CS106B Practice Midterm Spring 2023

This is a closed note, closed-book exam. You are allowed **one back-and-front page of notes, and the reference sheet posted on the course website.** You may not use any laptops, cell phones, or internet devices of any sort, unless you are taking the exam on a laptop, which must only be used for the exam. You will be graded on functionality—but good style helps graders understand what you were attempting. You do not need to #include any libraries and you do not need to forward declare any functions. You have 2 hours. We hope this exam is an exciting journey.

Last Name: _____

First Name: _____

Sunet ID (eg jdoe): _____

I accept the letter and spirit of the honor code. I've neither given nor received aid on this exam. I pledge to write more neatly than I ever have in my entire life.

(signed) _____

Problem	Points
Algorithm Analysis and Big-O	10
Collections	12
Recursion Tracing	14
Histogram	15
Short Recursive Functions	15
Total	66

Question 1: Algorithm Analysis and Big O (10 Points)

Give a tight bound of the worst-case runtime complexity class for each of the following code fragments in Big-Oh notation, in terms of variable *N*. (Write the growth rate as *N* grows.) Write a simple expression that gives only a power of *N*, such as $O(N^2)$ or O(log *N*), *not* an exact calculation like $O(2N^3 + 4N + 14)$. Write your answer in the blanks on the right side.

Question	Answer

a)
`int sum = 0;`
`for (int i=0; i < N; i++) {`
`  for (int j=N; j > 0; j--) {`
`  for (int k = 0; k < N; k++) {`
`  for (int w = 0; w < 1000; w++) {`
`  sum += N;`
`  }`
`  }`
`  }`
`}`
`for (int i=0; i < 2 * N; i++) {`
`  sum--;`
`}`
	O( )

b)
`Map<int,int> map;`
`for (int i=0; i < N; i++) {`
`  map[i] = i * 10;`
`}`
	O( )

c)
`Vector<int> v;`
`for (int i=1; i < N; i = i * 2) {`
`  v.add(i + 42);`
`}`
	O( )

d)
`int sum = 0;`
`for (int i=0; i < N; i++) {`
`  sum++;`
`}`
`for (int j=N; j >= 0; j--) {`
`  sum--;`
`}`
`for (int k=1; k < N; k++) {`
`  sum++;`
`}`
	O( )

e)
`int result = 1;`
`  int x = 2;`
`  int y = 3;`
`  for (int i=0; i < 100; i++) {`
`  for (int j=0; j < N; j++) {`
`  for (int k=500; k >= 0; k--) {`
`  for (int p = N; p > 0; p--) {`
`  result += x * y + j - k + i;`
`  }`
`  }`
`  }`
`}`
	O( )

Question 2: Stacks and / or Queues (12 Points)

Write a function to reverse the first *k* elements in a Queue<int>. To do so, you may use ***only one*** additional data structure **other than a Vector** to temporarily hold values (e.g., Stack, Set, Map, Queue):

```
void reverseFirstKFromQueue(Queue<int> &queue, int k);
```

Here are some examples (the front of the queue is on the *left*):

Initial queue: {10, 10, 30, 40, 50, 60, 70, 80, 90, 100}
After `reverseFirstKFromQueue(queue, 6);`
{60, 50, 40, 30, 10, 10, 70, 80, 90, 100}

Initial queue: {1, 2, 3, 4, 5, 6, 6, 8, 9, 10}
After `reverseFirstKFromQueue(queue, 2);`
{2, 1, 3, 4, 5, 6, 6, 8, 9, 10}

Initial queue: {10, 20, 30, 40, 50, 60, 70, 80, 90, 100}
After `reverseFirstKFromQueue(queue, 10);`
{100, 90, 80, 70, 60, 50, 40, 30, 20, 10}

Values of *k* that are zero or negative, or greater than the number of elements in the queue should not change the queue. For example:

Initial queue: {10, 20, 30, 40, 50, 60, 70, 80, 90, 100}
After `reverseFirstKFromQueue(queue, 0);`
{10, 20, 30, 40, 50, 60, 70, 80, 90, 100}
After `reverseFirstKFromQueue(queue, -1);`
{10, 20, 30, 40, 50, 60, 70, 80, 90, 100}
After `reverseFirstKFromQueue(queue, 12);`
{10, 20, 30, 40, 50, 60, 70, 80, 90, 100}

Notes:

1. You are only allowed to use *only one* additional data structure to solve the problem. Using more than one additional data structure will incur a deduction of 50% of the points for the problem.
2. If you can only think of a way to do the problem using a Vector, you may, but it will be a four point deduction.
3. You do not have to include any headers in your solution.

