CSC252 Spring 2016
Hw2 Induction
Isaiah Mann

1. **Prove by induction (to the best of your abilities) that the sum of the first n integers 1+2+… +n equals n(n+1)/2.**
    a. Base Case:
        i. n = 1
        ii. 1 (1 + 1) / 2 = 1
        iii. This assumption holds for the base case
    b. Assume:
        i. This statements holds for n
    c. Prove:
        i. Prove this statement holds for n + 1
        ii. The sum of integers 0..n+1 = n(n+1)/2 + n + 1
        iii. = n(n+1)/2 + 2n/2 + 2/2
        iv. = (n(n+1) + 2n + 2)/2
        v. = $(n^2 + n + 2n + 2)/2$
        vi. = $(n^2 + 3n + 2)/2$
        vii. = (n + 1)(n + 2)/2
        viii. = (n + 1)(n + 1 + 1)/2
        ix. This equality is true for n + 1, therefore it most hold for all integers ■
2. **Search in sorted list: takes in a sorted list of numbers and a number x, and returns the position in the array where the search element is, or should be**
    a. Input: Sorted list of numbers: **A**, number: **x**
    b. Output: Index of where the number x would/is placed in the list
    c. Method:

```
for i in 1..A.length
    if A[i] <= x
        return i
```

3. **Insert in sorted list: takes in a sorted list of numbers and a number x, and inserts x in the position where it would fit so that the result remains sorted.**
    a. Input: Sorted list of numbers: **A**, number **x**
    b. Output: None
    c. Method:

```
placeFound = false
indexOfX = -1
for i in 1..A.length
    if placeFound is true
        A[i] = A[i-1]
    if A[i] <= x and placeFound is false
        indexOfX = i
        placeFound = true
if indexOfX equals -1
    indexOfX = A.length + 1
A[indexOfX] = x
```

4. **Insertion sort: describe it in pseudo-code using the previously define Insert in sorted list function.**
    a. Input: Unsorted list of numbers **A**
    b. Output: Sorted list of numbers
    c. Method:

```
    for i in 2..A.length
        Insert A[i] into list A[1]..A[i-1]
    return A
```

5. **Find min position: takes an unsorted array and finds the position of the minimum element**
    a. Input: list of numbers **A**
    b. Output: index of minimum element
    c. Method

```
    minSeen = maxValue
    indexOfMinSeen = -1
    for i in 1..A.length
        if (A[i] < minSeen)
            minSeen = A[i]
            indexOfMinSeen = i
    return indexOfMinSeen
```

6. **Swap: takes an array and two indices representing positions in the array, and swaps the corresponding elements.**
    a. Input: list of numbers **A**, indexes to swap: **x1**, **x2**
    b. Output: None
    c. Method:

```
    tempValue = A[x1]
    A[x1] = A[x2]
    A[x2] = tempValue
```

7. **Selection Sort: describe it in pseudo-code using the previously described 'Find min position and Swap functions.**
    a. Input: Unsorted list of numbers **A**
    b. Output: Sorted list of numbers
    c. Method:

```
for i in 2..A.length
    Swap i-1 and MinPosition of list i..A.length
return A
```

8. **For each of the functions described above, state their running time (in terms of number of elementary operations), using the Big Oh, little oh, omega etc. notation, and in terms of best case, worst case and average case analysis.**
    a. Search in a Sorted List
        i. Best: O(1)
        ii. Worst: O(n)
        iii. Average: O(n)
    b. Insert in a Sorted List
        i. Best: O(n)
        ii. Worst: O(n)
        iii. Average: O(n)
    c. Insertion Sort
        i. Best: O(n)
        ii. Worst: O(n^2)
        iii. Average: O(n^2)
    d. Min Position
        i. Best: O(n)
        ii. Worst: O(n)
        iii. Average: O(n)
    e. Swap (assuming array indexing)
        i. Best: O(1)
        ii. Worst: O(1)
        iii. Average: O(1)
        iv. w/out Array Indexing:
            1. Best: O(1)
            2. Worst: O(n)
            3. Average: O(n)
    f. the following calculation assumes that the Swap function that has array indexing:
    g. Selection Sort:
        i. Best: O(n^2)
        ii. Worst: O(n^2)
        iii. Average: O(n^2)
9. **Extra Credit: Give recursive descriptions for the functions described above (except for Swap).**
    a. Search in a Sorted List
        i. FindPosition (**A** list, **x** number, **curIndex** = 1)

```
    if (x <= A[curIndex])
        return curIndex
    else
```

```
                curIndex = curIndex + 1
                return FindPosition(A, x, curIndex)
```

b. Insert in a Sorted List

```
i.  Insert (A list, x number, curIndex = 1, placeFound  = false)
        if (x <= A[curIndex] and not placeFound)
            A[curIndex] = x
            placeFound = true
        else if (placeFound)
            A[curIndex] = A[curIndex - 1]
        curIndex = curIndex + 1
        if (curIndex < A.length)
            InsertInList(A, x, curIndex, placeFound)
```

c. Insertion Sort

```
i.  InsertionSort (A list, i current index)
        Insert (A[1]..A[i-1], A[i], 1, false)
        i = i + 1
        if (i < A.length)
            InsertionSort(A, i)
```

d. Min Position

```
i.  MinPosition (A list, i current index = 1, minIndex = 1, minValue = A[1])
        if (A[i] < minValue)
            minValue = A[i]
            minIndex = i
        i = i + 1
        if (i >= A.length)
            return minIndex
        else
            return MinPosition (A, i, minIndex, minValue)
```

e. Selection Sort:

```
i.  SelectionSort (A list, i current index = 1)
        Swap (A[i - 1], MinPosition (A[i]..A.length)
        i = i + 1
        if (i <= A.length)
            SelectionSort(A, i)
```