Q1.

```python
def karatsuba_polynomial_multiply(A, B):
    n = len(A)

    # Base case
    if n == 1:
        return [A[0] * B[0]]

    # Split the polynomials A and B into two halves
    half = n // 2
    A0, A1 = A[:half], A[half:]
    B0, B1 = B[:half], B[half:]

    # Recursive calls to compute the three products
    P0 = karatsuba_polynomial_multiply(A0, B0)
    P2 = karatsuba_polynomial_multiply(A1, B1)

    # Compute (A0 + A1) and (B0 + B1)
    A0_A1_sum = [A0[i] + A1[i] for i in range(half)]
    B0_B1_sum = [B0[i] + B1[i] for i in range(half)]

    # Recursive call to compute (A0 + A1) * (B0 + B1)
    P1 = karatsuba_polynomial_multiply(A0_A1_sum, B0_B1_sum)

    # Calculate the coefficients of the result polynomial
    result = [0] * (2 * n - 1)

    # Combine the results using the Karatsuba trick
    for i in range(n - 1):
        result[i] += P0[i]
        result[i + half] += P1[i] - P0[i] - P2[i]
        result[i + 2 * half] += P2[i]

    return result
```

To analyze the running time,
we can use the recurrence relation
$T(n) = 3T(n/2) + O(n)$
Using master theorem where $a=3, b=2$,
the running time of this algorithm is $O(n^{\log_2 3})$

Q2.

```
power(x,y)
{
    if(y==0)
        return 1;
    temp = power(x,y/2)
    if(y%2 == 0)
        return (temp*temp)
    else
        return(x*temp*temp)
}
```

Q3.

(a) The master theorem is applicable to recurrence relations of the form
: $T(n) = aT(n/b) + f(n)$.
In this situation, a and b are not constants.

(b) If $S(n) = T(n)/n$, $T(n) = n \cdot S(n)$

$T(n) = \sqrt{n} \cdot T(\sqrt{n}) + O(n)$

$n \cdot S(n) = \sqrt{n} \cdot \sqrt{n} \cdot S(\sqrt{n}) + O(n)$

$\therefore S(n) = S(n^{1/2}) + O(1)$

(c) If $n = 2^m$, $R(m) = S(2^m) = S(n)$, $\sqrt{n} = 2^{m/2}$

$S(n) = S(n^{1/2}) + O(1)$ $\left(S(\sqrt{n}) = S(2^{m/2}) = R\left(\frac{m}{2}\right) \right)$

$R(m) = R\left(\frac{m}{2}\right) + O(1)$

(d) Now using master theorem, $a=1, b=2, d=0$.

$a = b^d$ so we have $R(m) = \log m$

we also know that $m = \log_2 n$, so we have $S(2^m) = \log m$

$S(2^m) = S(n) = \log(\log_2 n)$

and we know that $S(n) = T(n)/n$

$T(n)/n = \log(\log_2 n)$ $\therefore T(n) = n \log(\log_2 n) = O(n \log(\log n))$

Q4. Using master theorem

$T(n) = a \cdot T(n/b) + O(n^d) \rightarrow T(n) = \begin{cases} O(n^d) & \text{if } a < b^d \\ O(n^d \log n) & \text{if } a = b^d \\ O(n^{\log_b a}) & \text{if } a > b^d \end{cases}$

① $T(n) = 9T(n/4) + 1$

$a=9, b=4, d=0, a > b^d \rightarrow T(n) = O(n^{\log_4 9})$

② $U(n) = 6U(n/4) + n^{1.5}\log^2 n$

$n^{1.5}\log^2 n = \Omega(n^{1.5})$

$a=6, b=4, d=1.5, a < b^d \rightarrow U(n) = O(n^{1.5})$

③ $V(n) = 8V(n/4) + n^{1.5}$

$a=8, b=4, d=1.5, a = b^d \rightarrow V(n) = O(n^{1.5}\log n)$

$\log_4 9 \approx 1.584 \quad \therefore U(n) < V(n) < T(n)$

Q5.

For any column vector u of length n, let u_1 denote the column vector of length $n/2$ consisting of the first $n/2$ consisting of the first $n/2$ coordinates of u. Similarly, define u_2 to be the vector of the remaining coordinates. Note then that

$$(H_k v)_1 = H_{k-1} v_1 + H_{k-1} v_2 = H_{k-1}(v_1 + v_2)$$

and

$$(H_k v)_2 = H_{k-1} v_1 - H_{k-1} v_2 = H_{k-1}(v_1 - v_2)$$

Recursion 1 : This shows that we can find $H_k v$ by calculating

$$v_1 + v_2 \quad \text{and} \quad v_1 - v_2$$

and recursively computing

$$H_{k-1}(v_1 + v_2) \quad \text{and} \quad H_{k-1}(v_1 - v_2)$$

Recursion2 : We need only to compute two subproblems

$$H_{k-1} v_1 \quad \text{and} \quad H_{k-1} v_2$$

and combining the solutions of the two subproblems using addition and subtraction, both taking $O(n)$ time.

The running time of this algorithm by both recursions above can be described as

$$T(n) = 2T(n/2) + O(n)$$

where the linear term is the time taken to perform the addition and the subtraction.

This has solution

$$T(n) = O(n \log n)$$

by the master theorem.