Exercise 2: heap and hashmap

TA: Ren, Peng

Date: 11/08/2023

Question 1:

1. **Code Completion**

   The C code snippet below is designed to perform a heapify operation. Complete the missing parts of the code.

```c
#include <stdio.h>
#include <stdlib.h>
void down_hill(int* a, int size, int root) {
    while (1) {
        int children = root * 2 + 1;
        if (children >= size) {
            return;
        }
        if (children < size - 1 && a[children] > a[children + 1]) {
            children++;
        }
        if (a[root] > a[children]) {
            // Swap the elements
            // update root
        } else {
            return;
        }
    }
}

void heapify(int* a, int size) {
    for (int i = size / 2 - 1; i >= 0; i--) {
        down_hill(a, size, i);
    }
}
```

2. **Heap Result**

   Considering the provided `heapify` function, what will be the resulting heap after its execution? Indicate whether it will be a max-heap or a min-heap.

3. **Heapify Function Indexing**

In the `heapify` function, the index `i` starts at `size/2 - 1`. Could `i` instead start at `size - 1`? Explain your reasoning.

Question 2:

1. **Debugging Two Sum with Hash Table**

   Examine the following C code that attempts to solve the "Two Sum" problem using a pre-filled hash table. The code is intended to find two distinct elements in an array `nums` that add up to a given `target` sum.

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

// Assume these hash table functions are available
void insert(int key, int value); // Inserts a key-value pair into the hash table
int search(int key); // Searches for the key, returns the value if found, -1 otherwise

// Function to put every element into the hash table
void fillHashTable(int* nums, int numsSize) {
    for (int i = 0; i < numsSize; i++) {
        insert(nums[i], i);
    }
}

// Function to find two distinct elements that add up to target
int* findTwoSum(int* nums, int numsSize, int target) {
    for (int i = 0; i < numsSize; i++) {
        int complementIndex = search(target - nums[i]);
        if (complementIndex != -1) {
            int* result = (int*)malloc(2 * sizeof(int));
            result[0] = i;
            result[1] = complementIndex;
            return result; // Found
        }
    }
    return NULL; // Not found
}

int main() {
    int nums[] = {2, 7, 11, 15};
    int target = 9;

    // Fill the hash table with array elements
    fillHashTable(nums, sizeof(nums) / sizeof(nums[0]));

    // Find two sum
    int* indices = findTwoSum(nums, sizeof(nums) / sizeof(nums[0]), target);
    if (indices != NULL) {
        printf("Indices: %d, %d\n", indices[0], indices[1]);
        free(indices); // Remember to free the memory allocated for indices
    } else {
        printf("No two sum solution found.\n");
    }

    return 0;
}
```

After reviewing the code, explain why the implementation may not always work correctly.

Hint: Pay special attention to the `findTwoSum` function.

2. **Time complexity**

   What is the time complexity of the `findTwoSum` function provided above?

3. **Follow-up Question:**

   If the array `nums` is sorted, can you solve the problem in O(N) time complexity without using a hashmap?

   Hint: Use two pointers.