Eric Roberts
CS 106B

Handout #24
February 2, 2015

Problem Set #1

Due: Friday, February 6 by 5:00 P.M. Problem sets are pencil-and-paper exercises and may be submitted in section, in lecture, or to the box outside my office (Gates 202).

Problem 1: Computational complexity

1a. Assume that the function `enigma` has been defined as follows:

```
int enigma(int n) {
   if (n == 0) {
      return 0;
   } else {
      return enigma(n - 1) + 1 + enigma(n - 1);
   }
}
```

i. What is the value of `enigma(5)`?

ii. What is the computational complexity of the `enigma` function expressed in terms of big-O notation, where *N* is the value of the argument `n`, which you may assume is nonnegative.

iii. How does the computational complexity of the `enigma` function change if you replace the recursive case with the following, mathematically equivalent statement:

```
return 2 * enigma(n - 1) + 1;
```

1b. Assume that the function `mystery` has been defined as follows:

```
string mystery(string str) {
   string result = "";
   int n = str.length();
   for (int i = 0; i < n; i++) {
      if (i % 2 == 0) {
         result = str[i] + result;
      } else {
         result += str[i];
      }
   }
   return result;
}
```

i. What is the value of `mystery("retest")`?

ii. What is the computational complexity of the `mystery` function expressed in terms of big-O notation, where *N* is the length of the argument `str`. In this problem, you may assume that selecting a character from a string and calling the `length` method are constant-time operations and that concatenation and string assignment are linear in the length of the strings. Thus, concatenating `"abcd"` and `"e"` requires five times some constant per-character cost.

2. Heap-stack diagrams

Using the heap-stack diagrams from Chapter 12 as a model, draw a diagram showing how memory is allocated for each of the following problems. You need not include explicit addresses in your diagram, but must indicate—either through addresses or arrows—where reference values point in memory.

2a.

```
void f(int x, int k);
void g(int *p, int & k);

int main() {
   int k = 0;
   int x = 1;
   f(k, x);
   return 0;
}

void f(int x, int k) {
   g(&x, k);
}

void g(int *p, int & k) {
   k = 2 * *p;
}
```

←Diagram at this point just before the function returns

heap | *stack*

2b.

```
void foo(int array[], int n);

int main() {
   int *p = new int[3];
   foo(p, 3);
   return 0;
}

void foo(int array[], int n) {
   int *p = &array[n];
   while (p-- != array) {
      *p = array - p;
   }
}
```

←Diagram at this point just before the function returns

heap

stack

2c.

```
struct Student {
   int id;
   double gpa;
};

void initStudent(Student & s, int id);

int main() {
   Student advisees[2];
   advisees[0].id = 1414214;
   advisees->gpa = 3.50;
   initStudent(advisees[1], 1618034);
   return 0;
}

void initStudent(Student & s, int id) {
   Student *sptr = new Student;
   sptr->id = id;
   sptr->gpa = 3.80;
   s = *sptr;
}
```

←Diagram at this point just before the function returns

heap

stack