Homework Assignment #3

1 (5 pts) Demonstrate how mergesort works when sorting the following list of numbers:

| 6 | 1 | 4 | 2 | 3 | 8 | 7 | 5 |

---------------------------------
Given the following array (list), follows the following pseudocode to partition the array using the first element, 6. For your reference, here is an article on quicksort with python code: http://interactivepython.org/courselib/static/pythonds/SortSearch/TheQuickSort.html

// partition list A[startI...endI] using
// A[startI] (first element in the sublist)
Partition (A, startI, endI)

1. pivot = A[startI] //set first element as pivot value
2. left = startI+1; //left index
3. right = endI; //right index

4. do {
5.   // increment left until we find a value larger than pivot or
6.   // until left==right
7.   while (a[left] <= pivot && left != right) //TYPO in original version
8.     left++
9.   // decrement right until we find a value smaller than pivot or until
10.  // left==right
11.   while (a[right] >= pivot && left != right)
12.     right--
13.   if (left < right)
14.     swap (a[left], a[right])
15. } while (left <= right)
16. if (right != startI)
17.   swap (a[right], a[startI])
18.
19. return right //the index of pivot value in the array

-------------------------------
| 6 | 1 | 4 | 2 | 3 | 8 | 7 | 5 |
-------------------------------

(1) Show the array after each time swap (line 14 and line 18 above) is executed.
(2). Consider that the array might contain duplicates (including multiple elements equal to the pivot value), add a few lines of codes after line 19 in the above pseudocode so that all pivot values are moved to the middle, and the function returns the starting and ending index of the middle section that contains elements that are equal to the pivot.

(3). Read the DPV book section on Selection (page 60), and comment on where the 2nd smallest element should be, i.e., the left partition, middle partition or right partition of the above partition result?
Efficient exponentiation algorithm below.

(a) Finish (or correct) the in-class exercise below. (Hint: you can type in your code and test it).

```c
// return a^n, i.e., raise a to the power of n
// a is an integer, n is a natural number
Exp (int a, int n)

//1. write the base case here (for what n value, we can just solve it without
// making recursive call

result = Exp (a, n/2) //n/2 is integer division

if n is odd
    // fill in what to do here

else
    // fill in what to do here
```

(b) Recall that to verify a recursive algorithm is correct, three questions need to be asked. Answer the three questions for the above pseudocode to show your above algorithm is correct.

(c) Analyze the above algorithm’s running time.
(a) Radix sort works on integers as follows: first sort the data using the ones digit (as the key), then sort the data using the tens digit, ... and lastly sort the data using the most significant digit. Follow the example given in class (and in slides) to demonstrate how the following list of integers are sorted using radix sort:

149, 99, 321, 264, 501, 249, 354, 123

(b) Why we cannot reverse the order of radix sorting, i.e., sort using the highest digit, and then the second highest digit, and lastly sort using the ones digit?

(c) If we want to consider the most significant digit first, we could use bucket sort, by first put all elements with 1 in 100s digit in a bucket, all elements with 2 in 100s digit in second bucket, ... How would you complete the sorting algorithm?
Suppose a list $P[1...n]$ gives the daily stock price of a certain company for a duration of $n$ days, we want to find an optimal day $d_1$ to buy the stock and then a later day $d_2$ to sell the stock, so that the profit is maximized (i.e., $P[d_2] - P[d_1]$ is maximized. For example, if

$$P[1...6] = 20, 10, 30, 50, 5, 14$$

Then we should buy in at day 2, and sell at day 4 to earn profit of $50 - 10$ per share. (Simply buying at the lowest point and selling at the highest point will not necessarily work, as one have to buy before sell.)

(a) Think about how you solve this problem algorithmically, and write down the pseudocode and analyze the running time in terms of input size $n$ (the length of the stock price data). Hint: A good starting point to any problem is try brute-force approach first, i.e., try all possible pairs of buy/sell days, and see which one yields largest profit.

(b) Think about how to solve this problem more efficiently using divide-and-conquer paradigm, and describe your algorithm in pseudocode. Hint: Consider the following analysis of the problem, and “translate” this into a pseudocode. For an input of length $n$ (i.e., prices of the stock for $n$ consecutive days), the optimal way to make profit is the best among the following three possibilities:

i. either the optimal buy/sell days are both in the first half of the $n$ days (i.e., left half of the array).

ii. or the optimal buy/sell days are both in the second half of the $n$ days (i.e., left half of the array).

iii. or the optimal days is buy in the first half of the $n$ days, and sell in the second half of the $n$ days.

Note that for possibility 1 and 2 above, they are just solutions to the two subproblems (obtained by cutting the array into two halves), while the solution to the third possibility can be found by searching for the lowest price among first half of the array, and highest price among second half of the array.

(c) (Optional) Implement both algorithms above, and test them using the the following data given in CVS file (downloaded from Yahoo finance), using the Open (second column) price for each data to calculate the optimal buy/sell dates.

http://storm.cis.fordham.edu/zhang/cs4080/data/GOOG.csv