Question #1 (3 points)

Consider the program below. What will it print?

```
typedef struct {
  int id;
  char* name;
} person;

person* create_person(int new_id, char* new_name) {
  person p;
  p.id = new_id;
  p.name = new_name;
  person* ptr = &p;
  return ptr;
}

int main() {
  person* p = create_person(0,"Jack");
  person* q = create_person(1,"Steve");

  printf("p is %s ", p->name);
  printf("q is %s\n", q->name);
  return 0;
}
```

- ○ A. The code will not compile
- ○ B. Will print "p = Jack q = Steve"
- ○ C. Will print "p = Steve q = Steve"
- ○ D. Will print "p = Jack q = Jack"
- ○ E. No guarantees because the create_person returns a pointer to a local variable
- ○ F. No guarantees because jack and steve don't have '\0' in the end
- ○ G. None of the above

Question #2 (4 points)

Consider the following implementation of foo(int n).

```
void foo(int n) {
  int i,j;
  for (i=0;i<n;i++) {
    j = 1;
    while (j<i) {
      j = j*2;
      printf("%d ", j);
    }
    printf("\n");
  }
}
```

Use the Big-O notation to express the runtime of foo as a function of n.

Choose the tightest possible answer.

- ○ A. O(1)
- ○ B. O(n)
- ○ C. O(log(n))
- ○ D. O(n log(n))
- ○ E. O(n^2)
- ○ F. None of the above
- ○ G. The code will not compile

Question #3 (3 points)

Consider the following implementation of bar(int* ar, int n). The function gets an array ar of length n, and works as follows.

```
void bar(int* ar, int n) {
  if (n==0)
    return;

  printf("%d ",ar[0]);
  bar(ar+1,n-1);
  printf("%d ",ar[0]);
}
```

Use the Big-O notation to express the runtime of bar as a function of n.

You may assume that n>0 and ar is allocated properly.

Choose the tightest possible answer.

- ○ A. O(1)
- ○ B. O(2)
- ○ C. O(log(n))
- ○ D. O(n)
- ○ E. O(n^2)
- ○ F. The recursion will never stop
- ○ G. The code will not compile because recursion is wrong
- ○ H. None of the above