For the following exercise, find the exact count of the total number of additions, multiplications, comparisons, and assignments being done by each algorithm. Then determine the complexity.

Determine the complexity of the following implementations of the algorithms for adding, multiplying, and transposing $n \times n$ matrices:

```
for (i = 0; i < n; i++)
    for (j = 0; j < n; j++)
        a[i][j] = b[i][j] + c[i][j];

for (i = 0; i < n; i++)
    for (j = 0; j < n; j++)
        for (k = a[i][j] = 0; k < n; k++)
            a[i][j] += b[i][k] * c[k][j];

for (i = 0; i < n - 1; i++)
    for (j = i+1; j < n; j++) {
        tmp = a[i][j];
        a[i][j] = a[j][i];
        a[j][i] = tmp;
    }
```

For the following exercise, find the exact count of the total number of additions, multiplications, comparisons, and assignments being done by each algorithm. Then determine the complexity.

Find the computational complexity for the following four loops:

a.
```
for (cnt1 = 0, i = 1; i <= n; i++)
    for (j = 1; j <= n; j++)
        cnt1++;
```

b.
```
for (cnt2 = 0, i = 1; i <= n; i++)
    for (j = 1; j <= i; j++)
        cnt2++;
```

c.
```
for (cnt3 = 0, i = 1; i <= n; i *= 2)
    for (j = 1; j <= n; j++)
        cnt3++;
```

d.
```
for (cnt4 = 0, i = 1; i <= n; i *= 2)
    for (j = 1; j <= i; j++)
        cnt4++;
```

For the following implementations of the linear search and the binary search, find the exact count of the total number of additions, multiplications, comparisons, and assignments being done by each algorithm. Then determine the complexity. Do this for both the best case scenario and the worst case scenario.

```
template <class T> int binarySearch(T A[], int left, int right, T target) {
  while (left <= right) {
    int mid = (right + left) / 2;

    if (A[mid] == target)
      return mid;
    else if (A[mid] > target)
      right = mid - 1;
    else
      left = mid + 1;
  }

  return -1;
}

template <class T> int linearSearch(T A[], int size, T target) {
  for (int i = 0; i < size; i++)
    if (A[i] == target)
      return i;

  return -1;
}
```

For the following implementations of the bubble sort, selection sort, and insertion sort, find the exact count of the total number of additions, multiplications, comparisons, and assignments being done by each algorithm. Then determine the complexity. Do this for both the best case scenario and the worst case scenario.

```
template <class T> void bubble(T A[], int size) {
  for (int i = 0; i < size - 1; i++) {
    for (int j = 0; j < size - i - 1; j++) {
      if (A[j] > A[j + 1]) {
        T temp = A[j];
        A[j] = A[j + 1];
        A[j + 1] = temp;
      }
    }
  }
}

template <class T> void insertion(T A[], int size) {
  for (int i = 0; i < size; i++) {
    int j = 0;
    T val = A[i];
    for (j = i; j > 0 && A[j - 1] > val; j--)
      A[j] = A[j - 1];

    A[j] = val;
  }
}

template <class T> void selection(T A[], int size) {
  int minindex;

  for (int i = 0; i < size; i++) {
    minindex = i;
    for (int j = i; j < size; j++)
      if (A[j] < A[minindex])
```

```
31          minindex = j;
32
33      T val = A[i];
34      A[i] = A[minindex];
35      A[minindex] = val;
36    }
37  }
```

Prove the following statements:

a. $\Sigma^{n}_{i=1}\, i^2$ is $O(n^3)$ and more generally, $\Sigma^{n}_{i=1}\, i^k$ is $O(n^{k+1})$.

b. $an^k/\lg n$ is $O(n^k)$, but $an^k/\lg n$ is not $\Theta(n^k)$.

c. $n^{1.1} + n \lg n$ is $\Theta(n^{1.1})$.

d. 2^n is $O(n!)$ and $n!$ is not $O(2^n)$.

e. 2^{n+a} is $O(2^n)$.

f. 2^{2n+a} is not $O(2^n)$.

g. $2^{\sqrt{\lg n}}$ is $O(n^a)$.