1.
```
void splice( Node *a, Node *b ) {
    if( a==NULL || b==NULL ) return;
    Node *temp = b->next;
    b->next = a;
    while( a->next ) a = a->next;
    a->next = temp;
}
```

2. (a) For each street, keep the list of house numbers on the street in sorted order in an array A. Let n be the length of the list.

   To **find** the neighbors of a given house number X:

   i. Use binary seaarch to find the index i of house number X in the array A.

   ii. Output the house numbers preceding ($A[i-1]$) and following ($A[i+1]$) the house number X in the array A.

   This takes $\Theta(\log n)$ worst-case time.

   To **insert** a new house after a given house number X and return its new house number:

   i. Use binary search (or linear search) to find the index i of house number X in the array A.

   ii. Calculate the new house number as $(A[i] + A[i+1])/2$ (or, if X is the last house on the street, use $A[i] + 1$).

   iii. Add the new house number into the array following $A[i]$. (This requires setting $A[j] = A[j-1]$ for $j = n..i+1$. It might also require resizing the array.)

   This takes $\Theta(n)$ worst-case time.

   (b) For each street, keep the list of house numbers on the street in sorted order in the linked list L. Let n be the length of the list.

   To **find** the neighbors of a given house number X:

   i. Use linear search to find a pointer p to the *predecessor* of the node containing house number X in the linked list L.

   ii. Output the house numbers preceding (`p->houseNum`) and following (`p->next->next->houseNum`) the house number X in the linked list L.

   This takes $\Theta(n)$ worst-case time.

   To **insert** a new house after a given house number X and return its new house number:

   i. Use linear search) to find a pointer p to the node containing house number X in the linked list L.

   ii. Calculate the new house number as `(p->houseNum + p->next->houseNum)/2.0` (or, if X is the last house on the street, use `p->houseNum + 1`.

   iii. Add the new house number into the linked list following p (by creating a new node containing the new house number whose next pointer equals `p->next` and then setting `p->next` to be the new node's address.)

This takes $\Theta(n)$ worst-case time.

(c) I would use the array. It's easy enough to implement even though it uses binary search and resizeable arrays, **find** is fast, and linear time for **insert** is a small price to pay for simple code. Note: You could get $O(\log n)$ time for both **find** and **insert** using, for example, a balanced search tree or expected $O(\log n)$ time for both operations using a skiplist. It would be difficult to use a hash table for this problem because it requires finding the neighbors of a given house number.

3\.

(a) $5n$ (b) n^2 (c) 3 (d) -1 (e) n^{12}

4\.

$n \log n$	$n^{2 \lg n}$	$\log n$	2^{2^n}	$\sqrt{n}$	n^n	n	2^n	$\sum_{i=1}^{n} i^3$	$\lg(n!)$
4 or 5	7	1	10	2	9	3	8	6	4 or 5

5\. (a) $\Theta(1)$

(b) $\Theta(n^2)$

(c) $T(n) = 2^c \sum_{i=0}^{n} 2^i = 2^c \cdot (2^{n+1} - 1) \in \Theta(2^n)$

(d) $T(n) = c \sum_{i=1}^{n} (n^2 - i^2 + 1) \le c \sum_{i=1}^{n} n^2 = cn^3$, so $T(n) \in O(n^3)$

$$T(n) = c \sum_{i=1}^{n} (n^2 - i^2 + 1) \ge c \sum_{i=1}^{n/2} (n^2 - i^2 + 1)$$
$$\ge c \sum_{i=1}^{n/2} (n^2 - (n/2)^2 + 1) = cn \cdot (3/4n^2 + 1) \in \Omega(n^3)$$

(e)
$$\begin{aligned} T(n) &= 2T(n-2) + 4 \\ &= 2(2T(n-4) + 4) + 4 = 4T(n-4) + 4 \cdot 3 \\ &= 4(2T(n-6) + 4) + 4 \cdot 3 = 8T(n-6) + 4 \cdot 7 \\ &= 2^k T(n - 2k) + 4(2^{k+1} - 1) \\ &= 2^{n/2} T(0) + 4(2^{n/2+1} - 1) = 2^{n/2} + 8 \cdot 2^{n/2} - 4 \\ &= 9 \cdot 2^{n/2} - 4 \in \Theta((\sqrt{2})^n) \end{aligned}$$

