Homework #1

1. (10 points) What is the smallest integer value of n>1 such that an algorithm whose running time is $53n(\log_2 n)^3$ runs slower than an algorithm whose running time is $185n(\log_2 n)^2$ on the same machine? Justify your answer.

Ans:

$53n(\log_2 n)^3 > 185n(\log_2 n)^2$

Then, we have:

$$53n(\log_2 n)^3 > 185n(\log_2 n)^2$$
$$\log_2 n > 3.49$$
$$n > 2^{3.49} \approx 11.24$$

So we know that the smallest value of n satisfying $53n(\log_2 n)^3 > 185n(\log_2 n)^2$ is 12.

2. (10 points) Exercise 1.2-2 (page 14).

Ans: We consider the following equation: $f(n) = 8n^2 - 64n\lg n$. The problem becomes to finding the values of n such that $f(n) < 0$.

The following table shows the values of $f(n)$ for n in the range [1,6]

n	1	2	3	4	5	6	7
f(n)	8	-6.5	-19.6	-26.1	-23.7	-10.8	13.4

Moreover, we have the derivative of $f(n)$:

$$f'(n) = 16n - 64\lg n - \frac{64}{ln10} = 16(n - lgn - 1.74)$$

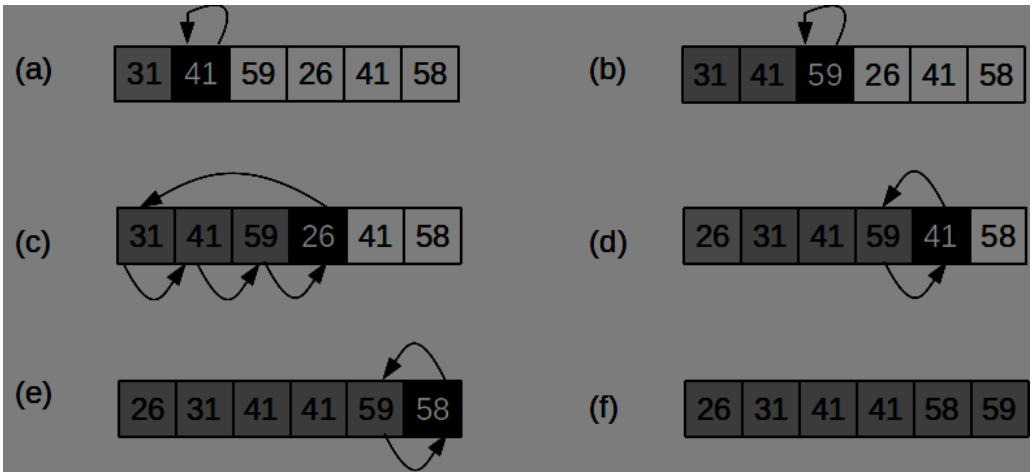
from which, we know that $f'(n) > 0$ for $n>5$. That means $f(n)$ is monotonically increasing on the increasing of n, which gives us the following:

$$f(n) > f(7) = 13.4 > 0, for\ n > 7$$

Therefore, the integer values of n such that $f(n) < 0$ can only be: 2, 3, 4, 5, 6.

3. **(10 points) Exercise 2.1-1 (page 22). Illustration of INSERTION-SORT.**

Ans:



4. (20 points) Exercise 2.1-3 (page 22).

LinearSearch (A, v)

```
for i = 1 to A.length
        if A[i] == v return i
        // i = i +1
        // if i <= n
        //return i
return NIL
```

At the start of the each iteration, loop invariant: v not ∈ {A[1], . . . , A[i-1]}.

• Initialization: At the beginning of the first iteration, we have

i = 1, so the statement is empty.

• Maintenance: The code have n't returned the value i yet, so there v would not in {A[1], …, A[i-1]}, so the invariant is maintained by the loop.

• Termination: Since the loop is a for loop over a finite sequence 0 ...len(A)-1, the loop will always terminate. If the algorithm finds v in the array A, we have A[i] = v, and the algorithm returns the index I is correct. Otherwise, the loop terminates after len(A) iterations, in which case the invariant states that v not ∈ {A[1], . . . , A[len(A)]}, which is the whole array A, so we can guarantee that NIL, the value the algorithm returns, is correct.

Therefore, the function linear search is correct by the loop invariant.

