1 Express the function $n^3/1000-100*n^2-100*n+3$ in terms of Θ-notation.

$\Theta(n^3)$

2 Consider sorting n numbers stored in array A by thirst finding the smallest element of A and exchanging it with the element in A[1]. Then find the second smallest element of A, and exchange it with A[2]. Continue in this manner for the first $n-1$ elements of A. Write pseudocode for this algorithm, which is known as selection sort. What loop invariant does this algorithm maintain? Why does it need to run for only the first $n-1$ elements, rather than for all n elements. Give the best-case and worst-case running times of selection sort in Θ-notation.

```
def sort(array):                                    #cost     times
    for i in range(0, len(array)-1, 1):             #c1       n
        smallest = array[i];                        #c2       n-1
        smallestIndex = i;                          #c3       n-1
        for j in range(i+1, len(array), 1):         #c4       n*(n+1)/2
            if array[j] < smallest:                 #c5       n*(n+1)/2 - 1
                smallest = array[j];                #c6       t
                smallestIndex = j;                  #c7       t
        array[smallestIndex] = array[i];            #c8       n-1
        array[i] = smallest;                        #c9       n-1
    return array;

print(sort([1,2,3,4,5]));
print(sort([5,4,3,2,1]));
print(sort([1]));
print(sort([]));
```

Loop invariant: At the start of each iteration elements A[1, i-1] contains result sorter subarray. Initialization: Result subbaray contains 0 elements. Main-

tenance: Let invariant hold at i iteration. Then before i+1 loop invariant holds. From subarray A[i+1, n] find min element. Swap this element with A[i+1]. subarray[1, i+1] is sorted. Termination: The condition causing loop to terminate is that i¿n-1. We don't need to check last element A[n] because subarray A[1, n-1] is sorted. So the invariant holds.

Best case runing time: $T(n) = c1*n+c2*(n-1)+c3*(n-1)+c4*(n*(n+1)/2)+c5*(n*(n+1)/2-1)+c6*0+c7*0+c8*(n-1)+c9*(n-1)$: $\Theta(n^2)$ Words case runing time: $T(n) = c1*n+c2*(n-1)+c3*(n-1)+c4*(n*(n+1)/2)+c5*(n*(n+1)/2-1)+c6*(n*(n+1)/2-1)+c7*(n*(n+1)/2-1)+c8*(n-1)+c9*(n-1)$: $\Theta(n^2)$

3 Consider linear search again(see Exercise 2.1-3). How many elements of the input sequence need to be checked on the average, assuming that the element being searched for is equally likely to be any element in the array? How about in the worst case? What are the average-case and worst-case running times of linear search in Θ notation? Justify your answer.

```
def search(array, v):
    for i in range(0, len(array), 1):
        if array[i] == v:
            return i;
    return None;

print(search([1,2,3,4,5,6], 6));
print(search([1,2,3,4,5,6], 1));
print(search([1,2,3,4,5,6], 7));
```

Everage: $n/2$. Worst: n. Everage $\Theta(n)$ Worst $\Theta(n)$

In everage n/2 is still linear dependence so $\Theta(n)$.

4 How can we modify almost any algorithm to have a good best-case running time

Use spetial cases. Use loops that breaks than the answer is finded.