CS302 Midterm Exam - March 7, 2017

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

Question 1 - 21 points

Please make the following assumptions with this question:

- The disjoint set implementation is union-by-rank with path compression.
- *n* is greater than *m*, and *m* is greater than **log(*n*)**.
- Quicksort uses the "median-of-three" pivot selection.

Give me the big-O running times of the following operations. All of your answers should be in terms of *n* and/or *m*. I want the most precise answer, not something redundant like *O(n)+O(m)*. This would be *O(n)*, because I've told you that *n* is greater than *m*.

- **A**: Sorting a "nearly" sorted vector, composed of *n* elements, with Insertion sort.
- **B**: Performing *m* `Pop()` operations on a priority queue of *n* elements.
- **C**: Creating a multimap from a vector with *n* elements.
- **D**: The slowest possible running time of Merge sort on a vector of *n* elements.
- **E**: Given a map with *n* elements, creating a vector composed of the *m* smallest values greater than a given element.
- **F**: Calling `Percolate_Up()` on element *n/2* on a priority queue with *n* elements.
- **G**: The slowest possible running time of Quicksort on a vector of *n* elements.
- **H**: Performing *m* `Find()` operations on an instance of disjoint sets with *n* elements.
- **I**: Performing *m* `Union()` operations on an instance of disjoint sets with *n* elements.
- **J**: Calling `Percolate_Down()` on element *n/2* on a priority queue with *n* elements.
- **K**: Creating a priority queue from a vector with *n* elements, and performing one `Pop()` operation.
- **L**: Given a multiset with *n* elements, creating a sorted vector composed of those elements.
- **M**: Sorting a "nearly" sorted vector, composed of *n* elements, with Merge sort.
- **N**: Sorting a "nearly" sorted vector, composed of *n* elements, with Quicksort.
- **O**: The slowest possible running time of Insertion sort on a vector of *n* elements.

Question 2 -- 15 points

In this question, please give me answers which are mathematical expressions involving exponents and factorials. I'd prefer that you not simplify them or multiply them out.

- **Part A**: I am a salesman whose route covers 12 cities. In how many different orders can I visit all 12 cities, if I'm not allowed to visit a city twice?
- **Part B**: On the planet *vowelopia*, everyone's name has to contain exactly 10 characters, which can only be vowels (A, E, I, O, U). How many distinct names are there?
- **Part C** My basketball team has 12 players. At any one time, there can only be 5 players on the court. How many different combinations of players can there be on the court?
- **Part D**: How many subsets are there of the set { 34, 52, 22, 66, 43, 1 }
- **Part E**: If I have 15 different colored balls, how many ways are there to put them into 15 numbered boxes?
- **Part F**: If I have 5 identical white balls and 18 identical black balls, how many ways are there to put them into 23 numbered boxes?

Question 3 - 25 points

In the box to the right are a bunch of vectors, which will be used for the the questions below. When you're answering the questions, make sure that you are referring to the correct vector!

index	0	1	2	3	4	5	6	7	8
A	2	4	3	6	7	3	5	8	9
B	1	3	2	5	6	4	7	9	8
C	5	6	3	1	2	8	0	4	7
D	5	2	0	4	6	1	3	8	7
E	7	4	2	0	5	8	1	3	6
F	5	0	6	2	8	3	7	1	4
G	6	6	0	8	6	4	6	7	2
H	5	0	1	-1	0	-1	1	-1	3
I	1	2	1	2	1	3	2	2	1
J	7	3	6	2	1	4	5	8	9

Part A: Suppose vector `A` is a heap. On the answer sheet, show me what `A` becomes after you call **Push(1)** on `A`.

Part B: Suppose vector `B` is a heap. On the answer sheet, show me what `B` becomes after you call **Pop()** on `B`.

Part C: You are running selection sort on vector `C`. On the answer sheet, show me what `C` becomes after two iterations of the outer **for** loop.

Part D: You are running insertion sort on vector `D`. On the answer sheet, show me what `D` becomes after four iterations of the outer **for** loop. Assume you are not performing a sentinelization pass beforehand.

Part E: You are running quicksort on vector `E`, using "median-of-three." What is the value of the pivot?

Part F: You are running quicksort on vector `F`. The pivot has a value of 5. How many swaps will you perform when you do the first partition? Do not count the final swap of the pivot.

Part G: You are running quicksort on vector `G`. The pivot has a value of 6. How many swaps will you perform when you do the first partition? Do not count the final swap of the pivot.

Part H: Suppose that vector `H` is the `links` vector in an implementation of disjoint sets. Tell me the set id for every value of *i* from 0 to 8.

Part I: Suppose that vectors `H` and `I` are the first 9 entries of the `links` and `ranks` vectors respectively in an implementation of disjoint sets. And suppose that the implementation is Union-by-Rank with path compression. I want to call `Union(a,b)` such that the `ranks` vector will change when I make the call. Give me values of `a` and `b` that will cause this to happen (you don't have to give me all such values -- just any values of `a` and `b` that will make the `ranks` vector change).

Part J: I am in the middle of performing heap sort and my vector looks like vector `J`. How many elements are in the heap?

Question 4 - 15 points

Part A: Given a string **S**, an *ordered sub-anagram* **A** of **S** is a string with the following properties:

- Every character in **A** is also in **S**.
- The characters in **A** are in alphabetical order.
- There are no repeating characters in **A**.
- **A** can be the empty string.

Now, suppose **S** is a string with *n* characters, all of which are upper-case letters, and **S** has no duplicate letters. How many ordered sub-anagrams of **S** are there?

Part B Write a procedure called **OSA** whose prototype is below. **OSA** should print all of the ordered sub-anagrams of **S**. You may assume that **S** is composed of upper-case letters with no duplicates. You may print the strings in any order.

```
void OSA(string S);
```

Question 5 - 14 points

To the right is my class definition for disjoint sets. Please implement **Find**() using path compression. I want you to implement it twice. In your first implementation, make it non-recursive. In your second implementation, make it recursive.

```
class Disjoint {
  public:
    Disjoint(int nelements);
    int Union(int s1, int s2);
    int Find(int element);
    void Print();
  protected:
    vector <int> links;
    vector <int> ranks;
};
```

Question 6 - 10 points

Your friend has written the program to the right. When he runs it three times with inputs of 1, 12 and 13, he gets the following:

```
UNIX> echo 1 | a.out
01234567890
UNIX> echo 12 | a.out
012345678901234567890
UNIX> echo 13 | a.out

UNIX>
```

He is confused as to why his program is acting this way. Please explain to me why the output of these three runs is what it is. One sentence will not suffice here -- shoot for 3-5 sentences and be as precise as you can be.

```
int main()
{
  string s;
  const char *sp;
  int i, j;

  cin >> j;
  s = "0123456789";
  sp = s.c_str();
  for (i = 0; i < j; i++) s.push_back('0'+i%10);
  printf("%s\n", sp);
}
```