CS302 Midterm Exam - October 18, 2018

Do your answers on the answer sheets provided. When you write code, you do not need to have "include" or "using" statements. The header files for **PQueue.h** and **DisjointSet.h** are on the last page of this exam.

Question 1 - 20 points

In this question, you are to assume that variables are declared and initialized as specified in the table to the right. Also assume that the procedure **a()** takes an integer as an argument, returns an integer, and is O(1).

For each snippet of code below, tell me the precise big-O running time in terms of m, n and r.

Variable	Type	Additional Information
`v`	`vector <int>`	Contains n elements
`x`	`map <int,int>`	Contains n elements
`xit`	`map <int,int>::iterator`	Uninitialized
`d`	`DisjointSetByRankWPC *`	Points to an instance that contains n elements
`e`	`DisjointSet *`	Initially set to `NULL`
`p`	`PQueueHeap *`	Points to an instance that contains n elements
`q`	`PQueueHeap *`	Initially set to `NULL`
`i, j, k`	`int`	Initially set to 0
`m`	`int`	Is a value < n
`r`	`int`	Is a value > n

```
// Snippet 1
for (i = 0; i < r; i++) {
  j = d->Find(a(i)%n);
  k = d->Find(a(i+r)%n);
  if (j != k) d->Union(j, k);
}
```

```
// Snippet 2
e = new DisjointSetBySize(n);
for (i = 1; i < n; i++) {
  j = e->Union(j, i);
}
```

```
// Snippet 3
for (i = 0; i < v.size(); i++) {
  x.insert(make_pair(p->Pop(), a(i)));
}
```

```
// Snippet 4
for (xit = x.begin(); xit != x.end(); xit++) {
  k += xit->second;
}
```

```
// Snippet 5
for (i = 0; i < v.size(); i++) {
  for (j = 1; j < r; j *= 2) {
    k = j + v[i];
    v[i] = a(k);
  }
}
```

```
// Snippet 6
for (i = 0; i < n; i++) {
  for (j = 0; j < i; j++) {
    k += a(k+i+j);
  }
}
```

```
// Snippet 7
for (i = 0; i < (1 << m); i++) {
  for (j = 0; j < r; j++) {
    k += a(i);
  }
}
```

```
// Snippet 8
for (i = 0; i < n; i++) {
  xit = x.upper_bound(a(i));
  v[i] = xit->second;
}
```

```
// Snippet 9
for (i = 0; i < r; i++) {
  p->Push(a(i));
}
```

```
// Snippet 10
q = new PQueueHeap(v);
```

CS302 Midterm Exam - October 18, 2018

Question 2 - 32 points

In each part of this question, I give you a vector and ask you to do something with this vector. On the answer sheet for this question, I replicate each vector. Answer the question by showing how the vector changes. I have answered **Part 0** on the answer sheet to show you how I want this done.

Part 0: Given a vector `{ 2, 4, 8, 9, 5 }`, add one to the elements with odd values.

Part 1: Given a priority queue stored in the vector `{ 5, 29, 28, 47, 50, 86, 60, 84, 59, 83, 72 }`, show the vector that you get when you call **Push(3)**.

Part 2: Given a priority queue stored in the vector `{ 8, 25, 31, 43, 35, 62, 94, 83, 82, 38, 90 }`, show the vector that you get when you call **Pop()**.

Part 3: Show the vector representation of priority queue that results when you create it from this vector:
`{ 31 , 69 , 90 , 78 , 10 , 87 , 73 }`.

Part 4: You have an instance of **DisjointSetBySize** named **d**, with the following **links** and **sizes** vectors. Show how each of these changes if you call `d->Union(d->Find(9),d->Find(10))`.

```
Node:     0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
Links:   -1 -1  0  1  1 -1  1 -1 11  0  1 -1 11 -1  0 -1
Sizes:    4  5  1  1  1  1  1  1  1  1  1  3  1  1  1  1
```

Part 5: You have an instance of **DisjointSetByRankWPC** named **d**, with the following **links** and **ranks** vectors. Show how each of these changes if you call `d->Union(d->Find(14),d->Find(8))`.

```
Node:     0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
Links:   23 11 16  0 13 23 23 23  4 13 13 23 23 23 16 10 -1 23 19 13  4  7 23 -1 -1
Ranks:    2  1  1  1  2  1  1  2  1  1  2  2  1  3  1  1  2  1  1  2  1  1  1  4  1
```

Part 6: You are sorting the following vector with bubble sort. Show me what the vector looks like after two iterations of the outer loop of the algorithm: `{ 80, 64, 41, 66, 47, 4, 51, 56, 40, 3 }`.

Part 7: You are sorting the following vector with selection sort. Show me what the vector looks like after three iterations of the outer loop of the algorithm: `{ 87, 77, 54, 92, 40, 46, 59, 82, 12, 83 }`.

Part 8: You are sorting the following vector with insertion sort. Show me what the vector looks like after four iterations of the outer loop of the algorithm: `{ 85, 73, 60, 13, 88, 62, 9, 10, 65, 86 }`.

CS302 Midterm Exam - October 18, 2018

Question 3 - 16 points

The function *g(x)* is defined as follows:

- *g(0)* equals one.
- Otherwise, suppose the highest bit that is set in *x* is the *y-th* bit (zero indexed). And suppose that *z* is equal to *x* with the *y-th* bit unset. Then *g(x)* equals *y*(g(z)+z)*.

