Review Exercise 1

1. Why is the big-Oh notation so widely used?

- It facilitates development of bounds on the order of growth, even for complicated algorithms for which more precise analysis might not be feasible. Moreover, it is compatible with the "big-Omega" and "big-Theta" notations that theoretical computer scientists use to classify algorithms by bounding their worst-case performance. We say that $f(N)$ is $\Omega(g(N))$ if there exists constants c and N_0 such that $|f(N)| > cg(N)$ for $N > N_0$, and if $f(N)$ is $O(g(N))$ and $\Omega(g(N))$, we say that $f(N)$ is $\Theta(g(N))$. The "big-Omega" notation is typically used to describe a lower bound on the worst case, and the "big-Theta" notation is typically used to describe the performance of algorithms that are optimal in the sense that no algorithm can have better asymptotic worst-case order of growth. Optimal algorithms are certainly worth considering in pratical applications, but there are many other considerations.

1.4.4

2. Develop a table like the one on page 181 for TwoSum.

Statement Block	Time in Seconds	Frequency	Total Time
D	t_0	x (depends on input)	$t_0 x$
C	t_1	$\frac{N^2}{2} - \frac{N}{2}$	$t_1\left(\frac{N^2}{2} - \frac{N}{2}\right)$
B	t_2	N	$t_2 N$
A	t_3	1	t_4

Grandtotal: $\left(\frac{t_1}{2}\right)N^2 + \left(-\frac{t_1}{2} + t_2\right)N + t_3 + t_0 x$

Tilde Approximation: $\sim\left(\frac{t_1}{2}\right)N^2$ (assuming x is small)

Order of growth: N^2

1.4.5

a) $N+1 \quad \sim N$

b) $1 + \frac{1}{N} \quad \sim 1$

c) $\left(1+\frac{1}{N}\right)\left(1+\frac{2}{N}\right) \quad \sim 1$

d) $2N^3 - 15N^2 + N \quad \sim 2N^3$

e) $\lg(2N) / \lg N \quad \sim 1$

f) $\lg(N^2+1) / \lg N \quad \sim 1$

g) $N^{100} / 2^N \quad \sim 0$

3. $\sum_{i=0}^{N} 2^i = 2^{N+1} - 1$ —— ①

Lets prove ① is true for $N=0$

Left hand side $2^0 = 1$

Right hand side $2^{0+1} - 1 = 2 - 1 = 1$

① is true for $N=0$

Induction

Lets assume ① is true for k

$\sum_{i=0}^{k} 2^i = 2^{k+1} - 1$

Now for k+1

$\sum_{i=0}^{k+1} 2^i = (2^{(k+1)} - 1) + 2^{k+1}$

$= 2^{k+1} - 1 + 2^{k+1}$

$= 2 \cdot 2^{k+1} - 1$

$= 2^{(k+1)+1} - 1$

4. Write a piece of pseudo code to compare the N^{th} fibonacci number. Show its worst case algorithm analysis step by step and its time complexity in big Oh notation.

```
long fib(int n){
/*1*/   if (n<=1)
/*2*/       return 1;
        else
/*3*/       return fib(n-1) + fib(n-2);
}
```

$T(N) = T(N-1) + T(N-2) + 2 \Rightarrow T(N) >= fib(n)$

5. $3n^2 - 100n + 6 = O(n^2)$ - True because $n^2 \leq n^2$

$3n^2 - 100n + 6 = O(n^3)$ - True because $n^2 \leq n^3$

$3n^2 - 100n + 6 = O(n)$ - False because $n^2 > n$

$3n^2 - 100n + 6 = \Omega(n^2)$ - True because $n^2 \geq n^2$

$3n^2 - 100n + 6 = \Omega(n^3)$ - False because $n^2 \leq n^3$

$3n^2 - 100n + 6 = \Omega(n)$ - True because $n^2 \geq n$

$3n^2 - 100n + 6 = \Theta(n^2)$ - True because $n^2 = n^2$

$3n^2 - 100n + 6 = \Theta(n^3)$ - False because $n^2 \neq n^3$

$3n^2 - 100n + 6 = \Theta(n)$ - False because $n^2 \neq n$

$3n^2 - 100n + 6 = o(n^3)$ - True because $n^2 < n^3$

---

1. What does it mean a problem is in P? Name a problem that is in the class P.

- A problem is said to be polynomial (or in the class P) if it can be solved in time polynomial in its size. Binary Search is a problem in the P category.

2. What does it mean a problem is in NP? Name a problem that is in the class NP.

- A problem is said to be nondeterministically polynomial (or in the class NP) if a conjectured answer can be verified in time polynomial size. The satisfiability problem is a problem in the NP class.

3. What does it mean a problem is NP-hard?

- A problem is NP-hard if it is at least as hard as any problem in NP.

4. SAT problem is first NP-complete problem we discussed in class. Give a 3-SAT problem that is satisfiable and give another SAT problem that is not satisfiable.

