CS302 Midterm Exam - October 13, 2015

Do your answers on the answer sheets provided. When you write code, you do not need to have "include" or "using" statements.

Question 1

For each of these answers, give me a numerical expression that only includes numbers, addition, subtraction, multiplication, division and/or exponentiation. I don't want factorials or more complex notation. It doesn't have to be completely reduced, or calculated out, but it should be clear.

```
main()
{
  int i, j, k;

  cin >> k >> j;

  for (i = 0; i < (1 << k); i++) {
    if (i & (1 << j)) printf("Yes\n");
  }
  exit(0);
}
```

Part A: If the program to the right receives "5 1" on standard input, how many lines will it print?

Part B: If the program to the right receives "8 0" on standard input, how many lines will it print?

Part C: If the program to the right receives "3 3" on standard input, how many lines will it print?

Part D: Suppose I have 7 cities, numbered 1 through 7, that I want to visit, starting with city 1. There is a road from each city to each other city, and I can only visit each city once. In how many different orders can I visit all of the cities?

Part E: Suppose I am playing fantasy football, and I need to select a team with four running backs and two quarterbacks. And suppose that there are 75 running backs and 32 quarterbacks to choose from. How many different teams can I choose?

Part F: In class, I went over a topcoder problem called **SimpleRotationDecoder**, where a three-letter, lower-case password was required to decode a string containing *n* lower-case letters and spaces. We didn't know the password, so we performed an enumeration. How many elements did we enumerate?

CS302 Midterm Exam - October 13, 2015 - Page Two

Question 2

What are the running times of each of the following? Use the answer sheet for this. In the questions that have both *m* and *n*, assume that *m* is less than *n*, and make sure that your answer is as precise as possible. For example, if something is *O(mn)*, then it is true that it is also $O(n^2)$; however, in that case *O(mn)* is the correct answer.

- A. Sorting a random vector of *n* elements with insertion sort.
- B. Creating a map from *n* elements.
- C. The statement **s2 = s1**, when **s1** is a string with *n* elements, and **s2** is a string with *m* elements.
- D. Creating a priority queue from *n* elements.
- E. Creating a sorted vector from a map with *n* elements.
- F. Performing *n* **Push()** operations on a priority queue with *m* elements.
- G. The statement **s2 = s1.c_str()**, when **s1** is a string with *n* elements, and **s2** a **(char *)** that points to a C-style string with *m* elements.
- H. Performing *m* **Pop()** operations on a priority queue with *n* elements.
- I. Sorting a random vector of *n* elements with selection sort.
- J. Finding the *m* largest elements on a map with *n* elements.
- K. Sorting a "nearly" sorted vector of *n* elements with insertion sort.
- L. Finding the *m* elements that are closest in value to a given element, on a map with *n* elements.
- M. Performing *m* **Push()** operations on a priority queue with *n* elements.
- N. Erasing *m* elements from a map with *n* elements.
- O. Sorting a "nearly" sorted vector of *n* elements with selection sort.

Question 3

Part A: Which of the following vectors represent binary heaps?

- **A-1**: `{ 12, 19, 24, 51, 43, 67, 31, 48, 66, 78 }`
- **A-2**: `{ 1, 3, 14, 5, 24, 90, 78, 77, 30, 81 }`
- **A-3**: `{ 18, 20, 20, 37, 41, 64, 34, 44, 38, 85 }`
- **A-4**: `{ 8, 25, 30, 35, 78, 70, 28, 92, 33, 83 }`

Part B: Give me the vector version of the binary heap that results when you call **Push(3)** on the following heap: { 9, 23, 11, 57, 38, 17, 12, 86, 88, 80 }

Part C: Give me the vector version of the binary heap that results when you call **Pop()** on the following heap: { 6, 17, 45, 21, 26, 51, 52, 74, 67, 89 }

Question 4

You are sorting the following vector of integers:

{ 10, 47, 99, 64, 77, 35, 56, 21 }

Selection: Suppose you run selection sort on this vector. From the choices below, circle all of the lines of output that will be printed, if you print the vector after each pass of selection sort. You don't have to worry about the order in which the lines are printed -- just circle the letters that correspond to the lines that are printed.

Insertion: Suppose you run insertion sort on this vector. From the choices below, circle all of the lines of output that will be printed, if you print the vector after each pass of insertion sort.

Note -- these choices are sorted lexicographically, so that you can find the ones you want easily.

```
a.  10  21  35  47  56  64  77  99
b.  10  21  35  47  56  64  99  77
c.  10  21  35  47  56  99  64  77
d.  10  21  35  47  56  99  77  64
e.  10  21  35  47  77  99  56  64
f.  10  21  35  47  99  64  77  56
g.  10  21  35  64  77  99  56  47
h.  10  21  47  77  64  35  99  56
i.  10  21  47  99  64  77  35  56
j.  10  21  99  64  77  35  56  47
k.  10  35  21  47  56  64  77  99
l.  10  35  47  21  56  64  77  99
m.  10  35  47  56  21  64  77  99
n.  10  35  47  56  64  77  99  21
o.  10  35  47  64  77  99  56  21
p.  10  35  56  64  77  99  47  21
q.  10  47  35  56  21  64  77  99
r.  10  47  35  56  64  21  77  99
s.  10  47  64  35  56  21  77  99
t.  10  47  64  77  35  56  21  99
u.  10  47  64  77  99  35  56  21
v.  10  47  64  99  77  35  56  21
w.  10  47  99  64  77  35  56  21
x.  10  56  77  47  99  64  35  21
y.  10  77  99  47  56  35  64  21
z.  10  99  56  35  21  77  64  47
```

Question 5

On the answer sheet provided, implement the constructor, **Union()** and **Find()** operations of Disjoint Sets. You may implement any of the three options that I described in class. When you're done, in the space provided, specify which option that you implemented, and what the running times of all three operations are. I don't care which option you implement, and there is no bonus or penalty for one option over another.

Question 6

Write a recursive function with the following prototype:

```
long long two_ks(int K, long long S);
```

This function should return the value $2^{\mathbf{K}}\%\mathbf{S}$, and it should run in time *O(log(**K**))*. You may find the following two observations very useful:

- If **K** is odd, then $2^{\mathbf{K}}\%\mathbf{S} = (2* (2^{(\mathbf{K}-1)}\%\mathbf{S}))\%\mathbf{S}$.
- If **K** is even, then $2^{\mathbf{K}}\%\mathbf{S} = ((2^{(\mathbf{K}/2)}\%\mathbf{S}) * (2^{(\mathbf{K}/2)}\%\mathbf{S})) \%\mathbf{S}$.