For example:

x	*y*	*z*	*g(z)* as a formula	*g(z)*
0	-	-	1	1
1	0	0	0 * (*g(0)* + 0)	0
2	1	0	1 * (*g(0)* + 0)	1
3	1	1	1 * (*g(1)* + 1)	1
4	2	0	2 * (*g(0)* + 0)	2
5	2	1	2 * (*g(1)* + 1)	2
6	2	2	2 * (*g(2)* + 2)	6

x	*y*	*z*	*g(z)* as a formula	*g(z)*
7	2	3	2 * (*g(3)* + 3)	8
8	3	0	3 * (*g(0)* + 0)	3
9	3	1	3 * (*g(1)* + 1)	3
10	3	2	3 * (*g(2)* + 2)	9
11	3	3	3 * (*g(3)* + 3)	12
12	3	4	3 * (*g(4)* + 4)	18
13	3	5	3 * (*g(5)* + 5)	21

Write a function that computes *g()*. It should have the following prototype:

```
int g(int x);
```

In case it's not clear, `g()` should be recursive. You don't need to figure out anything fancy about `g()` -- you simply need to implement it as a recursive function from that definition above. I only include the examples so that you can see how the recursion works. Assume that *x* is between 0 and 1024.

Question 4 - 16 points

A DNA sequence is a string consisting of the letters A, C, G and T. Write a program that reads a value *n* from standard input, and then enumerates all DNA sequences whose lengths are exactly *n*. It should print one sequence on each line. Assume that *n* is between 0 and 20.

You are not allowed to use the C++ math functions **exp()** or **pow()**. You don't need them.

Question 5 - 16 points

In your city, there is one really long street. On this street, there are restaurants, and there are cars. Each restaurant has a position on the street, which will be a double between -10,000 and 10,000. No two restaurants have the same positions. Each car also has a position on the street, and a number of occupants.

You assume that each car is going to go to the closest restaurant, and all of the occupants of the car will eat at the restaurant. If a car is equidistant to two restaurants, it will go to the one with the lower position.

Your job is to write the procedure **Occupants()**. It has the following prototype:

```
void Occupants(vector <double> &Rpos,   // Positions of the restaurants
               vector <double> &Cpos,   // Positions of the cars
               vector <int> &Cocc);     // Occupants of the cars
```

Neither **Rpos** not **Cpos** will be sorted, but each element of **Rpos** will be distinct and between -10,000 and 10,000. **Cocc[i]** will contain the number of occupants in the car at position **Cpos[i]**. The elements of **Cpos** will also be between -10,000 and 10,000.

Your procedure should print out the position and number of occupants of each restaurant, sorted by the position of the restaurant. Print them one per line, and use "%9.2lf" to print out the positions.

Your program should run in time *O(C log(R) + R log(R))*, where *C* is the number of cars and *R* is the number of restaurants. In other words, **Rpos**, **Cpos** and **Cocc** can be pretty large, say, up to 1,000,000 elements each.

Example:

- **Rpos** = { 238, 400, -300 }
- **Cpos** = { 300, 200, 500 }
- **Cocc** = { 5, 4, 3 }

The output should be :

```
  -300.00 0
   238.00 9
   400.00 3
```

CS302 Midterm Exam - October 18, 2018

DisjointSet.h

```
#pragma once
#include <vector>
using namespace std;

class DisjointSet {
  public:
    virtual ~DisjointSet() {};
    virtual int Union(int s1, int s2) = 0;
    virtual int Find(int element) = 0;
    virtual void Print() = 0;
};

class DisjointSetBySize : public DisjointSet {
  public:
    DisjointSetBySize(int nelements);
    int Union(int s1, int s2);
    int Find(int element);
    void Print();

  protected:
    vector <int> links;
    vector <int> sizes;
};
```

```
class DisjointSetByHeight : public DisjointSet {
  public:
    DisjointSetByHeight(int nelements);
    int Union(int s1, int s2);
    int Find(int element);
    void Print();

  protected:
    vector <int> links;
    vector <int> heights;
};

class DisjointSetByRankWPC : public DisjointSet {
  public:
    DisjointSetByRankWPC(int nelements);
    int Union(int s1, int s2);
    int Find(int element);
    void Print();

  protected:
    vector <int> links;
    vector <int> ranks;
};
```

PQueue.h

```
#include <vector>
#include <set>
using namespace std;

class PQueue {
  public:
    virtual void    Push(double d) = 0;
    virtual double  Pop()          = 0;
    virtual int     Size()         = 0;
    virtual bool    Empty()        = 0;
    virtual void    Print()        = 0;
};
```

```
class PQueueSet : public PQueue {
  public:
    void    Push(double d);
    double  Pop();
    int     Size();
    bool    Empty();
    void    Print();

    PQueueSet();
  protected:
    multiset <double> elements;
};
```

```
class PQueueHeap : public PQueue {
  public:
    void    Push(double d);
    double  Pop();
    int     Size();
    bool    Empty();
    void    Print();

    PQueueHeap();
    PQueueHeap(vector <double> &init);
  protected:
    vector <double> h;
    void Percolate_Down(int index);
};
```

Map Functions

- **insert(pair)**
- **iterator find(key)**
- **iterator begin()**
- **iterator end()**
- **reverse_iterator rbegin()**
- **reverse_iterator rend()**
- **iterator upper_bound(key)** - strictly greater than
- **iterator lower_bound(key)** - greater than or equal to