1.

```
C/C++
void f1(int n)
{
    int i=2;
    while(i < n){
        /* do something that takes O(1) time */
        i = i*i;
    }
}
```

Θ(logn)
Code repeats until i is greater than n, and i is squaring itself every cycle. This leads to the code repeating for logn times.

2.

```
C/C++
void f2(int n)
{
    for(int i=1; i <= n; i++){
        if( (i % (int)sqrt(n)) == 0){
            for(int k=0; k < pow(i,3); k++) {
              /* do something that takes O(1) time */
            }
        }
    }
}
```

Outer loop happens for Θ(n) time. It repeats n times. The innermost for statement depends on the square root of n, so n is the primary source here for that loop. n*n is n^2 so the answer is Θ(n^2)

3.

```
C/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){
        // do something that takes O(1) time
        // Assume the contents of the A[] array are not changed
      }
    }
  }
}
```

Most outer loop happens Θ(n) times. The next nested loop happens Θ(n) times as well. The most inner loop happens for Θ(logn) times. Multiplying these would get Θ(n^2 * logn) which is our answer.

4.

C/C++

```
int f (int n)
{
  int *a = new int [10];
  int size = 10;
  for (int i = 0; i < n; i ++)
     {
        if (i == size)
          {
             int newsize = 3*size/2;
             int *b = new int [newsize];
             for (int j = 0; j < size; j ++) b[j] = a[j];
             delete [] a;
             a = b;
             size = newsize;
          }
        a[i] = i*i;
     }
}
```

Most outer loop happens for Θ(n) times. If the if condition is initiated, which only happens if n is greater than 10, then the size is multiplied by 1.5. There is no nested loops though, and the resizes aren't too great, so Θ(n) is the final runtime.