Please put your answer to question 2 here:

```
void reverseFirstKFromQueue(Queue<int> &queue, int k) {
```

Question 3: Recursion Tracing (14 points)

Part A: For each of the calls to `recursionMystery1` below, indicate the return value:

```
int recursionMystery1(int a, int b) {
   if (a < b) return a;
   return recursionMystery1(a-b,b);
}
```

Call	Return Value
a) `recursionMystery1(10,4);`	
b) `recursionMystery1(3,7);`	
c) `recursionMystery1(15,5);`	

Part B: What mathematical function does `recursionMystery1` calculate?

Part C: For each of the calls to `recursionMystery2` below, indicate what is printed out from the function:

```
void recursionMystery2(int a, char c) {
    if (a != 0) {
        cout << c;
        if (a % 2 == 0) {
            cout << "O";
            recursionMystery2(a - 1, c - 1);
        } else {
            recursionMystery2(a - 1, c + 1);
            cout << "R";
        }
    } else {
        cout << endl;
    }
}
```

Note: `char` characters are in increasing order, e.g., `'a' + 1 == 'b'`.

Call	What is printed to the screen
a) `recursionMystery2(5,'f');`	
b) `recursionMystery2(4,'i');`	

Question 4: Histogram (15 points)

A common way to visualize how a class of students perform on exams is by using a histogram, which provides an estimate of the probability distribution of the grades for the exam. For example, given the following scores on an exam, we can draw the histogram (shown to the right), which represents how many students received grades in the 60s, 70s, 80s, and 90s.

Student	Grade
StudentA	97
studentB	89
studentC	93
studentD	75
studentE	94
studentF	85
studentG	88
studentH	68
studentI	79
studentJ	84

```
Histogram: of Averages
60s: *
70s: **
80s: ****
90s: ***
```

A histogram can also be used to determine the distribution of grades for an entire quarter, based on an average of each student's grades.

Consider the following map which associates student names with a vector of their grades for the quarter. We would like to produce a histogram of student averages. In other words, average each student's grades, then produce a histogram of the averages.

The histogram for the averages is shown to the right:

Student	Grade	Average
studentA	97, 92, 88	92.3
studentB	89, 93, 77	86.3
studentC	93, 95, 105	97.7
studentD	75, 25, 50	50
studentE	94, 94, 94	94
studentF	85, 82, 73	80
studentG	88, 91, 99	92.7
studentH	68, 78, 88	78
studentI	79, 85, 77	80.3
studentJ	84, 85, 86	85

```
Histogram: of Averages
50s: *
70s: *
80s: ****
90s: ****
```

Given a map of student names (strings) as keys, and a vector (ints) to each student's scores, your job is to write the following three functions:

[4 points]

```
// Returns the average value of a vector of integers.

// Assumes there is at least one grade.

double average(Vector<int> & gradeVec)
```

[7 points]

```
// Produce a map of average grade distributions, grouped by
// tens (e.g., if 8 people scored an average in the 90s, there
// would be a key in the map for 90, and its value would be 8)
void histogram(Map<string,Vector<int>> & grades,
               Map<int,int> & hist)
```

[4 points]

```
// Print a histogram in the following form:
// 50s:***
// 60s:*****
// 70s:**
// 80s:***
// 90s:******
//
// For the example above, the map holds the
// following key/value pairs: {50:3, 60:5, 70:2, 80:3, 90:6}
// Assume that keys and values are positive and that keys are
// multiples of ten.
void printHistogram(Map & hist)
```

Question 5: Short Recursive Functions (15 points)

- **Count 11:** Given a string, compute recursively (no loops) the number of "11" substrings in the string. The "11" substrings should not overlap. Examples:
  - `count11("11abc11") → 2`
  - `count11("abc11x11x11") → 3`
  - `count11("111") → 1`

```
int count11(string s) {
    if (str.length() <= 1) {
        return 0;
    }
    if (str[0] == '1' && str[1] == '1') {
        return 1 + count11(str.substring(2));
    }
    return count11(str.substring(1));
}
```

- **ChangeXY:** Given a string, compute recursively (no loops) a new string where all the lowercase `'x'` chars have been changed to `'y'` chars. Examples:
  - `changeXY("codex") → "codey"`
  - `changeXY("xxhixx") → "yyhiyy"`
  - `changeXY("xhixhix") → "yhiyhiy"`

```
string changeXY(string s) {
```

- **Permutations with no adjacent duplicates:** Given a string, print out only the permutations of the string where adjacent characters are not the same. Your solution should print duplicate permutations, where expected. You will need a helper function to do this recursively. Examples:
  - `permuteNoDups("aabc") → abac abca acab acba abac abca acab acba baca baca caba caba`
  - `permuteNoDups("xxy") → xyx xyx`
  - `permuteNoDups("xxxxyy") → (no output)`

```
void permuteNoDupsHelper(                              ) {










}

void permuteNoDups(string s) {

}
```