Algorithm Analysis
Handout 3
Deadline October 17

Exercise 3.1
Using Figure 8.2 in your textbook as a model, illustrate the operation of COUNTING-SORT on the array A = (6, 0, 2, 0, 1, 3, 4, 6, 1, 3, 2).

Exercise 3.2 (option, no duty)
The QUICKSORT algorithm of the textbook contains two recursive calls to itself. After the call to PARTITION, the left subarray is recursively sorted and then the right subarray is recursively sorted. The second recursive call in QUICKSORT is not really necessary; it can be avoided by using an iterative control structure. This technique, called ***tail recursion***, is provided automatically by good compilers.
Consider the following version of quicksort, which simulates tail recursion.

```
TAIL-RECURSIVE-QUICKSORT(A, p, r)
while p < r
do // Partition and sort left subarray.
   q ← PARTITION(A, p, r)
   TAIL-RECURSIVE-QUICKSORT (A, p, q - 1)
   p ← q + 1
```

a) Argue that `TAIL-RECURSIVE-QUICKSORT (A, 1, length[A])` correctly sorts the array A.

Compilers usually execute recursive procedures by using a ***stack*** that contains pertinent information, including the parameter values, for each recursive call. The information for the most recent call is at the top of the stack, and the information for the initial call is at the bottom. When a procedure is invoked, its information is ***pushed*** onto the stack; when it terminates, its information is ***popped***. Since we assume that array parameters are represented by pointers, the information for each procedure call on the stack requires $O(1)$ stack space. The ***stack depth*** is the maximum amount of stack space used at any time during a computation.

b) Describe a scenario in which the stack depth of `TAIL-RECURSIVE-QUICKSORT` is $\Theta(n)$ on an n-element input array.

c) Modify the code for `TAIL-RECURSIVE-QUICKSORT` so that the worst-case stack depth is $\Theta(\lg n)$. Maintain the $O(n \lg n)$ expected running time of the algorithm.

Exercise 3.3 (option, no duty)
Show that there is no comparison sort whose running time is linear for at least half of the n! many inputs of length n.
Tip: This is quite easy to prove by using the decision tree.

Exercise 3.1

A : [6, 0, 2, 0, 1, 3, 4, 6, 1, 3, 2]
B : [ ]
C : [2, 2, 2, 2, 1, 0, 2] => [2, 4, 6, 8, 9, 9, 11]

```
B :    [ , , , , , 2, , , , , ]
       [ , , , , , 2, , 3, , , ]
       [ , , , 1, , 2, , 3, , , ]
       [ , , , 1, , 2, , 3, , , 6]
       [ , , , 1, , 2, , 3, 4, , 6]
       [ , , , 1, , 2, 3, 3, 4, , 6]
       [ , , 1, 1, , 2, 3, 3, 4, , 6]
       [ , 0, 1, 1, , 2, 3, 3, 4, , 6]
       [ , 0, 1, 1, 2, 2, 3, 3, 4, , 6]
       [ 0, 0, 1, 1, 2, 2, 3, 3, 4, , 6]
       [ 0, 0, 1, 1, 2, 2, 3, 3, 4, 6, 6]
```

⇨ While one element is filled in array B, element in array C count down 1.
The result of array C is [0, 2, 4, 6, 8, 9]

Exercise 3.2

a)
First loop, the array is divided to left and right array like common loop not using tail recursive. Left array will be sorted in line 4. Then the left array will be sorted recursively. In the second loop, it’s time to sort the right array. Left of the right array will be sorted in line 4, and the right of the right array will be sorted in line 3 of the next loop. This technique uses iterative thing, by increasing the pointer p.

b)
The sorted case.

c)
```
MODIFIED-TAIL-RECURSIVE-QUICKSORT(A, p, r)
    While p < r
        q = PARTITION(A, p, r)
        if q < floor((p + r) / 2)
            MODIFIED-TAIL-RECURSIVE-QUICKSORT(A, p, q - 1)
            P = q + 1
        else
            MODIFIED-TAIL-RECURSIVE-QUICKSORT(A, q + 1, r)
        R = q - 1
```
Exercise 3.3

The comparison sorts can be represented by a tree that has a n factorial leaves.(=n!)
Using stirling approximation, n factorial can be written to sqrt(2*pi*n) * (n/e)^n.

So, the height of the tree is
log(n!) = log(2*pi*n)+nlog(n/e) (Let’s call this height(n).)

Following the asymptotic notation,

theta(heigth(n)) = ~~log(2*pi)~~+log(n)+nlog(n)+~~nlog(e)~~ = (n~~+1~~)log(n) ~ nlog(n)