Problem 1. Select the worst-case running time of each function.

```c
bool search(int x, int* A, int n)
{
    if (linear_search(x, A, n))
        return true;
    insertion_sort(A, n);
    return binary_search(x, A, n);
}

bool search_n_sort(int* A, int n)
{
    for (int x = 1; x <= n; ++x)
        if (linear_search(x, A, n))
            return true;
    insertion_sort(A, n);
    return false;
}

bool sort_n_search(int* A, int n)
{
    insertion_sort(A, n);
    for (int x = 1; x <= n; ++x)
        if (binary_search(x, A, n))
            return true;
    return false;
}

void very_sort(int* A, int n)
{
    for (int i = 0; i < n; ++i)
        insertion_sort(A, n);
}
```
Problem 2. Select the best-case running time of each function.

```cpp
bool search(int x, int* A, int n) {
    if (linear_search(x, A, n)) return true;
    insertion_sort(A, n);
    return binary_search(x, A, n);
}

bool search_n_sort(int* A, int n) {
    for (int x = 1; x <= n; ++x)
        if (linear_search(x, A, n)) return true;
    insertion_sort(A, n);
    return false;
}

bool sort_n_search(int* A, int n) {
    insertion_sort(A, n);
    for (int x = 1; x <= n; ++x)
        if (binary_search(x, A, n)) return true;
    return false;
}

void very_sort(int* A, int n) {
    for (int i = 0; i < n; ++i)
        insertion_sort(A, n);
}
```
Problem 3. Fill in the blanks below to perform asymptotic analysis of the worst-case running time of the following function that searches in an unsorted array:

```c
bool search(int x, int* A, int n)
{
    if (n < 1)
        return false;
    if (A[0] == x)
        return true;
    return search(x, &(A[1]), n-1);
}
```

The recurrence relation \( f(n) = \text{function of } f(n-1) \) with \( f(0) = \Theta(\text{function of } n) \) describes the worst-case running time of `search`.

Using repeated substitution once on the recurrence relation gives \( f(n) = \text{function of } f(n-2) \).

A closed form for the worst-case running time of `search` is \( f(n) = \Theta(\text{function of } n) \).
Problem 4. Complete the following recursive template implementation of insertion sort.

template <typename T>
void insertion_sort(vector<T> &A)
{
    if (A.size() _____ 0)
        return;

    T last = A.back();
    A.____;
    insertion_sort(____);

    int i = A.length()-1;
    while (i > 0 _____ A[i] < A[i-1])
    {
        T tmp = A[i];
        A[i] = A[i-1];
        A[i-1] = tmp;
    }
}

Problem 5. Fill in the blanks below based on the insertionsort function in Problem 4, where \( n \) is the length of the input vector A.

A call to the vector class’s pop_back method takes \( \Theta(______) \) time.

The recurrence relation \( f(n) = \frac{function of f(n-1)}{function of n} \) with \( f(0) = \Theta(______) \) describes the worst-case running time of insertion_sort.

Using repeated substitution once on the recurrence relation gives \( f(n) = \frac{function of f(n-2)}{function of n} \).

A closed form for the worst-case running time of insertion_sort is \( f(n) = \Theta(______) \).