Satisfiable

$(x_1 \vee \overline{x}_2 \vee \overline{x}_3) \wedge (\overline{x}_1 \vee \overline{x}_2 \vee x_3) \wedge (x_2 \vee x_3)$

$x_1 = 1$ $\quad (1 \vee \overline{0} \vee \overline{1}) \wedge (\overline{1} \vee \overline{0} \vee 1) \wedge (0 \vee 1)$

$x_2 = 0$ $\quad (1) \wedge (1) \wedge (1) = 1$

$x_3 = 1$

Unsatisfiable

$(\bar{x}_1 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee \bar{x}_3) \wedge (x_2)$

$x_1 = 1$ $\quad (\bar{1} \vee 0 \vee \bar{1}) \wedge (\bar{1} \vee \bar{1}) \wedge (0)$

$x_2 = 0$ $\quad (0) \wedge (0) \wedge (0)$

$x_3 = 1$ $\quad$ Unsatisfiable

5. Determine whether or not the following expression is satisfiable. Explain your answer.

$(x_1 \vee x_2 \vee x_3) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee x_3) \wedge (\bar{x}_1 \vee \bar{x}_2)$

$x_1 = 0$ $\quad (0 \vee 0 \vee 1) \wedge (\bar{0} \vee \bar{0} \vee 1) \wedge (\bar{0} \vee \bar{0})$

$x_2 = 0$ $\quad (1) \wedge (1) \wedge (1) = 1$

$x_3 = 1$ $\quad$ Satisfiable

---

1. public class UF

```
UF(int N)                          // initializes N sites with integers names (0 to N-1)
void union(int p, int q)           // add connection between p and q
int find(int p)                    // component identifier for p (0 to N-1)
boolean connected(int p, int q)    // return true if p and q are in the same component
int count();                       // number of components
```

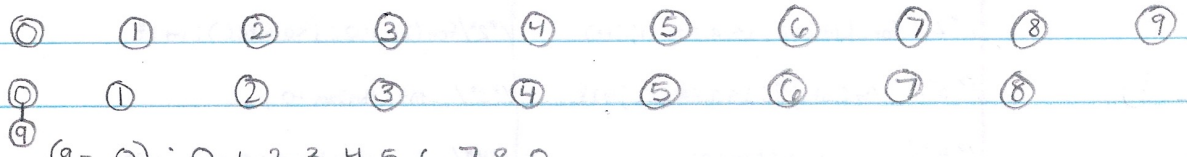
2. 1.5.1 ;

9-0	3-4	5-8	7-2	2-1	5-7	0-3	4-2
i :	0	1	2	3	4	5	6	7	8	9
(9-0)	0	1	2	3	4	5	6	7	8	0
(3-4)	0	1	2	4	4	5	6	7	8	0
(5-8)	0	1	2	4	4	8	6	7	8	0
(7-2)	0	1	2	4	4	8	6	2	8	0
(2-1)	0	1	1	4	4	8	6	1	8	0
(5-7)	0	1	1	4	4	1	6	1	1	0

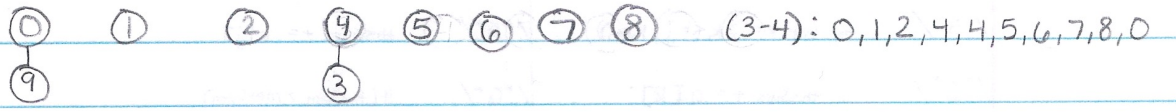
(0-3) 4 1 1 4 4 1 6 1 1 4

(4-2) 1 1 1 1 1 1 6 1 1 1

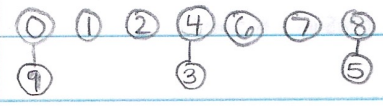
1.5.2: 9-0 3-4 5-8 7-2 2-1 5-7 0-3 4-2



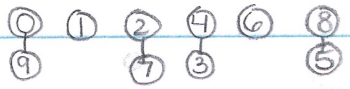
(9-0): 0,1,2,3,4,5,6,7,8,0



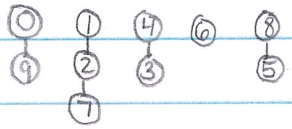
(3-4): 0,1,2,4,4,5,6,7,8,0



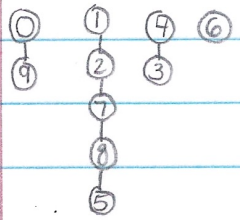
(5-8): 0,1,2,4,4,8,6,7,8,0



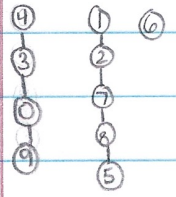
(7-2): 0,1,2,4,4,8,6,2,8,0



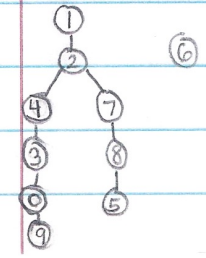
(2-1): 0,1,1,4,4,8,6,2,8,0



(5-7): 0,1,1,4,4,8,6,2,1,0



(0-3): 4,1,1,4,4,8,6,2,1,0



(4-2): 4,1,1,4,1,8,6,2,7,0

3. MaxSubSum1:

```
int maxSubSum1(const vector<int> & a){
/*1*/ int maxSum=0;
/*2*/ for (int i=0; i<a.size(); i++)
/*3*/   for(int j=i; j<a.size(); j++){
/*4*/     int thisSum=0;
/*5*/     for (int k=i; k<=j; k++)
/*6*/       thisSum += a[k];
/*7*/     if (thisSum > maxSum)
/*8*/       maxSum= thisSum;
        }
/*9*/ return maxSum;
}
```

MaxSubSum2:

```
int maxSubSum2(const vector<int> &a){
/*1*/ int maxSum=0;
/*2*/ for(int i=0; i<a.size(); i++){
/*3*/   int thisSum=0;
/*4*/   for (int j=i; j<a.size(); j++){
/*5*/     thisSum += a[j];
/*6*/     if(thisSum > maxSum)
/*7*/       maxSum = thisSum;
      }
    }
/*8*/ return maxSum;
}
```

4.
```
long gcd(long m, long n) //assuming m≥n{
/*1*/ while (n!=0){
/*2*/   long rem= m%n;
/*3*/   m=n;
/*4*/   n=rem; }
/*5*/ return m;
}
```