Due Date: April 29, 2019, BEFORE the class begins

This assignment is an **OPTIONAL** homework and provides **extra credits** to help students to improve the homework grade. You may choose to finish none, some or all of them. The assignment first helps you review Chapter 10 and 12 and prepare for later chapters. Students should have learned these materials in previous classes (e.g., Computing I, Computing II). You need to review these two chapters even you choose not to submit it. The assignment also gives you an opportunity to apply what you have learned to solve challenging problems.

1. **Linked List** (20 points)

(1) (10 points) Rewrite algorithm COUNTING-SORT on p195 in the textbook using a Linked list data structure. Let's assume that the input (needs to sort) does not only contain a key and it also contains some type of satellite data (defined in textbook p147). For example, an employee record contains social security number as a key and also contains the employee's name, contact information, etc. Make sure that the algorithm is still *stable*.

(2) (10 points) What is the running time of your algorithm? Show a detailed analysis.

2. **Binary Search Tree and Heap** (10 points)
Exercise 12.1-2, textbook p289

3. **Binary Search Tree** (10 points)
Exercise 12.2-1, textbook p293

4. **Algorithm Design and Analysis** (30 points)

You are given an array A, which stores n distinct numbers. The sequence of numbers in the array is unimodal. In other words, there is an index i such that the sequence A[1 . . . i] is increasing ($A[j] < A[j + 1]$ for $1 \leq j < i$), and the sequence A[i . . . n] is decreasing ($A[j] > A[j+1]$ for $i \leq j \leq n$). The index i is called the mode of A. For example, <1, 3, 5, 7, 12, 13, 14, 10, 9, 6, 2> is a unimodal array and the mode of A is 7 (the index of element 14).

Design an *efficient* **divide-and-conquer** algorithm (should be better than the brute-force algorithm) that only accepts a unimodal array A and n as inputs and returns the mode of A, i.e., the **index *i***.

(1) (20 points) Write **Pseudocode for the algorithm.** *(please use textbook conventions)*
(2) (10 points) **Analysis:** Derive a recurrence for the running time of your algorithm. Justify your answer by listing the cost for executing each line of code and the number of executions for each line and show how you solve the recurrence.

5. **Algorithm Design and Analysis** (30 points)
Design an efficient Divide and Conquer Algorithm to find the kth smallest element in Array A (assuming elements are distinct). <u>Hint</u>: modify quicksort and the algorithm should be better than $\Theta(n\lg n)$.

<u>Input</u>: an array *A* contains *n* elements, and an integer *k* (a valid index, $1 \le k \le n$)
<u>Output</u>: the kth smallest element (array A can be changed)

(1) (20 points) **Pseudocode**: *(please use textbook conventions)*
(2) (10 points) **Analysis:** Derive a recurrence for the running time of your algorithm. Justify your answer by listing the cost for executing each line of code and the number of executions for each line. Solve the recurrence.

Algorithms -- COMP.4040 Honor Statement
(Courtesy of Prof. Tom Costello and Karen Daniels with modifications)

Must be attached to each submission

Academic achievement is ordinarily evaluated on the basis of work that a student produces independently. Infringement of this Code of Honor entails penalties ranging from reprimand to suspension, dismissal or expulsion from the University.

Your name on any exercise is regarded as assurance and certification that what you are submitting for that exercise is the result of your own thoughts and study. Where collaboration is authorized, you should state very clearly which parts of any assignment were performed with collaboration and name your collaborators.

In writing examinations and quizzes, you are expected and required to respond entirely on the basis of your own memory and capacity, without any assistance whatsoever except such as what is specifically authorized by the instructor.

I certify that the work submitted with this assignment is mine and was generated in a manner consistent with this document, the course academic policy on the course website on Blackboard, and the UMass Lowell academic code.

Date: 04/29/2019

Name (please print): DANGNHI NGO

Signature: [illegible]

20 1/ Linked List

90/100

(1) Rewrite algorithm COUNTING_SORT using a Linked list data structure

```
Counting_sort_linked_list (A, B, R)        (R is a range of elements in A)
 1. Let l be a linked list
 2. for i = 0 to R
 3.     list - insert (l, 0)
 4. for j = 0 to A.length
 5.     temp1 = get_node (A, j)
 6.     increment_node (l, temp1.key)
 7. temp2 = l.head.next
 8. while temp2 ≠ null
 9.     temp2.key = temp2.key + temp2.prev.key
10. for j = A.length downto 1
11.     temp1 = get_node (A, j)
12.     temp2 = get_node (l, temp1.key)
13.     get_node (B, temp2.key) = A.iter.key
14.     temp2.key = temp2.key - 1
```

```
get_node (L, index)
1.   h = L.head
2.   for i = 0 to index
3.       h = h.next
4.   return h
```

```
increment_node (L, index)
1.   h = L.head
2.   for i = 0 to index
3.       h = h.length
4.       h.key = h.key + 1
```

get_node (L, index): traverse linked list to the desired node

increment_node (L, index): traverse linked list to the desired node and add 1 to its value

(2) Running time for the algorithm:

- $O(l)$: create linked list l
- $O(n^2)$: iterate through A, and traverse through l to desired index for every iteration on A
- $O(l)$: change the counting linked list to l into the index position linked list l
  counting linked list: number of times elements in A appear
  index position: where to place elements in A into linked list B
- $O(n^2 + ln)$: traverse through A backwards $[O(n)]$, traverse through l every iteration $[O(n^2)]$
  and traverse linked list B

$$T(n) = O(l) + O(n^2) + O(l) + O(n^2 + ln)$$
$$= O(n^2 + ln)$$

10 2/ Binary Search Tree and Heap

Exercise 12.1 - 2

The difference between the binary-search-tree property and the min-heap property is that, in the binary-search-tree, the left child is smaller than its parent, and the right child is larger than its parent; but in the min-heap, both left child and right child are larger than their parents.