(f) If we restrict our attention to $n = 2^k$ for integer k, then we can ignore the ceiling in the recurrence and by the substitution method we can see that $T(n) = 3 \lg n + 1$:

$$T(n) = T(n/2) + 3 = T(n/4) + 3 + 3 = T(n/2^k) + 3k = 1 + 3 \lg n.$$

If n is not an integral power of two, then let $k = \lceil \lg n \rceil$ so $2^{k-1} < n \le 2^k$. Since $T(n)$ is a non-decreasing function, $T(2^{k-1}) < T(n) \le T(2^k)$. Since 2^{k-1} and 2^k are powers of two, (1) $T(n) > T(2^{k-1}) = 3(k-1)+1 = 3(\lceil \lg n \rceil - 1)+1 \ge 3(\lg n - 1)+1 = 3 \lg n - 2 \ge 2 \lg n$ for $n \ge 4$; and (2) $T(n) \le T(2^k) = 3k+1 = 3(\lceil \lg n \rceil)+1 \le 3(\lg n+1)+1 = 3 \lg n + 4 \le 4 \lg n$ for $n \ge 16$. So $T(n) \in \Theta(\log n)$. (In fact, $T(n) = 3\lceil \lg n \rceil + 1$.)

6. (a) Ω: $54n^3 + 17 \geq 54n^3$ for all $n \geq 1$, so we can set $c = 54$ and $n_0 = 1$ in the definition of Ω-notation.

   O: $54n^3 + 17 \leq 71n^3$ for all $n \geq 1$, so we can set $c = 71$ and $n_0 = 1$ in the definition of big-O-notation.

   (b) It's enough to show that $54n^3+17$ is not in $O(n^2)$, as Θ-notation requires both conditions. *Proof by contradiction.* Assume $54n^3+17 \in O(n^2)$. Then by definition, there are constants $c > 0$ and $n_0 > 0$ such that for all $n \geq n_0$, $54n^3 + 17 \leq cn^2$. Divide this by n^2, and we have $54n + 1 \leq 54n + 17/n^2 \leq c$. No matter how large constant c is, this will fail to be true for large values of n (larger than $(c-1)/54$). A contradiction.
   Hence, $54n^3 + 17 \notin O(n^2)$.

   (c) O: Let $a = \max\{a_0, a_1, \ldots, a_d\}$, then

$$T(n) = \sum_{i=0}^{d} a_i n^i \leq \sum_{i=0}^{d} a n^i \leq a \sum_{i=0}^{d} n^d = a(d+1)n^d.$$

   Thus, $T(n) \leq cn^d$ for all $n \geq n_0$ where $c = a(d+1)$ and $n_0 = 1$, so $T(n) \in O(n^d)$.
   Note: Even if d is not a constant but is a function of n, $T(n) \in O(n^d)$. To show this, we can use the formula for the sum of a geometric series:

$$T(n) = \sum_{i=0}^{d} a_i n^i \leq \sum_{i=0}^{d} a n^i = a\frac{n^{d+1}-1}{n-1} \leq a\frac{n^{d+1}}{n/2} \text{ (for } n \geq 2) = 2an^d.$$

   Ω: Let $b = \max\{|a_0|, |a_1|, \ldots, |a_d|\}$. Now b is positive, even though some of the a_i might be negative, and $-b \leq a_i$ for all i. So,

$$T(n) = \sum_{i=0}^{d} a_i n^i \geq a_d n^d - \sum_{i=0}^{d-1} bn^i \geq a_d n^d - bdn^{d-1} = \frac{a_d}{2}n^d + \frac{a_d}{2}n^d - bdn^{d-1} \geq \frac{a_d}{2}n^d$$

   as long as $\frac{a_d}{2}n^d \geq bdn^{d-1}$. This happens when $n \geq 2bd/a_d$. Thus, $T(n) \geq cn^d$ for all $n \geq n_0$ where $c = a_d/2$ and $n_0 = 2bd/a_d$, so $T(n) \in \Omega(n^d)$.
   Note: Even if d is not a constant, $T(n) \in \Omega(n^d)$ by again using the formula for the sum of a geometric series.

7. (a) Recurrence:

$$T(n) = \begin{cases} c & \text{if } n = 0 \\ T(n/2) + d & \text{if } n \geq 1 \end{cases}$$

   where $c = 1$ and $d = 4$, if we wish to count the number of executed lines. Using the substitution method, we get $T(n) = T(n/2^k) + dk = T(1) + d\lg n$ for $n = 2^k$. Since $T(1) = c + d$ is a constant, $T(n) \in \Theta(\log n)$.

   (b) Let's count the lines. The inner loop iterated i times and the outer loop n times. We have $T(n) = 2 + \sum_{i=0}^{n-1}(4 + 3i) \in \Theta(n^2)$.