CS 32 Worksheet 6

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 contact mcmallett@ucla.edu or sarahelizabethcastillo@gmail.com or go to any of the LA office hours.

Concepts

Hash Tables, Sorting, Heaps

1. Here are the elements of an array after each of the first few passes of a sorting algorithm discussed in class. Which sorting algorithm is it? (F.Y.)

   3 7 4 9 5 2 6 1

   3 7 4 9 5 2 6 1

   3 **7** 4 9 5 2 6 1

   3 **4** 7 9 5 2 6 1

   3 4 7 **9** 5 2 6 1

   3 4 **5** 7 9 2 6 1

   2 3 4 5 7 9 6 1

   2 3 4 5 **6** 7 9 1

   1 2 3 4 5 6 7 9

   a. bubble sort
   b. insertion sort
   c. quicksort with the pivot always being chosen as the first element
   d. quicksort with the pivot always being chosen as the last element

2. Given an array arr[0..n-1] of distinct elements and a range [low, high], use a hash table to find all numbers that are in the range, but not in the array. Print out the missing elements in sorted order. (F.Y.)

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

   Input:     arr[] = {1, 14, 11, 51, 15}, low = 50, high = 55
   ```

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

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

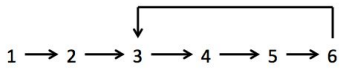
3. Given the following vectors of integers and sorting algorithms, write down what the vector will look like after 3 iterations or steps and whether it has been perfectly sorted.

   a. `{45, 3, 21, 6, 8, 10, 12, 15}` insertion sort
   b. `{5, 1, 2, 4, 8}` bubble sort
   c. `{-4, 19, 8, 2, -44, 3, 1, 0}` quicksort (where pivot is always the last element)

4. Given an array of *n* integers, where each integer is guaranteed to be between 1 and 100 (inclusive) and duplicates are allowed, write a function to sort the array in O(*n*) time.
   (Hint: the key to getting a sort faster than O(*n log n*) is to avoid directly comparing elements of the array!) (MV)

```
void sort(int a[], int n);
```

5. 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(const Node* head);
```

6. Given an array of integers and a target sum, determine if two integers in the array 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, you cannot use the brute force method in which you compare each element with every other element using nested for loops. Example:

Array: 4 8 3 7 9 2 5
Target: 15

You can take 8 and 7 from the array, and their sum equals the target of 15. Thus, the function we will write will return true. Use the following function header to get started:

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

7. Given an vector of strings, group anagrams together. An anagram is a word formed by rearranging the letters of another, such as *cinema* formed by *iceman*. Return a vector of vectors of strings. Solve this problem using a hash table. You may assume only lower case letters.

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

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

8. 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 contain only lowercase letters. Use a hashtable to solve this problem.
   Examples:
   s = "leetcode"
   return 0
   s = "loveleetcode",
   return 2

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

9. Implement the following function:

```
bool isMaxHeap(const 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.

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

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

bool isMinHeap(const 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.

11. Given an array of *n* integers that is guaranteed to satisfy the max heap property, write a function that returns a pointer to a binary tree representing the same binary max heap as the array. (MV)

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

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

12. Write a function, sum3, that given an array of integers finds if some three elements of the array sum to 0. Return true if three such elements exist and false if not. 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(const int[] arr, int n);
```

13. 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 specified by an integer accountid. Write a class that supports insertion of a deposit and search of a given user and account id. Insert should update the amount if the given `accountId` and `userId` already exists. 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
```

14. Write a function `travelItinerary` which takes in a list of tickets and prints the corresponding travel itinerary. There is only one ticket from every city, besides the final destination and the list of tickets is not cyclic. The maximum time complexity of your algorithm should be O(n). Hint: you will need to use two hash tables.

```
void travelItinerary(map<string, string> tickets);

Input:
Bali → Tokyo
London → Bangkok
Bangkok → Dubai
Dubai → Bali

Output:
London → Bangkok
```

Bangkok → Dubai
Dubai → Bali
Bali → Tokyo