Homework 1

Notebook:	CS 5006
Created:	2018/1/11 星期四 6:43	**Updated:**	2018/1/11 星期四 17:12
Author:	Yue Yuanjie
URL:	https://course.ccs.neu.edu/cs5006sp18-seattle/week1.html

Due by Thursday, January 18nd 10 PM, submit via your git repo.

- Exercise 1.2-2
- Exercise 1.2-3
- Exercise 2.1-1
- Exercise 2.1-2
- Exercise 2.1-3
- Exercise 2.2-1
- Exercise 2.2-2
- Problem 1-1

Exercises

1.2-2

Suppose we are comparing implementations of insertion sort and merge sort on the same machine. For inputs of size n, insertion sort runs in 8n^2 steps, while merge sort runs in 64nlgn steps. For which values of n does insertion sort beat merge sort?

Solution： 43

The question could be simplified as for which values of n that 8n^2 < 64nlgn.

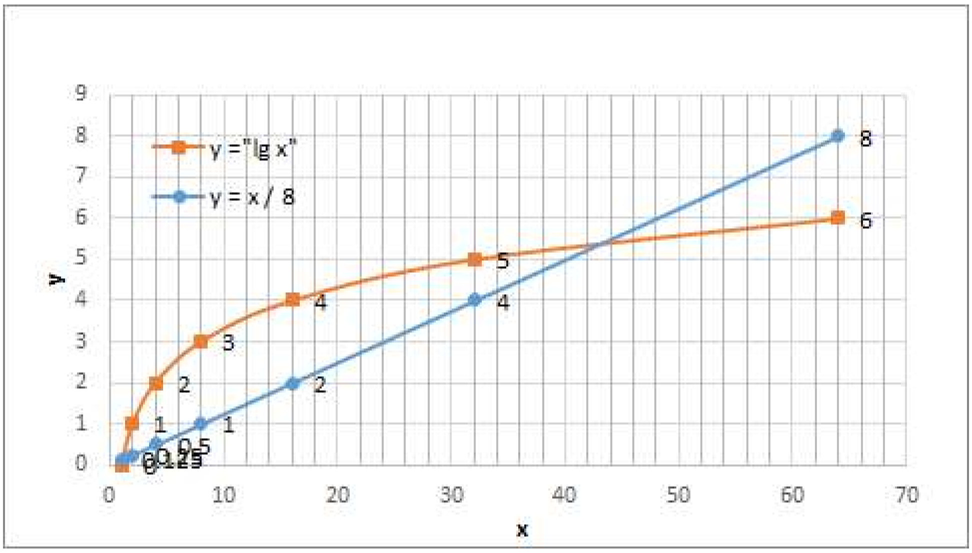
8n^2 < 64nlgn

=> lgn > n / 8

then we can calculate the increase of values of lg n and n / 8 as shown in the table

from the table we know that it is starting from n = 2 until a number between 32 and 64 that matches the equation.

x	**y = lg x**	**y = x / 8**	**lg x > x / 8**
1	0	1 / 8	×
2	1	1 / 4	√
4	2	1 / 2	√
8	3	1	√
16	4	2	√
32	5	4	√
64	6	8	×



and we need to draw a diagram that helps us finding the number is between 43 and 44.
lg 43 = 5. 425, and 43 / 8 = 5.375, which means 43 does match
lg 44 = 5.459, and 44 / 8 = 5.500, which means 44 does not match
In conclusion, when n ≤ 43 insertion sort runs faster when we're sorting at most 43 items. otherwise merge sort is faster.

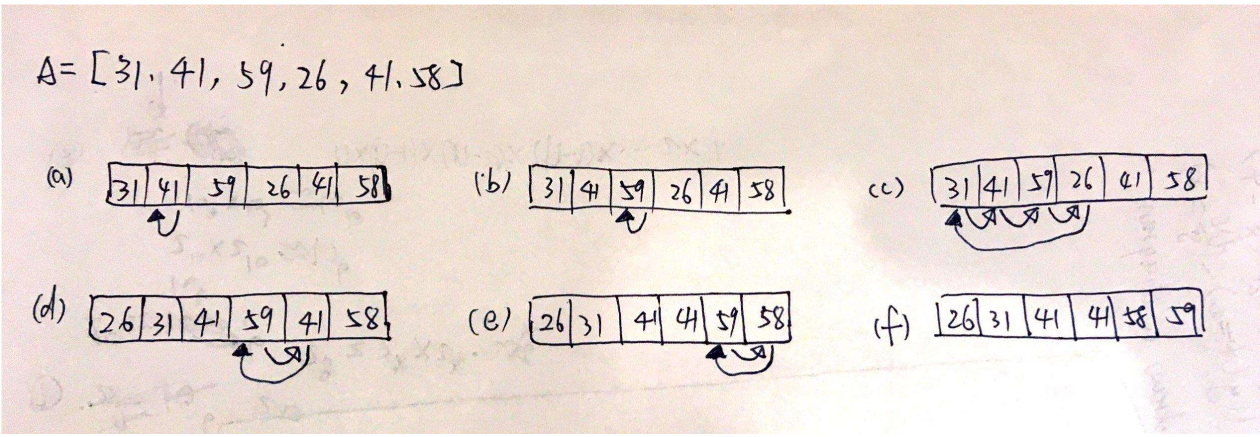
1.2-3
What is the smallest value of n such that an algorithm whose running time is 100n^2 runs faster than an algorithm whose running time is 2^n on the same machine?

