CS 101 Exam 1 **Summer 2016** **Solution**

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

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

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

```
for (a=0; a<=n; a++)
    for (b=0; b<=n; b++)
        z++;
```

$O(n^2)$

```
for (a=n; a<=2*n; a++)
    for (b=a-10; b<=a+10; b++)
        z++;
```

O(n)

```
for (a=0; a*a*a<=n; a++)
        z++;
```

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

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

O(lg n)

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

n + n/2 + n/4 + n/8 + … + 1 = 2n–1 = O(n)

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

$O(\sqrt{n}$ lg n)

```
P=1;
for (a=1; a<=n; a++)
    p*=5;
for (b=1; b<=p; b++)
    z++;
```

$O(5^n)$

```
for (a=0; a<=n; a++)
    for (b=0; b<=a; b++)
        for (c=a; c<=n; c++)
            for (d=b; d<=c; d++)
                z++;
```

$O(n^4)$

3. 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, e, f, i, j, w, x, y, z;
 	
  w=2!=3; x=!w?4:5; cout << w << x << endl;
 	
  y=6; z=7; y==z?y:z=8; cout << y << z << endl;
 	
  a=b=1; a=b==2; cout << a << b << endl;
 	
  c=1, d=3, e=--c, f=d--; cout << c << d << e << f << endl;
 	
  w=4, x=7, y=3, w+=x-=y*=2; cout << w << x << y << endl;
 	
  a=7<6<5; b=7>6>5; cout << a << b << endl;
 	
  w=1; x=2; y=0 && w++; z=1 && x++;
  cout << w << x << y << z << endl;
 	
  w=1; x=2; y=0 || --w; z=1 || --x;
  cout << w << x << y << z << endl;
 	
  switch (2) {
  case 1: cout << 1;
  case 2: cout << 2;
  case 3: cout << 3;
  default: cout << 4;
  }
  cout << endl;
 	
  for (i=2, j=7; i<j; i++) { j--; cout << i << j; }
  cout << endl;
 	
  x=8; while (x>8) cout << x;
  y=9; do cout << y; while (y>9);
  cout << endl;
 	
  z=6;
  while (z) {
  z--;
  if (z==4) continue;
  else if (z==2) break;
  cout << z;
  }
  cout << endl;
}
	Output:
	
	15
	
	68
	
	01
	
	0203
	
	516
	
	10
	
	1301
	
	0201
	
	234
	
	263544
	
	9
	
	53

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

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

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

int main( ) {
    F (11);
    return 0;
}
```

Output:

n=11

n=9

n=7

z=14

n=6

z=10

z=33

n=8

n=6

z=10

z=22

n=7

z=14

n=6

z=10

z=79

5. Trace each sorting algorithm as specified below, and show the contents of the array (or two subarrays) at the specified point during the algorithm. Use the array given below as input for each sorting algorithm. For each algorithm, start with the original array. **[10 points]**

2	8	4	0	6	3	9	5	1	7

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

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

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

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

or

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

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

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

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

0	**2**	**4**	**6**	**8**

and

1	**3**	**5**	**7**	**9**

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

Pivot = A[(0+9)/2] = A[4] = 6

2	**1**	**4**	**0**	**5**	**3**

and

9	**6**	**8**	**7**

6. Write a C++ class Triple that permits the given program to operate as shown to produce the given output. Each Triple represents a point or vector in 3-dimensional space. The class provides a dot-product operation * which is defined by $(x_1,y_1,z_1)*(x_2,y_2,z_2) = x_1x_2 + y_1y_2 + z_1z_2$. The meanings of all the remaining operations should be self-explanatory. **(27 points)**

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

// class Triple should go here

int main( ) {
     Triple p;
     p.display( );
     p.set_value(1,6); p.set_value(2,7); p.set_value(3,10);
     cout << p.at(1) << ' ' << p.at(2) << ' ' << p.at(3) << endl;
     p.display( );
     Triple q(4,3,4);
     q.display( );
     Triple r(2,-1,-2);
     r.display( );
     Triple s = p+r;
     s.display( );
     Triple u = q-r;
     u.display( );
     Triple v = q*2;
     v.display( );
     cout << (p*q) << ' ' << (r*s) << endl;
     cout << p.min( ) << ' ' << q.min( ) << ' ' << r.min( ) << endl;
     cout << p.max( ) << ' ' << q.max( ) << ' ' << r.max( ) << endl;
     cout << boolalpha;
     cout << (s==u) << ' ' << (s==v) << endl;
     cout << (s!=u) << ' ' << (s!=v) << endl;
     return 0;
}
```

Output

```
(0,0,0)
6 7 10
(6,7,10)
(4,3,4)
(2,-1,-2)
(8,6,8)
(2,4,6)
(8,6,8)
85 -6
6 3 -2
10 4 2
false true
true false
```

```
class Triple {
      int x, y, z;
public:
      Triple ( ):
            x(0), y(0), z(0) { }
      Triple (int a, int b, int c):
            x(a), y(b), z(c) { }
      Triple operator+ (Triple t) {
            return Triple (x+t.x, y+t.y, z+t.z);
      }
      Triple operator- (Triple t) {
            return Triple (x-t.x, y-t.y, z-t.z);
      }
      Triple operator* (int s) {
            return Triple (x*s, y*s, z*s);
      }
      int operator* (Triple t) {
            return x*t.x + y*t.y + z*t.z;
      }
      bool operator== (Triple t) {
            return x==t.x && y==t.y && z==t.z;
      }
      bool operator!= (Triple t) {
            return x!=t.x || y!=t.y || z!=t.z;
      }
      int min ( ) {
            if (x<=y && x<=z) return x;
            else if (y<=z) return y;
            else return z;
      }
      int max ( ) {
            if (x>=y && x>=z) return x;
            else if (y>=z) return y;
            else return z;
      }
      int at (int k) {
            if (k==1) return x;
            else if (k==2) return y;
            else if (k==3) return z;
            else return 0;
      }
      void set_value (int k, int value) {
            if (k==1) x=value;
            else if (k==2) y=value;
            else if (k==3) z=value;
      }
      void display( ) {
            cout << '(' << x << ',' << y << ',' << z << ')'
                 << endl;
      }
};
```