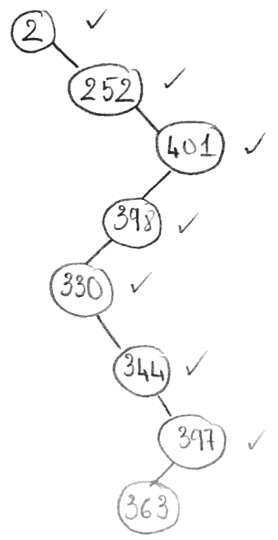
The min-heap property cannot be used to print out the keys of an n-node tree in sorted order in O(n) time because in min heap, the larger element can be in the left child or right child. The running time for sorting the min heap is O(nlgn) that is more than O(n) running time.

10 3/ Binary Search Tree

Exercise 12.2 - 1

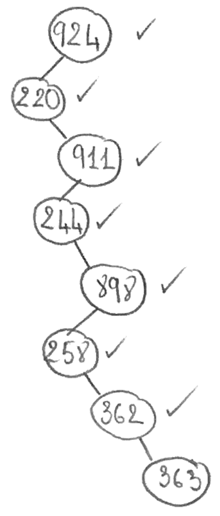
Suppose that we have numbers between 1 and 1000 in a binary search tree, and we want to search for the number 363.

a/ 2, 252, 401, 398, 330, 344, 397, 363



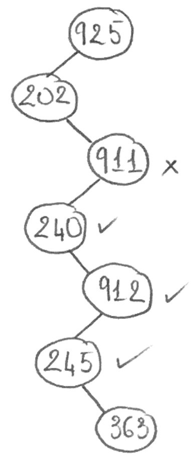
This could be a sequence of nodes.
All nodes that are smaller than their parents are to the left, and all nodes that are larger than their parents are to the right.

b/ 924, 220, 911, 244, 898, 258, 362, 363



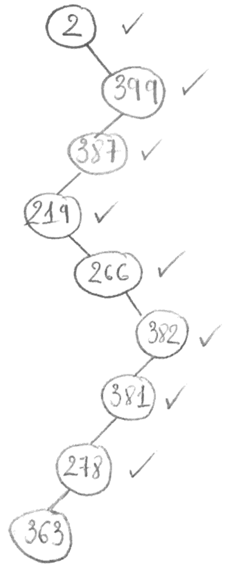
This could be a sequence of nodes.
The binary search pattern is maintained for all nodes shown in this path.

c/ 925, 202, 911, 240, 912, 245, 363



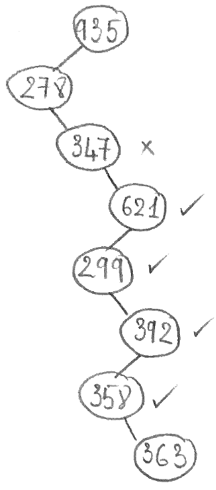
This could not be a sequence of nodes
Because the node $\textcircled{912}$ is to the left side of the node $\textcircled{911}$, which violates the binary search tree property

d/ 2, 399, 387, 219, 266, 382, 381, 278, 363



This could be a sequence of nodes
All nodes in the path follow the rules of the binary-search-tree property

e/ 935, 278, 347, 621, 299, 392, 358, 363



This could not be a sequence of nodes
Because the node $\boxed{299}$ is to the right of the node $\boxed{347}$, which violates the binary - search - tree property

30 4/ Algorithm Design and Analysis

(1) Pseudocode for the algorithm

```
Find_mode (A, n)                                                              Cost     # of executions
1. if n == 1                                                                  C1       1
2.        return 1                                                            C2       1
3. if A[⌊n/2⌋] > A[⌊n/2⌋ + 1]                                                 C3       1
4.        index = Find_mode (A[1.. ⌊n/2⌋], ⌊n/2⌋)                             T(n/2)   1
5. else                                                                       C4       1
6.        index = Find_mode (A[⌊n/2⌋ + 1 ... n, n - ⌊n/2⌋) + ⌊n/2⌋            T(n/2)   1
7. return index                                                               C5       1
```

(2) The recurrence for the running time of your algorithm

$T(n) = C1 + C2 + C3 + T(\frac{n}{2}) + C4 + C5$ (Only line 4 or line 6 happens $\Rightarrow$ cost for 2 lines is $T(\frac{n}{2})$)

$= T(\frac{n}{2}) + \theta(1)$

$= \theta(\lg n)$ (Case 2 Master Method)

20 5/ Algorithm Design and Analysis

Find the k^{th} smallest element in Array A

Input: an array A contains n elements, and an integer k (a valid index, $1 \leqslant k \leqslant n$)

Output: the k^{th} smallest element

(1) Pseudocode

```
    FIND_SMALLEST(A, p, r, k)
1. if p == r                                        ] Θ(1)
2.     return A[p]
3. q = RANDOMIZED_PARTITION(A, p, r)                ]
4. i = p - q + 1                                    ]
5. if k == i                                        ]
6.     return A[q]                                  ] Θ(n²)
7. elseif k < i                                     ]
8.     return FIND_SMALLEST(A, p, q - 1, k)         ]
9. else return FIND_SMALLEST(A, p + 1, r, k - i)    ]
```

(2) Best case : $T(n) = \Theta(1)$

Worst case : $T(n) = T(n-1) + \Theta(n) = \underline{\Theta(n^2)}$