5.(20 points) Exercise 2.2-3 (page 29)

Ans:

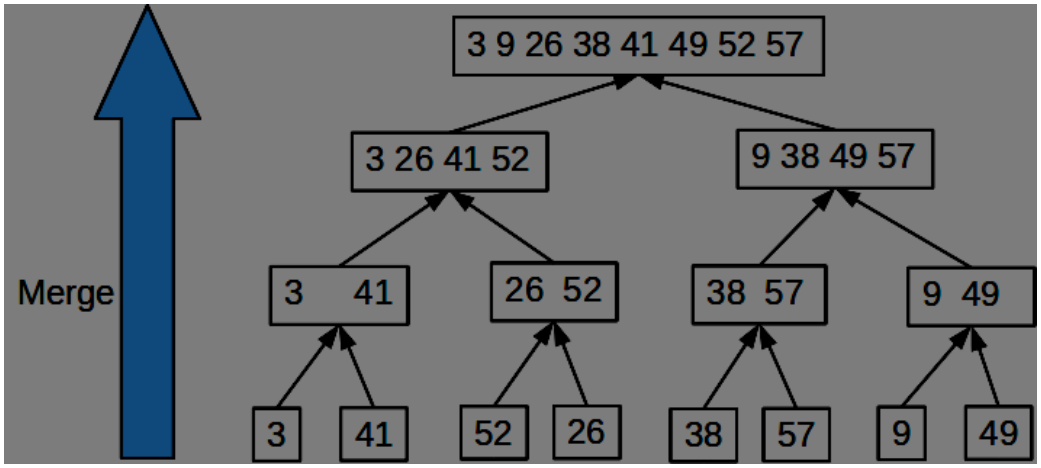
We use the comparison of two numbers as the primitive operation for calculating the running time. We assume that in the unsorted array$A = \{a_{1,}a_{2,} \dots, a_n\}, \quad a_i \neq a_j \; if \; i \neq j$ for any *i,j in [1,n]*. Since the target value v is equally likely to be any element in A, then the probability of $v=a_i$ is: $Pr(v = a_i) = \frac{1}{n}, i = 1,2, \dots, n.$

We also know that the linear searching takes exactly i comparisons to find $v=a_i$ by scanning the array from a_1 to a_i. Therefore, the average running time is:

$$T(n) = \sum_{i=1}^{n} \; cost(a_i) \cdot Pr(v = a_i) = \sum_{i=1}^{n} \; i \cdot \frac{1}{n} = \frac{\frac{1}{n} \cdot n(n+1)}{2} = \frac{n+1}{2} \sim \Theta(n)$$

The worst case of linear searching is $v=a_n$ such that the linear searching will scan the whole array. Therefore, the running time of the worst case is: $T(n) = n = \Theta(n)$

6. (10 points) Exercise 2.3-1 (page 37). Illustration of MERGE-SORT.



7. (20 points) Exercise 2.3-5 (page 39). Binary search.

1) The pseudocode of Binary Search:

```
BINARY-SEARCH ( A, v ): // A[1..n] is the sorted array, v is the target value
1.      low = 1
2.      high = A.length
3.      while low <= high
4.          mid = (low + high) / 2
5.          if A[mid] == v
6.              return mid    // Succ
7.          if A[mid] < v
8.              low = mid + 1
9.          else
10.             high = mid – 1
11.     return 0    // FAIL: v is not found
```

We also have the recursive version of BINARY-SEARCH:

```
BINARY-SEARCH-HELPER (start, end, A, v):
1.      if start > end
2.          return 0
3.      mid = (start + end) / 2    // mid is an integer
4.      if A[mid] == v
5.          return mid
6.      if A[mid] < v
7.          return BINARY-SEARCH (mid+1, end, A, v)
8.      else
9.          return BINARY-SEARCH (start, mid+1, A, v)
```

2) The running time of the binary search.

In the pseudo code of BINARY-SEARCH, each iteration in the while loop (line 3 ~ 10), only the midpoint of the sorted sub-array will be checked, and only half of them will be used for the further search if the midpoint is not equal to the target value v. In the worst case, the while loop will terminate when the sub-array can not be divided any more, which means the length of the sub-array is 1. Since in each step, the loop takes two comparisons, then we have the recurrence as follows:

$$T(n) = \begin{cases} c & n = 1 \\ T(\frac{n}{2}) + c & otherwise \end{cases}$$

According to the master theorem in section 4.5, we have the solution: $T(n) = \Theta(logn)$