HW 1 Written Problem 1

(a)

```
void f1(int n)
{
    int i = 2;
    while (i<n){
        /* O(1) time task */
        i = i*i;
    }
}
```

$2 \rightarrow 4 \rightarrow 16 \rightarrow 256.$

This function squares itself until it is no longer smaller than integer n. Thus, if the while loop can be run m times: $2^{(2^m)} \geq n$. will always hold.

f1's runtime is based on the number of times the while loop is run, so we must find m's value in terms of n. We can achieve this by placing both sides in $\log_2()$.

$$\log_2 2^{(2^m)} \geq \log_2 n$$

$$2^m \geq \log_2 n \quad \text{----- repeat}$$

$$m \geq \log_2(\log_2 n)$$

Thus, Big θ runtime is equal to $\theta(\log_2(\log_2 n))$

(b)

```
void f2 (int n)
{
    for (int i = 1; i <= n; i++) {                       'n' times
        if (i % (int) sqrt(n)) == 0) {                   '√n' times
            for (int k = 0; k < pow(i, 3); k++) {        '(c·√n)^3' times
                /* O(1) time task */
            }
        }
    }
}
```

The outer-most loop runs a total of 'n' times.

The if-statement after will occur only when 'i' is equal to sqrt(n). ($i == \sqrt{n}$). So this statement will run '$\sqrt{n}$' times.

The inner for-loop after the if-statement will occur when integer 'k' is smaller than i^3. ($k < i^3$). Because it increments up by 1 at a time, it will run 'i^3' times. But this for loop only runs when the if-statement above it is true, meaning that 'i' is a multiple of $\sqrt{n}$. In that case, k runs from $0 \sim (c \cdot \sqrt{n})^3$ times each iteration of the for loop.

where c is equal to the amount of multiples of $\sqrt{n}$ can have without surpassing n.

$n/\sqrt{n} = \sqrt{n}$ times $= c$.

$$\sum_{c=1}^{\sqrt{n}} (c \cdot \sqrt{n})^3 = \sqrt{n}^3 \cdot \sum_{c=1}^{\sqrt{n}} c^3 = \sqrt{n}^3 \left(\frac{\sqrt{n}(\sqrt{n}+1)}{2}\right)^2$$

So, the inner-loop runs for $n^{1.5} \cdot n^2 = n^{3.5}$

$\therefore$ Big-Θ runtime is $\Theta(n^{3.5})$.

(c)
```
for (int i=1; i<=n; i++) {
    for (int k=1; k<=n; k++) {
        if ( A[k] == i ) {
            for (int m=1; m<=n; m=m+m) {
                /* O(1) time task */
            }
        }
    }
}
```

In the first for loop, there are 'n' iterations.
The next for loop (nested) is also ran 'n' times.
The 'if-statement' is only ran when the value of A[k] is equal to 'i'. Because we do not know the contents of array A, we must assume the worst-case scenario in which it runs all 'n' times.
Thus, the conditional check occurs once for each iteration of 'k'.
In the third loop, m is doubled each iteration. So to solve the total number of iterations, we must solve $2^k \leq n$ → place $\log_2( )$ on both sides to get $k = \log_2 n$. This loop runs $\log_2 n$ times.
Thus we get $n * n * \log_2 n = n^2 \log_2 n$

$\therefore \Theta(n^2 \log_2 n)$

(d)

The first for loop iterates 'n' times. On the 10th iteration of the loop, a new array of size 1.5x its current size is created called b. Then, all elements of the original array are copied into the newly allocated array. which takes 'n' time. Because the outer loop is $\theta(n)$ and the inner copying processes take $\theta(n)$. However, the resizing operation only occurs when 'i' reaches "size", so this does not affect the time complexity. ∴ $\underline{\theta(n)}$.

HW 1 Written Problem 2

```
struct Node {
    int val;
    Node* next;
};

Node* llrec(Node* in1, Node* in2)
{ if (in1 == nullptr) {
        return in2;
    }
    else if (in2 == nullptr) {
        return in1;
    }
    else {
        in1->next = llrec(in2, in1->next);
        return in1;
    }
}
```

(a)

First iteration:

in1 = 1, 2, 3, 4 ↔ in2 = 5, 6.

neither null.

in1 → next = llrec(5, 2)

in1: 5 in1→next: 6
in2: 2 in2→next: 3

= in1 → next = llrec(2, 6)

in1: 2
in1→next: 3
in2: 6

= in1→next = llrec(6, 3)

continued

in1: 6 in1→next: null
in2: 3 in2→next: 4

in1→next = llrec(3, null).

in1: 3, in1→next: 4.
in2: null.

In this case, because the input for Node* in2 is a null ptr, we return in1. which is 3.

Therefore, we get:

1 → 5 → 2 → 6 → 3

(b)

in1 = nullptr, in2 = 2.

↳ follows first if-statement (case in1 == nullptr) and return in2, which is one element, 2.