CS 101 Exam 1 **Summer 2017** **Solution**

This exam has 120 points possible (includes 20 built-in extra credit points).

1. For each algorithm below, write each indicated running time as the simplest Big O function of n. **[21 points]**

	Best case	Average case	Worst case
Merge sort	**O(n lg n)**	**O(n lg n)**	**O(n lg n)**
Quick sort	**O(n lg n)**	**O(n lg n)**	**$O(n^2)$**
Bubble sort	**O(n)**	**$O(n^2)$**	**$O(n^2)$**
Insertion sort	**O(n)**	**$O(n^2)$**	**$O(n^2)$**
Selection sort	**$O(n^2)$**	**$O(n^2)$**	**$O(n^2)$**
Linear search	**O(1)**	**O(n)**	**O(n)**
Binary search	**O(1)**	**O(lg n)**	**O(lg n)**

2. Arrange the given running times in Big O order from best (fastest, most efficient) to worst. **(15 points)**

3^n $\sqrt{n}$ $n \lg n$ $\lg n$ n^3 n^n 2^n $\sqrt[3]{n}$ $n!$ $\lg^2 n$ n^2 9999 $n/\lg n$ $\lg \lg n$ 2^{2^n} n

9999	**$\lg \lg n$**	**$\lg n$**	**$\lg^2 n$**	**$\sqrt[3]{n}$**	**$\sqrt{n}$**	**$n/\lg n$**	**n**	**$n \lg n$**	**n^2**	**n^3**	**2^n**	**3^n**	**$n!$**	**n^n**	**2^{2^n}**

Best **Worst**

3. Write the running time of each code fragment below as the simplest Big O function of n. **[20 points]**

```
for (a=1; a<=1000*n*n*n; a+=50*n)
    z++;
```

$O(n^3 / n) = O(n^2)$

```
for (a=2*n; a<=5*n; a++)
    for (b=a-100; b<=a+100; b++)
        z++;
```

$O((3n+1)*201) = O(n)$

```
for (a=1; a*a*a<=n; a++)
    for (b=1; b<=n; b+=b)
        z++;
```

$O(\sqrt[3]{n} \lg n)$

```
for (a=1; a<=n*n; a++)
    for (b=1; b<=a*a*a; b++)
        z++;
```

$O(n^2*(n^2)^3) = O(n^8)$

```
for (a=n; a>=1; a/=2)
    for (b=a; b>=1; b--)
        z++;
```

$O(n + n/2 + n/4 + \ldots + 1) = O(2n–1) = O(n)$

```
p=0;
for (a=1; a<=n; a*=2) p++;
for (b=1; b<=p*p; b++) z++;
```

$O(\lg n * \lg n) = O(\lg^2 n)$

```
p=1; q=1;
for (a=1; a<=n; a++) p*=2; // p=2^n
for (b=1; b<=n; b++) q*=p; // q=2^p
for (c=1; c<=q; c++) z++;
```

$O((2^n)^n) = O(2^{n^2})$

```
for (a=1; a<=n; a++)
    if (k*k>n) y++;
    else
        for (b=1; b<=n; b++)
            z++;
```

$O(n^2)$

I'd intended for the code to appear like this:

```
for (a=1; a<=n; a++)
    if (a*a>n) y++;
    else
        for (b=1; b<=n; b++)
            z++;
```

$O((n - \sqrt{n})*1 + \sqrt{n} *n) = O(n \sqrt{n})$

So both of the above answers will be accepted

4. Write the output of this C++ program on the indicated blanks. **[12 points]**

#include <iostream>
using namespace std;
 	
int main( ) {
  int a, b, c, d;
 	
  a=b=2; a=b==3; cout << a << b << endl;
 	
  a=4; b=5; a==b?a:b=6; cout << a << b << endl;
 	
  a=7!=7; b=!a?8:9; cout << a << b << endl;
 	
  a=1, b=3, c=a--, d=--b; cout << a << b << c << d << endl;
 	
  a=5, b=7, c=9, a+=b-=c%=5; cout << a << b << c << endl;
 	
  a=7>6>5; b=7<6<5; cout << a << b << endl;
 	
  a=1; b=2; c=0 || a--; d=1 || b--;
  cout << a << b << c << d << endl;
 	
  a=1; b=2; c=0 && ++a; d=1 && ++b;
  cout << a << b << c << d << endl;
 	
  switch (3) {
  case 4: cout << 5;
  case 3: cout << 6;
  case 2: cout << 7;
  default: cout << 8;
  }
  cout << endl;
 	
  for (a=4, b=9; a<b; b--) { a++; cout << a << b; }
  cout << endl;
 	
  a=1; while (a<1) cout << a;
  b=3; do cout << b; while (b<3);
  cout << endl;
 	
  a=9;
  while (a) {
  a--;
  if (a==6) continue;
  if (a==3) break;
  cout << a;
  }
  cout << endl;
}
	Output:
	
	02
	
	46
	
	08
	
	0212
	
	834
	
	01
	
	0211
	
	1301
	
	678
	
	596877
	
	3
	
	8754

5. Write the output of this C++ program on the indicated blanks. **[16 points]**

```
#include <iostream>
using namespace std;

int F (int n) {
    if (n<=6) return n;
    cout << "n = " << n << endl;
    int u = F(n-2);
    int v = F(n-3);
    int w = F(n-4);
    int x = F(n-5);
    int y = F(n-6);
    int z = u+v+w+x+y;
    cout << "z = " << z << endl;
    return z;
}

int main( ) {
    F (12);
    return 0;
}
```

Output

n = 12
n = 10
n = 8
z = 20
n = 7
z = 15
z = 50
n = 9
n = 7
z = 15
z = 33
n = 8
z = 20
n = 7
z = 15
z = 124

6. Trace each sorting algorithm as specified below, and show the contents of the array (or two subarrays) at the specified point during the algorithm. For each algorithm, start with the array shown below as input. **[12 points]**

7	5	0	8	3	6	1	9	4	2

a. Show the array contents after each of the first *two* iterations of the outer loop of bubble sort.

