CS 32 Week 8 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

Algorithm Analysis, Sorting

1. What is the time complexity of the following code?

```
int randomSum(int n) {
  int sum = 0;
  for(int i = 0; i < n; i++) {
    for(int j = 0; j < i; j++) {
      if(rand() % 2 == 1) {
        sum += 1;
      }
      for(int k = 0; k < j*i; k+=j) {
        if(rand() % 2 == 2) {
          sum += 1;
        }
      }
    }
  }
  return sum;
}
```

2. FInd the time complexity of the following code:

```
int operationFoo(int n, int m, int w) {
      int res = 0;

      for (int i = 0; i < n; ++i) {
            for (int j = m; j > 0; j /= 2) {
                  for (int jj = 0; jj < 50; jj++) {
                        for (int k = w; k > 0; k -= 3) {
                              res += i*j + k;
```

```
                        }
                    }
                }
            }
            return res;
        }
```

3. Find the time complexity of the following function

```
int obfuscate(int a, int b) {
        vector<int> v;
        set<int> s;
        for (int i = 0; i < a; i++) {
                v.push_back(i);
                s.insert(i);
        }
        v.clear();

        int total = 0;
        if (!s.empty()) {
                for (int x = a; x < b; x++) {
                        for (int y = b; y > 0; y--) {
                                total += (x + y);
                        }
                }
        }
        return v.size() + s.size() + total;
}
```

4. 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?

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

5. 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 (1st step starts at comparing a[1])
    b. `{5, 1, 2, 4, 8}` bubble sort (Consider the array after 3 "passes" and after 3 "swaps." Do the results differ? Does the algorithm know when it's "done" in either case?)
    c. `{-4, 19, 8, 2, -44, 3, 1, 0}` quicksort (where pivot is always the last element)

6. Consider this function that returns whether or not an integer is a prime number:

```
bool isPrime(int n) {
  if (n < 2 || n % 2 == 0) return false;
  if (n == 2) return true;
  for (int i = 3; (i * i) <= n; i += 2) {
    if (n % i == 0) return false;
  }
  return true;
}
```

    What is its time complexity?

7. Fill out the following table:

Time complexity	Doubly linked list (<u>given head, unless noted otherwise</u>)	Array/vector
Inserting an element to the beginning		

Inserting an element to some position i		
Getting the value of an element at position i		
Changing the value of an element at position i		
Deleting an element given a reference to it		

8. Write a function for which, given a vector of words and a character, returns the number of times that character is present in the entire vector. Then, find the time complexity of your algorithm.

```
int countNumOccurrences(const vector<string>& words, char c);
```

Note: When calculating the time complexity, you can consider the size of the vector as N and the average length of one word is K.

9. 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);
```