1. (10 Points) Consider the following pseudo-code and analyze the best and the worst-case running time of this by providing the form of the function the running time takes in each of the cases (Analyze statement-by-statement and use the length of the array A as n):

```
Algorithm 1 AdjacentDuplicates(A)
for $i$ = 0 $to\ A.length$ −2 do     n-1
if $A[i]$ == $A[i$ +1] then           1,but runs (n-2) times
return 1                            1,but may runs (n-2) times
end if
end for
return 0                            1
```

In the best case scenario, the first element satisfies the equation,so the running time could be O(1).
In the worst case scenario, none of elements of arrayA satisfy the condition, so the algorithm must iterate through the entire array, so the running time should be O(n).

2. (10 Points) Consider the following pseudo-code and analyze the best and the worst-case running time of this by providing the form of the function the running time takes in each of the cases (Analyze statement-by-statement and use the length of the array A as n):

```
Algorithm 2 Linear-Search(A,v)
i= NIL                              1
for j = 1 to A.length do             1, but runs n times
if A[j] = v then                    1, but may runs n times
i=j                                 1
return i                            1
end if
end for
return I                             1
```

In the best case scenario, the first element equals the value v ,so the running time could be O(1).
In the worst case scenario, none of elements of arrayA satisfy the condition, so the algorithm must iterate through the entire array, so the running time should be O(n).

3. (5 points) Find the upper bound for $f_5(n) = n^4 + 10n^2 + 5$. Prove your

answer by giving values for the constants c and n_0. Choose the smallest integral value possible for c.

$f(n) = n^4 + 10n^2 + 5 < cn^4$

$1 + 10/n^2 + 5/n^4 < c$

So $c >= 16$, $n0 = 1$

Suppose $c = 2$, then

$n^4 + 10n^2 + 5 < 2n^4$

$10n^2 + 5 < n^4$

$10 + 5/n^2 < n^2$

So, $c = 2$, $n_0 = 4$

4. (10 points) Find an asymptotically tight bound for $f(n) = 3n^3 - 2n$.

Prove your answer by giving values for the constants c_1, c_2, and n_0.

Choose the tightest integral values possible for c_1 and c_2.

$f(n) = 3n^3 - 2n \in \theta(n^3)$

The upper bound:

$3n^3 - 2n \leq 3n^3$ ($n \geq 1$)

The lower bound :

$3n^3 - 2n \geqslant n^3$

$2n^3 \geqslant 2n$ ($n \geq 1$)

So, $c1 = 1$, $c2 = 3$, $n0 = 1$, $3n^3 - 2n \in \theta(n^3)$

5. (5 points) Is $3n - 4 \in \Omega\, n^2$? Prove your response.

If $3n - 4 \in \Omega(n^2)$, then

$cn^2 < 3n - 4 < 3n$ $(\forall\, n > n_0)$

$cn^2 - 3n < 0$

$n(cn - 3) < 0$

$cn - 3 < 0$

$n < 3/c$

Since n can grow to infinity, it is impossible to find a positive constant c such that n is bounded above by the constant 3/c. Therefore, the c we need to find does not exist. Therefore $3n - 4 \notin \Omega(n^2)$.

6. (20 points) Express the complexity of the following functions with the most appropriate notation

```
int function1(int n) {
int count = 0;
for (int i=n/2;i<=n; i++){                    n/2
for (int j= 1;j<=n;j*=2) {                    log2n
  count++;
      }
   }
return count;
}
```

So the running time will be $(n/2) \log_2 n$, the time complexity is θ(nl0g(n))

```
int function2(int n) {
int count = 0;
for (int i=1; i*i*i<= n; i++){               ∛n
   count++;
   }
return count;
}
```

So the running time will be $\sqrt[3]{n}$, the time complexity is $\theta(\sqrt[3]{n})$

```
int function3(int n) {
int count =0;
for (int i= 1; i <= n; i++) {                   n
for (int j= 1; j <= n; j++) {                   n^2
for(int k=1;k<= n; k++) {                       n^3
     count++;
          }
      }
   }
return.count;
}
```

So the running time will be n^3, the time complexity is $\theta(n^3)$

```
int function4(int n) {
int count = 0;
for(int i= 1;i<= n; i++) {                    n
for (int j = 1; j <= n; j++){
     count++;
     break;                                    1
     }
}
return count;
}
```

The inner loop will execute only once because of break, so the running time will be n, the time complexity is θ(n)

```
int function5(int n) {
int count = 0;
for (int i= 1;i <= n; i++) {              n
   count++;
  }
for (int j= 1; j<= n; j++) {              n
   count++;
   }
return count;
}
```

The running time will be 2n, the time complexity is θ(n)

----

7. (10 points) Find an asymptotically tight bound for $f\ (n)$ = $n^2/2 - 7n$.

Prove your answer by giving values for the constants $c1$, $c2$, and $n0$.

Choose the tightest integral values possible for $c1$ and $c2$ [Hints: Solve

it for the smallest possible integer $n0$].

