CS 32 Week 7 Worksheet

Concepts: Algorithm Analysis, Sorting

Algorithmic Analysis Problems

1. (5 min) What's the time complexity of the following function? 🔍

```
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. (5 min) Nice! 😄 Now, what's the time complexity of this function?

```
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. Binary search 🤓 is an efficient algorithm finding if an element (x) exists in a sorted array. What's the time complexity of the following function? *Hint: try tracing through to find the mechanism of this algorithm, then consider what will be the worst case!*

```
int binarySearch(int arr[], int left, int right, int x)
{
    while (left <= right) {
        int middle = left + (right - left) / 2;

        if (arr[middle] == x)
            return middle;
        else if (arr[middle] < x)
            left = middle + 1;
        else
            right = middle - 1;
    }
    return -1;
}

int main()
{
    int arr[] = {2, 3, 4, 10, 40, 60, 80};
    int x = 60;
    int index = binarySearch(arr, 0, 6, x);
    if (index == -1) {
        cout << x << " doesn't exist in array." << endl;
    } else {
        cout << x << " is at " << index << " position." << endl;
    }
}
```

4. Just a few more to go! 😁 What's the time complexity of the following code?

```
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;
}
```

5. So far, so good! 😅 How about this function, what's its time complexity?

```
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;
}
```

6. Great job! Now, let's switch things up... 😏 For each operation (row), fill out the time complexities of performing that action using the given data structure (col).

Time complexity	**Doubly linked list (given head)**	**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		

7. Neat! Let's try some code now. 😎 Write a function which, given a vector of words and a character, returns the number of times that character is present in the vector.

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

Then, find the time complexity of your algorithm! *Note: When calculating the time complexity, let the size of the vector be N and the average length of one word be K.*

Sorting Problems

8. Last code writing question! 🙏 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!*

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

9. Cooldown with some MCQ! 🥰 Here are the elements of an array after each of the first few passes of a sorting algorithm. Which of the four sorting algorithms 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

10. And voila, last question! 🥳 For each of the following cases: given the vectors of integers and sorting algorithm, write down what the vector will look like after 3 iterations or steps and state whether the vector has been perfectly sorted.

A. **Applying Insertion Sort** on: `{45, 3, 21, 6, 8, 10, 12, 15}`
   (Note: 1st step starts at comparing `a[1]`)

B. **Applying Bubble Sort** on: `{5, 1, 2, 4, 8}`
   (Note: 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. **Applying Quicksort** on: `{-4, 19, 8, 2, -44, 3, 1, 0}`
   (Note: in this case, the pivot is always the last element)

And that's a wrap! You rocked it — another amazing week :)

From all of us who've come and gone through COM SCI 32 — Worksheet 7, we're more proud of you than you could ever imagine 💕

Celebrating a week of growth, *this one's for you* 🥂