5	**0**	**7**	**3**	**6**	**1**	**8**	**4**	**2**	**9**
0	**5**	**3**	**6**	**1**	**7**	**4**	**2**	**8**	**9**

b. Show the array contents after each of the first *two* iterations of the outer loop of selection sort.

7	**5**	**0**	**8**	**3**	**6**	**1**	**2**	**4**	**9**
7	**5**	**0**	**4**	**3**	**6**	**1**	**2**	**8**	**9**

or

0	**5**	**7**	**8**	**3**	**6**	**1**	**9**	**4**	**2**
0	**1**	**7**	**8**	**3**	**6**	**5**	**9**	**4**	**2**

c. Show the array contents after each of the first *four* iterations of the outer loop of insertion sort.

5	**7**	0	8	3	6	1	9	4	2
0	**5**	**7**	8	3	6	1	9	4	2
0	**5**	**7**	**8**	3	6	1	9	4	2
0	**3**	**5**	**7**	**8**	6	1	9	4	2

d. Show the two subarrays after the two top-level recursive calls in merge sort, but before the top-level call to merge.

0	**3**	**5**	**7**	**8**

and

1	**2**	**4**	**6**	**9**

e. Show the two subarrays after the call to partition, but before the two top-level recursive calls in quick sort. Choose the pivot element at index (low+high)/2.

Pivot = A[(0+9)/2] = A[4] = 3

2	**1**	**0**	**3**

and

8	**6**	**5**	**9**	**4**	**7**

7. A bit array is an abstract data type that can be used to implement subsets of a finite universal set. Complete each operation of the C++ class BitArray on the subsequent pages so that the example program shown below will produce the given output. Only the insert and remove methods can modify a previously-existing BitArray object. Most of the methods correspond to well-known set operations whose meanings should be self–explanatory. Error checking is not needed. **(24 points)**

```
#include <iostream>
using namespace std;

// class BitArray would appear here

int main( ) {
    BitArray p(10);
    cout << "p: " << p << endl;             // { }

    for (int k=1; k<10; k+=2)
        p.insert(k);
    cout << "p: " << p << endl;             // {1,3,5,7,9}

    BitArray q(10);
    for (int k=0; k<10; k+=3)
        q.insert(k);
    cout << "q: " << q << endl;             // {0,3,6,9}

    BitArray r = p.set_union(q);
    cout << "r: " << r << endl;             // {0,1,3,5,6,7,9}

    r.remove(3); r.remove(9);
    cout << "r: " << r << endl;             // {0,1,5,6,7}

    BitArray s = r.complement( );
    cout << "s: " << s << endl;             // {2,3,4,8,9}

    BitArray t = p.intersect(q);
    cout << "t: " << t << endl;             // {3,9}

    BitArray u = t.complement( );
    cout << "u: " << u << endl;             // {0,1,2,4,5,6,7,8}
    return 0;
}
```

Output

```
p: 0000000000
p: 0101010101
q: 1001001001
r: 1101011101
r: 1100011100
s: 0011100011
t: 0001000001
u: 1110111110
```

```
class BitArray {
private:

        int size;
        bool *arr;

public:
        BitArray (int);
        void insert (int);
        void remove (int);
        bool contains (int) const;
        BitArray & set_union (const BitArray &) const;
        BitArray & intersect (const BitArray &) const;
        BitArray & complement( ) const;
        friend ostream & operator<< (ostream &, const BitArray &);
};
BitArray::BitArray (int n) {


        size = n;
        arr = new bool[size];
        for (int k=0; k<size; k++)
                arr[k]=false;




}
void BitArray::insert (int k) {

        arr[k]=true;




}
void BitArray::remove (int k) {

        arr[k]=false;




}
bool BitArray::contains (int k) const {

        return arr[k];



}
```

```
BitArray & BitArray::set_union (const BitArray &v) const {


        BitArray *w = new BitArray(size);
        for (int k=0; k<size; k++)
                w->arr[k] = arr[k] || v.arr[k];
        return *w;



}
BitArray & BitArray::intersect (const BitArray &v) const {


        BitArray *w = new BitArray(size);
        for (int k=0; k<size; k++)
                w->arr[k] = arr[k] && v.arr[k];
        return *w;



}
BitArray & BitArray::complement( ) const {


        BitArray *w = new BitArray(size);
        for (int k=0; k<size; k++)
                w->arr[k] = !arr[k];
        return *w;



}
ostream & operator<<(std::ostream & os, const BitArray &v) {


        for (int k=0; k<v.size; k++)
                cout << v.arr[k];
        return os;



}
```