Solution: n = 15
The question could be simplified as for which value of n that 100n^2 < 2^n
we could do the following transformation that:
when n = 14, 100n^2 = 19600, and 2^n = 16384
when n = 15, 100n^2 = 22500, and 2^n = 32768
so for n >= 15, 100n^2 will always be smaller than 2^n.

1-1 Comparison of running times
Solution:

	1 Second	1 minute	1 hour	1 day	1 month	1 year	1 century
micro Seconds	10^6	6×10^7	3.6×10^9	8.64×10^{10}	2.59×10^{12}	3.15×10^{13}	3.15×10^{15}
$\lg n$	$2^{1\times10^6}$	$2^{6\times10^7}$	$2^{3.6\times10^9}$	$2^{8.64\times10^{10}}$	$2^{2.59\times10^{12}}$	$2^{3.15\times10^{13}}$	$2^{3.15\times10^{15}}$
$\sqrt{n}$	1×10^{12}	3.6×10^{15}	1.29×10^{19}	7.46×10^{21}	6.72×10^{24}	9.92×10^{26}	9.92×10^{30}
n	1×10^6	6×10^7	3.6×10^9	8.64×10^{10}	2.59×10^{12}	3.15×10^{13}	3.15×10^{15}
$n\lg n$	62746	2801417	133378058	2755147513	71870856404	797633893349	6.86×10^{13}
n^2	1000	7745	60000	293938	1609968	5615692	56176151
n^3	100	391	1532	4420	13736	31593	146679
2^n	19	25	31	36	41	44	51
$n!$	9	11	12	13	15	16	17

2.1-1
Using Figure 2.2 as a model, illustrate the operation of INSERTION -SORT on the array A=< 31 , 41, 59, 26, 41, 58>
Solution:



2.1-2
Rewrite the INSERTION -SORT procedure to sort into nonincreasing instead of non-decreasing order.
Solution:

```
INSERTION-SORT(A)
for j=2 to A.length
    key = A[j];
    i = j-1;
    while i>0 and A[i] < key
        A[i+1] = A[i]
        i = i-1
    end while
    A[i+1] = key
end for
```

2.1-3
Solution:

```
LINEAR SEARCH(A and v)
for i=1 to A.length
    if A[i] == v then
        return i
    end if
end for
return NIL
```

Loop Invariant: for all the indices k <= j - 1, there is no such index k that A[k] = v.
Initialization:
Obviously this is true before the first iteration.

Maintenance:
In order to proceed to the next iteration of the loop, we need that for the current value of j,
we do not have A[j] = v.
Termination:
If we found the i that A[i] = v, then the loop will terminate.
If we have go over all the loop without return in the middle, then we know that there is no index that has value v so we return NIL

2.2-1
Express the function n^3 /1000 - 100 n ^2 - 100 n +3 in terms of Θ notation.
Solution : *Θ (n ^ 3)*

2.2-2
Consider sorting n numbers stored in array A by first finding the smallest element of A and exchanging it with the element in AŒ1?. Then find the second smallest element of A, and exchange it with AŒ2?. 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 , -notation.
Solution:

```
SELECTION SORT(A)
for i=1 to A.length-1
    min = i
    for j = i+1 to A.length
        if A[j] < A[min] then
            min = j
        end if
    end for
    swap A[i] and A[min])
end for
```

As a loop invariant we choose that A[1,...,i−1] are sorted and all other elements are greater than these.
so when we have sorted the first n - 1 element, the nth element will surly be the largest

For time complexity: the inner loop which try to find the minimum element will go over until the end of the array, which make the running time
=> (n - 1) + (n - 2) + (n - 3) + ... + 2 + 1
=> (n - 1)n / 2
=> n^2
and more importantly, this holds for both the best-case and the worst-case.
So it a *Θ (n ^ 2)*