$0 \leqslant c_1(g(n)) \leqslant f(n) \leqslant c_2(g(n))$

$0 \leqslant c_1 n^2 \leqslant n^2/2 - 7n \leqslant c_2 n^2$

$c_1 \leqslant 1/2 - 7/n \leqslant c_2$

Suppose $c_2=1/2$, $c_1=1/6$, $n_0 = 21$, $n^2/2 - 7n \in \theta(n^2)$

8. (5*4 = 20 Points) Solve the following recurrence relations (using the backward substitution method with a demonstration of all the steps):

a) $x(n) = 3x(n-1)$ for $n > 1$, $x(1) = 4$

x(n-1) = 3x(n-1-1) x(n) = 3(3x(n-1-1)) = 9x(n-2)
x(n-2) = 3x(n-2-1) x(n) =9 (3x(n-2-1)) = 27x(n-3)
…
x(n) = 3^kx(n-k) n-k =1 x(n) =3^{n-1}x(1) = 4*3^{n-1}

----

b) $x(n) = x(n-1) + n$ for $n > 0$, $x(0) = 0$

x(n-1) = x(n-1-1) + (n-1) = x(n-2) + (n-1) x(n) = x(n-2) + n + (n-1)
x(n-2) = x(n-2-1) + (n-2) = x(n-3) + (n-2) x(n) = x(n-3) + n + (n-1)+ (n-2)
…
x(n) = x(n-i) + n + (n-1) +… + (n-(i-1))
n = i , x(n) = 0 + 1 +2 + … + n = (1+n)n/2 = O(n^2)

----

c) $x(n) = x(n/2) + n$ for $n > 1$, $x(1) = 1$

x(n/2) = x(n/4) + (n/2) x(n) = x(n/4) + (n/2) + n
x(n/4) = x(n/8) + (n/4) x(n) = x(n/8) + (n/4) + (n/2) + n
…
x(n) = x(n/2^k) + (n/2^{k-1}) +(n/2^{k-2}) + … + (n/$2^{k-\log n}$)
n = 2^k,k = logn,x(n) = x(n/$2^{\log n}$) +n(1/2 + …+ 1/2^{k-1}) = 1 + n = O(n)

----

d) $x(n) = x(n/3) + 1$ for $n > 1$, $x(1) = 1$

x(n/3) = x(n/9) + 1 x(n) = x(n/9) + 2
x(n/9) = x(n/27) + 1 x(n) = x(n/27) + 3
…
x(n) = x(n/3^k) + k
n = 3^k k =1ogn, x(n) = 1 + logn = O(logn)

9. (10 Points) For each function below, compute the recurrence relation for its running time and then use the Master Theorem to find its complexity by specifying the different terms of the term explicitly:

```
int f(int arr[], int n) {
if (n == 0) {
return 0;
}
intsum =0;
for (int j=0;j< n; ++j) {
sum += arr[j];                                          f(n) = n d=1
}
return f(arr,n / 2) + sum + f(arr, n / 2);              a = 2, b = 2
}
```

T(n) = 2T(n/2) + n, d = 1, $b^d = 2 = a$, so T(n) $\in$ θ ($n^d \log_b n$) = θ ($n \log_2 n$) = θ(nlogn)

```
void g(int n, int arrA[], int arrB[]){
if(n == 0){
return;
}
for(int i =0;i <n; ++i) {
  (for(int j=0;j<n; ++j){
    arrB[j] += arrA[i];                                f(n) = n^2 d=2
    }
    }
g (n / 2，arrA,arrB);                                  a = 1, b = 2
}
```

T(n) = T(n/2) + n^2, d = 2, $b^d = 4 > a$, so T(n) $\in$ θ (n^d) = θ (n^2)

10. (2*5 = 10 points) Use the Master Theorem to find the complexity of each of the following recurrence relations (showall the steps and the values of different terms to apply the theorem):

a) $T(n) = 4T(n/2) + n^2$

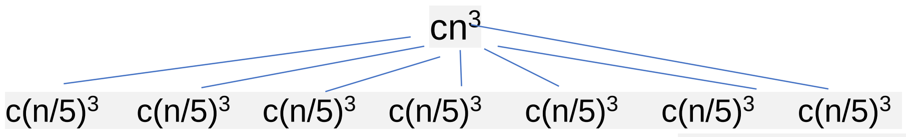
a = 4, b = 2, d = 2, $a = b^d$, so T(n) $\in$ θ ($n^d \log_b n$) = θ ($n^2 \log_2 n$) = θ ($n^2 \log n$)

b) $T(n) = 3T(n/3) + \sqrt{n}$

a = 3, b = 3, d = 1/2, $a > b^d$, so T(n) $\in$ θ ($n^{\log_b a}$) = θ (n)

11. (2*10 = 20 points) Use the Recursion tree method to find appropriate notation for the complexity of each of the following recurrence relations (draw the recursion tree and show all the steps to your solution):

a) $T(n) = 7T(n/5) + cn^3$


cn^3
$c(n/5)^3$ $c(n/5)^3$ $c(n/5)^3$ $c(n/5)^3$ $c(n/5)^3$ $c(n/5)^3$ $c(n/5)^3$

$7\,c(n/5)^3 = cn^3(7/5^3)$

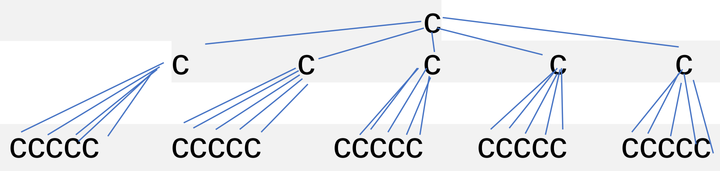
…………… $7^2\,c(n/5^2)^3 = cn^3(7/5^3)^2$

$7^3\,c(n/5^3)^3 = cn^3(7/5^3)^3$

T(1) T(1) T(1) T(1) … … … T(1) $7^k\,c(n/5^k)^3 = cn^3(7/5^3)^k$

$T(n) = \sum_{k=0}^{log5n} cn^3(7/5^3)^k = cn^3\sum_{k=0}^{log5n} (7/5^3)^k \in \theta(n^3)$

b) $T(n) = 5T(n-6) + c$



5c

5^2c

…………………..

T(0) T(0) T(0) T(0) … … … T(0) 5^kc

$T(n) = 5c + 5^2c + \ldots + 5^kc \in \theta(5^{n/6})$

C