodes must be as far left as possible.

6. You are working at a credit card company and need to store account balances. Each account holder has an integer userId number. Each user/userid can have as many bank accounts as they want, each specified by an integer accountId.
Write a class called `bank` that supports insertion of a deposit and search of a given user and account id. Insert should update the balance of an account, if the given `accountId` and `userId` already exists. If it does not exist, it should create a new entry with the given parameters.

Search should return the balance of the account, if the given `accountId` and `userId` exists. If it does not exist, the function must return `-1`.
The company wants to process a high volume of transactions so they demand search and insertion work in O(1) time, i.e they do not depend on the number of users or bank accounts.
Hint: Consider an STL container in an STL container

```
class Bank {
public:
void insert (int amount, int userId, int accountId);
int search (int userId, int accountId);
…
}

i.e
Bank B;
B.insert(10, 765, 937)
B.search(765, 937) // returns 10
```

Extra Practice Problems

7. Given an array of *n* integers that is guaranteed to satisfy the max heap property, write a function that constructs a binary tree representing the same binary max heap as the array and returns its root.

```
struct Node {
  int val;
  Node* left;
  Node* right;
};

Node* makeMaxHeap(int a[], int n);
```

8. Given an array of distinct elements and a range [low, high], use a hash table to output all numbers in the range that are not in the array. Print the missing elements in sorted order.

```
Examples:

Input:     arr[] = {10, 12, 11, 15}, low = 10, high = 15
Output:    13, 14

Input:     arr[] = {1, 14, 11, 51, 55}, low = 50, high = 55
```

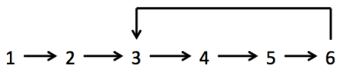
```
Output:    50, 52, 53, 54

void inRange(int arr[], int size, int low, int high);
```

9. Write a function, sum3, that takes in an array of integers and determines whether there exists exactly three elements in the array that sum to 0. Return true if three such elements exist and false if not. No repeated elements are allowed. Your function must run faster than the brute force $O(N^3)$.

   i.e [1,2,3,4,5,6] -> False
   [1,-1,2,-2] -> False
   [1,2,-3, 6, 8] -> True

```
bool sum3(int[] arr);
```

10. Given a linked list, determine if it has a cycle in it. This can be done by starting from the head of the linked list and traversing it until you reach a node you have already seen or the end of the list. The time complexity of your solution should be O(n) where n is the number of nodes in the list. Example:



This list has a cycle - Node 6 points back to Node 3

Use the following Node definition and function header to get started:

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

bool hasCycle(Node* head);
```