Exercise 2

```
int foo(int N) {
int result = 0;
for (int i=0; i<N; i++)
result++;
for (int j=0; j<1000000; j++) result+=j;
  return result;
}
```

*The time complexity is O(N). The first loop executes N times, and the rest of the method is O(1). Overall order: O(N) * O(1) = O(N)*

Exercise 3

```
int bar(int N) {
int result = 1;
for (int i=1; i<N; i*=2)
result+=2;
  return result;
}
```

The time complexity is O(log N). because i increases exponentially, the number of loops will decrease inversely (i.e. the number loops required for i to be greater than or equal to N increases logarithmically as N increases, because i increases exponentially).

Exercise 4

Assume a binary search is performed on the following array of integers:
{1, 14, 15, 24, 55, 59, 73, 90, 94, 99}
Trace through each iteration of the algorithm, writing the number that will be the middle element and the left and right bounds (indexes), when searching for the number 73.

Important Notes:
- A binary search requires a list to be sorted. This is already done in this particular example, but should be kept in mind.
- Index used to identify the middle values is calculated via integer division, which always round down to the largest integer value

Shirley Ramirez – Programming Fundamentals – Lab 6 Response (Ex 2-5)

Iteration 1:

{1, 14, 15, 24, **55**, 59, 73, 90, 94, 99}

Left bound: 1

Right bound: 99

Midpoint: 55

Iteration 2:

{59, 73, **90**, 94, 99}

Left bound: 59

Right bound: 99

Midpoint: 90

Iteration 3:

{**59**, 73}

Left bound: 59

Right bound: 73

Midpoint: 59

Iteration 4:

{**73**}

Left bound: 73

Right bound: 73

Midpoint: 73

Exercise 5

Trace the execution of the insertion and selection sort algorithms when executed on the following array of integers:
{1, 29, 14, 15, 94}
Show how the array will look like after each iteration of the outer loop.

Selection Sort:
{1, 29, 14, 15, 94}

Iteration 1: *Start with first item (index 0) in the list, and scan to the right to find the smallest value. There is no smaller value, so value remains the same (1).*

Iteration 2*: Move to the next position (index 1). In this example the value at this position is 29. Scan to the right to look for the smallest value and switch it with 29. In this example the smallest value is 14, so values 29 and 14 switch positions.*

Iteration 3: *Move to next position (index 2). The value at this position is 29. Scan to the right to look for the smallest value to switch places with it. In this example the smallest values is 15, so values 29 and 15 switch positions.*

Iteration 4: *Move to the next position (index 3). The value at this position is 29. Scan to the right to look for the smallest values to switch places with it. 29 is the smallest value, so it remains in the same position.*

Result: *Index has reached the end and the array is now sorted.*

Iteration	Index 0	Index 1	Index 2	Index 3	Index 4		Index 0	Index 1	Index 2	Index 3	Index 4
1	1	29	14	15	94	→	1	29	14	15	94
2	1	29	14	15	94	→	1	14	29	15	94
3	1	14	29	15	94	→	1	14	15	29	94
4	1	14	15	29	94	→	1	14	15	29	94
Result	1	14	15	29	94						

Shirley Ramirez – Programming Fundamentals – Lab 6 Response (Ex 2-5)

Insertion Sort:

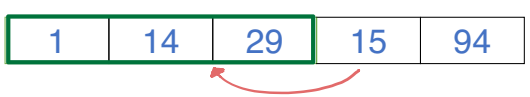
{1, 29, 14, 15, 94}

1	29	14	15	94

Iteration 1: *First item considered a sorted sublist of length 1.*

Iteration 2: *second item (29) inserted into the sorted sublist. No shifting needed since 29 is greater than 1.*

Iteration 3: *third item (14) is inserted into sorted sublist (of 1, 29). 29 shifts to the right because 1 < 14 <29*



Iteration 4: *Fourth item (15) is inserted into sorted sublist (1, 14, 29). 29 shifts to the right because 14 < 15 < 29*

Iteration 5: *Last item does not need to be inserted into a new position because it is greater than the rest of the values.*

1	14	15	29	94