DATA STRUCTURES AND ALGORITHMS
ASSIGNMENT 1

AAKASH JOG
ID : 989323563

Exercise 1.
Write a pseudocode for an algorithm that receives as input an array A containing n different numbers sorted in ascending order, and a value x such that

$$A[1] \leq x \leq A[n]$$

but x itself does not appear in the array A. The algorithm prints pair (lb, ub) such that

(1) Both lb and ub are values which appear in the array A.
(2) $lb \leq x \leq ub$
(3) lb is the closest value to x which is less than or equal to x, and ub is the closest value to x which is greater than or equal to x.

The running time of the algorithm should be $c_1 + c_2 \log(n)$ for some constants c_1 and c_2.

Solution 1.

Date: Tuesday 8^{th} March, 2016.

Algorithm 1

Input: Array A of size n, x
Output: Lower bound lb of x, upper bound ub of x

```
function FIND BOUNDS(A, x)
    lb ← FIND LOWER BOUND(A, x)
    ub ← FIND UPPER BOUND(A, x)
    return (lb, ub)
end function

function FIND LOWER BOUND(A, x)
    min ← 1
    max ← n
    found = FALSE
    while found = FALSE do
        mid ← ⌊(min+max)/2⌋
        if A[mid − 1] < x and A[mid] > x then
            lb ← A[mid − 1]
            found ← TRUE
        else if A[mid − 1] < x then
            max ← mid
        else
            min ← mid
        end if
    end while
    return lb
end function

function FIND UPPER BOUND(A, x)
    min ← 1
    max ← n
    found = FALSE
    mid ← ⌊(min+max)/2⌋
    while found = FALSE do
        if A[mid − 1] < x and A[mid] > x then
            ub ← A[mid]
            found ← TRUE
        else if A[mid − 1] > x then
            max ← mid
        else
            min ← mid
        end if
    end while
    return ub
end function
```

Exercise 2.
Let A be an array that contains n numbers and assume that the values stored in array A are not necessarily different from each other, but still sorted in ascending order. Let x be a number.
Write a pseudocode for a procedure whose running time is at most $c_1 + c_2 \log(n)$ for some constants c_1 and c_2, such that

(1) If the number x does not appear in array A, the procedure returns `NOT-FOUND`.
(2) If the number x does appear in array A, the procedure returns the index j of the first occurrence of x.

The procedure can call the Binary Search procedure. Give a short explanation.

Solution 2.

Algorithm 2

Input: Array A of size n, x
Output: Index j of first occurrence of x

```
function First Occurrence(A, x)
    index_of_any_occurrence ← Binary Search(n, A, x)
    if index_of_any_occurrence = NOT-FOUND then
        return NOT-FOUND
    end if
    min ← 1
    max ← index_of_any_occurrence
    while min < max do
        mid ← ⌊(min+max)/2⌋
        if A[mid] < x then
            min ← mid + 1
        else
            max ← mid + 1
        end if
    end while
    return min
end function
```

The function calls Binary Search to find any occurrence of the required number x, and stores this value in `index_of_any_occurrence`. It then takes into consideration the sub-array to the left of this position, including the position itself. It 'halves' the sub-array, and takes into consideration either the left or the right half depending on whether the central element is x or not. It repeats this process, repeatedly 'halving' the array, till the length of the sub-array is 1. At this point the only remaining element in the array is the first occurrence of x. It returns this index.

Exercise 3.

Let A be an array that contains n numbers. The array A is called unimodal if there exists an index $s(A)$, such that

- $1 \leq s(A) \leq n$.
- $A[i] < A[i+1]$ for each $1 \leq i \leq s(A)$.
- $A[i+1] < A[i]$ for each $s(A) \leq i \leq n$.

(1) What does the value $A[s(A)]$ hold in comparison to the other values of A? Explain shortly.

(2) Write a pseudocode for a procedure that receives as inputs an unimodal array A and its size n. The procedure returns $s(A)$.
The running time of the procedure should be at most $c_1 + c_2 \log(n)$ for some constants c_1 and c_2. Provide a short explanation of the idea behind the procedure and explain why the running time is the requested one.

Solution 3.

(1) All elements before and including $s(A)$ are in ascending order. All elements after and including $s(A)$ are in descending order. Therefore, $A[s(A)]$ is the greatest value in the array.

(2)

Algorithm 3

Input: Array A of size n
Output: Index $s(A)$ of the mode of A

```
function FIND MODE(A)
    min ← 1
    max ← n
    found = FALSE
    while found = FALSE do
        mid ← ⌊(min+max)/2⌋
        if mid = 1 then
            if A[mid] > A[mid + 1] then
                mode = mid
                found ← TRUE
            end if
        else if mid = n then
            if A[mid − 1] < A[mid] then
                mode = mid
                found ← TRUE
            end if
        else
            if A[mid − 1] < A[mid] > A[mid + 1] then
                mode = mid
                found ← TRUE
            else if A[mid − 1] ≤ A[mid] ≤ A[mid + 1] then
                min ← mid
            else
                max ← mid
            end if
        end if
    end while
    return mode
end function
```

On each execution of the while loop, the algorithm 'halves' the array. Therefore, the running time is $c_1 + c_2 \log(n)$.