CS 32 Worksheet 7

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.

Solutions are written in red. The solutions for **programming** problems are not absolute, it is okay if your code looks different; this is just one way to solve the specific problem.

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

Concepts

Algorithm Analysis, Sorting

1. 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?

   O($\sqrt{n}$). The function's loop only runs while i <= $\sqrt{n}$, and the square root is not the same as a constant multiple of n, so we include it in our Big-O analysis.

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

```
int randomSum(int n) {
  int sum = 0;
  for(int i = 0; i < n; i++) { 1^2 + 2^2 + … + (n-1)^2 is
O(n^3)
    for(int j = 0; j < i; j++) { O(i^2)
```

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

O(n^3), note it doesn't matter that "if(rand() %2 == 2)" is always false, rand is still run each time

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;
}
```

O(a log a + b^2)

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

Notice that the section to the left of the underlined digit is in sorted order. One of the most prominent insertion sort characteristics is having a section that is already in sorted order and continue to put the current element in the right place.

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 (first step is at a[1])
b. `{5, 1, 2, 4, 8}` bubble sort (consider the array after 3 "passes" and after 3 "swaps")
c. `{-4, 19, 8, 2, -44, 3, 1, 0}` quicksort (where pivot is always the last element)

a. {3, 6, 21, 45, 8, 10, 12, 15} (assume first step starts by sorting a[1] since a[0] is trivial) not perfectly sorted

b. {1, 2, 4, 5, 8} perfectly sorted, within 3 "steps" it does not know it's complete, but within 2 "passes" it will

c. 1st iteration -> {-4, -44, **0**, 2, 19, 3, 1, 8}
-4 and -44 might be in a different order
2, 19, 3, 1, 8 might be in a different order
2nd iteration, if starting from the order shown after the

```
    1st iteration, and left part is sorted before right
    part -> {-44, -4, 0, 2, 19, 3, 1, 8}
3rd iteration, if starting from the order shown after the
    2nd iteration, and left part is sorted before right
    part -> {-44, -4, 0, 2, 1, 3, 8, 19}
    2, 1, 3 might be in a different order
Not perfectly sorted
Next recurses into [2,1,3] and [19]
```

6. 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);
```

```
void sort(int a[], int n) {
  int counts[100] = {};  // Count occurrences of each integer.
  for (int i = 0; i < n; i++)
    counts[a[i] - 1]++;

  // Add that many of each integer to the array in order.
  int j = 0;
  for (int i = 0; i < 100; i++)
    for (; counts[i] > 0; counts[i]--)
      a[j++] = i + 1;
}
```

7. Fill out the following table:

Time complexity	Doubly linked list (given head)	Array/vector
Inserting an element (to the head)	O(1)	N/A
Inserting an element to some position i	O(n)	O(n)

Getting the value of an element at position i	O(n)	O(1)
Changing the value of an element at position i	O(n)	O(1)
Deleting an element given a reference to it	O(1)	O(n)

8. What is the time complexity of the following code?
// assuming vector v is of size N, and head is the head of a linked list of size M

```
void foo(vector<int> v, Node* head) {
   if (!v.empty() && v[0] == 0)
      v.erase(v.begin());  // O(N) -- but this is lower order than the later part

   Node* current = head;
   while (current != NULL) {  // O(M*N) total
      for (vector<int>::iterator itr = v.begin(); itr != v.end(); itr++) {  // O(N)
         if (*itr == current->val) {
            *itr = 0;
            break;  // you can ignore this because in worst case, we never hit this break
         }
      }
      current = current->next;
   }
}
```

Answer: O(N*M)