Question 1&2:

```c
#include <stdio.h>
#include <stdlib.h>
#include<time.h>

//Inser function Prototype
int *insert( int *array, int length, int index, int value);

//Main function

int main(){
    const int INSERTS_PER_READING =1000;
    int *array=NULL;
    int length=0;

    srand(time(NULL)); //randon generation

    printf("Array length\tSeconds per insert\n");
    for(int i=0; i<60;i++){
        clock_t start_time=clock();
        for (int j=0; j<INSERTS_PER_READING;j++){
            int index=rand() % (length+1);
            int value= rand();
            array=insert(array, length, index, value);
            if (!array){
                printf("Memory allocation error!");
                exit(1);
            }
            length++;
        }
        clock_t stop_time=clock();
        double time_pre_insert=(double)(stop_time -
start_time)/CLOCKS_PER_SEC/INSERTS_PER_READING;
        printf("%d\t%.6f\n",length, time_pre_insert);
    }
    free(array);
    return 0;
}

//Insert function
int *insert(int *array , int length, int index, int value){
    if(length==0){                                          //O(1)
        int *newArray = (int*) malloc(sizeof(int));         //O(1)
        if(!newArray)                                       //O(1)
```

```
            return NULL;//handle memory allocation failure //O(1)
        newArray[0]=value;                                  //O(1)
        return newArray;                                    //O(1)
    } else{
        int *newArray =(int *)malloc((length+1)*sizeof(int));//O(1)
        if (!newArray)
            return NULL;                                    //O(1)
        for (int i=0; i<index;i++)
            newArray[i]=array[i];                           //O(n)
            newArray[index]=value;                          //O(1)
        for(int i= index; i<length; i++)
            newArray[i+1]==array[i];                        //O(n)
        free(array);                                        //O(1)
        return newArray;                                    //O(1)
    }
}
```

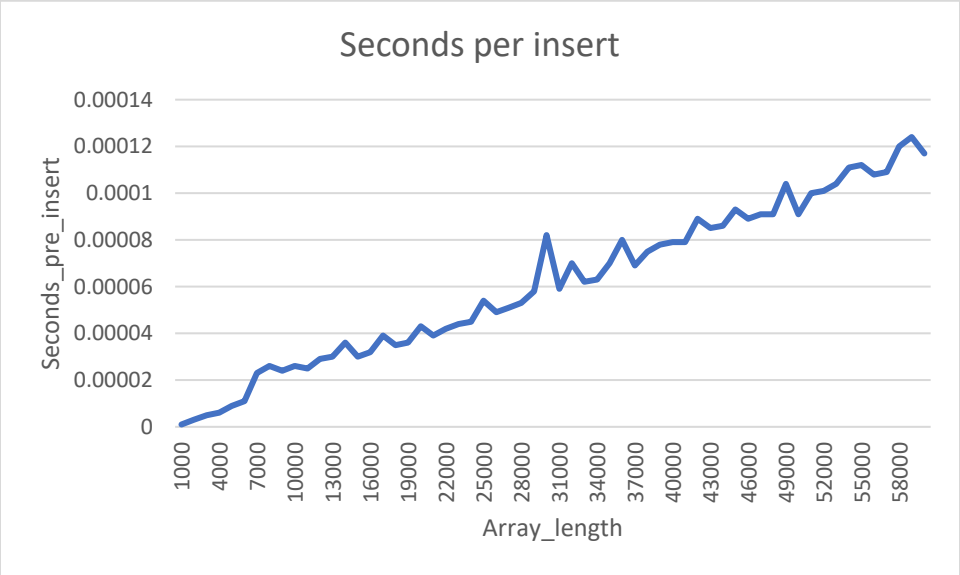
Output:

Array length	Seconds per insert
1000	0.000001
2000	0.000003
3000	0.000005
4000	0.000006
5000	0.000009
6000	0.000011
7000	0.000023
8000	0.000026
9000	0.000024
10000	0.000026
11000	0.000025
12000	0.000029
13000	0.000030
14000	0.000036
15000	0.000030
16000	0.000032

17000	**0.000039**
18000	**0.000035**
19000	**0.000036**
20000	**0.000043**
21000	**0.000039**
22000	**0.000042**
23000	**0.000044**
24000	**0.000045**
25000	**0.000054**
26000	**0.000049**
27000	**0.000051**
28000	**0.000053**
29000	**0.000058**
30000	**0.000082**
31000	**0.000059**
32000	**0.000070**
33000	**0.000062**
34000	**0.000063**
35000	**0.000070**
36000	**0.000080**
37000	**0.000069**
38000	**0.000075**
39000	**0.000078**
40000	**0.000079**
41000	**0.000079**
42000	**0.000089**
43000	**0.000085**
44000	**0.000086**
45000	**0.000093**
46000	**0.000089**

47000	**0.000091**
48000	**0.000091**
49000	**0.000104**
50000	**0.000091**
51000	**0.000100**
52000	**0.000101**
53000	**0.000104**
54000	**0.000111**
55000	**0.000112**
56000	**0.000108**
57000	**0.000109**
58000	**0.000120**
59000	**0.000124**
60000	**0.000117**

Question3



Question4:

The overall Big-O performance of insert function is O(n). This complexity is mainly contributed by two for-loops that copy the data into the new array.

Question5:

The graph of “Seconds per insert” shows that the operation time increases as the array length increases. It matches the O(n) complexity of array insert function. As the array grows, inserting an element becomes more time-consuming. It matches with our Big-O analysis.