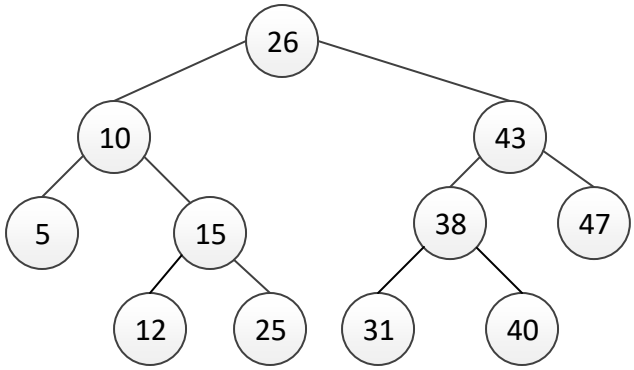
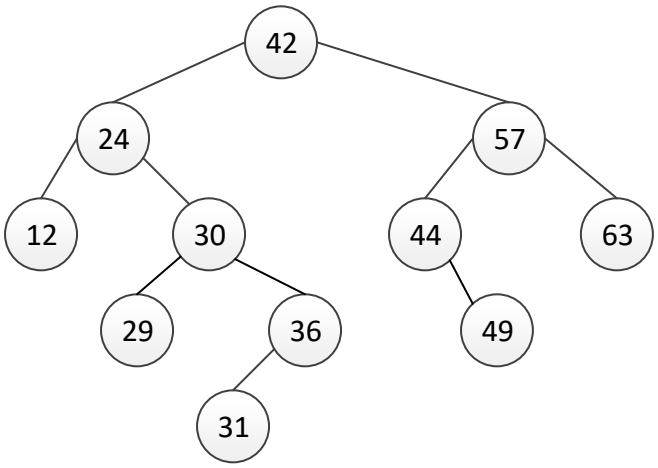
CS526 O2
Homework Assignment 5

**Problem 1 (10 points)**. Consider the following binary search tree:



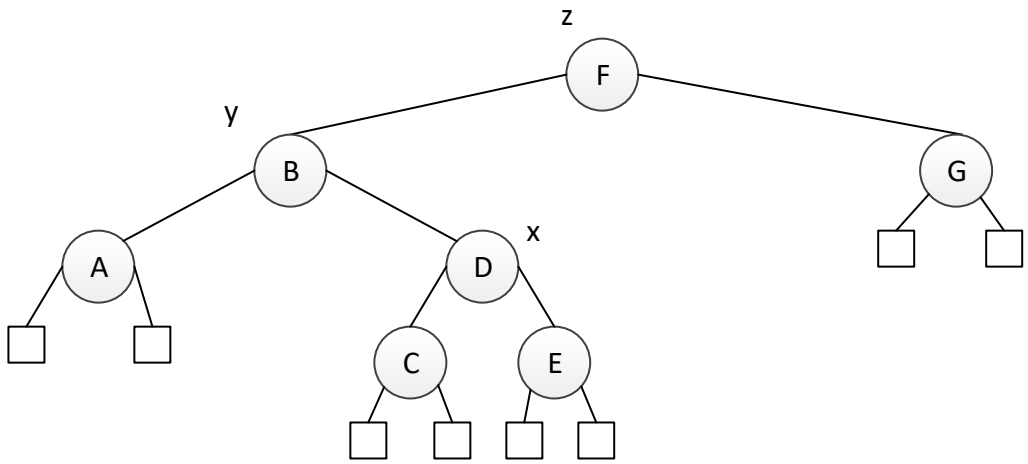

Show the resulting tree if you add the entry with key = 13 to the above tree. You need to describe, step by step, how the resulting tree is generated.

**Problem 2 (10 points)**. Consider the following binary search tree:



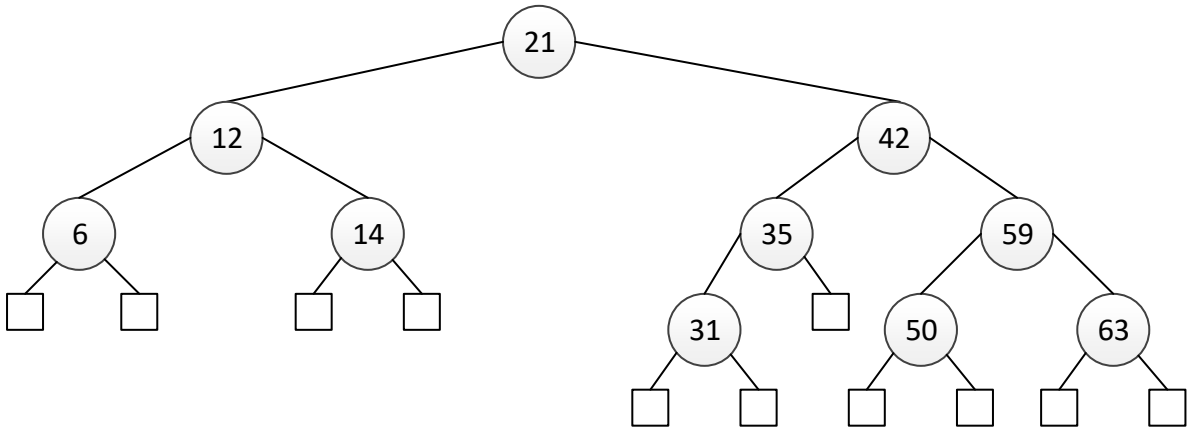

Show the resulting tree if you delete the entry with key = 42 from the above tree. You need to describe, step by step, how the resulting tree is generated.

**Problem 3 (10 points).** Consider the following AVL tree, which is unbalanced:



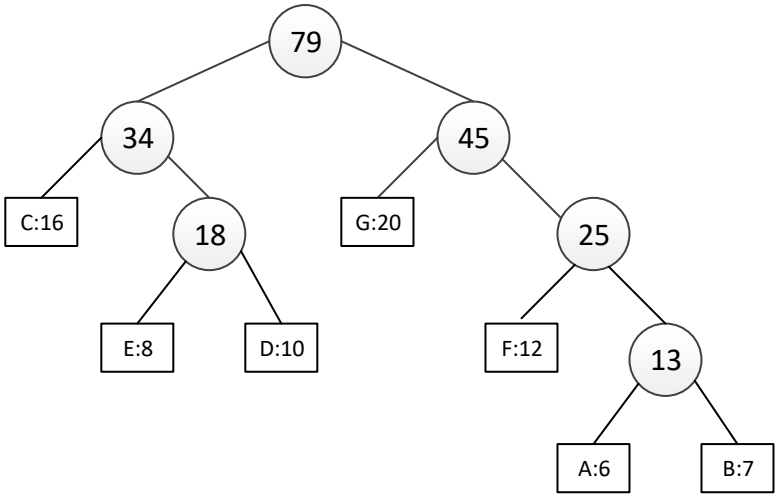

Note that the nodes *F*, *B* and *D* are labeled *z*, *y*, and *x*, respectively, following the notational convention used in the textbook. Apply a trinode restructuring on the tree and show the resulting, balanced tree.

**Problem 4 (10 points)**. Consider the following AVL tree.




Show the resulting, balanced tree after inserting an entry with key = 54 to the above tree. You must describe, step by step, how the resulting tree is obtained.

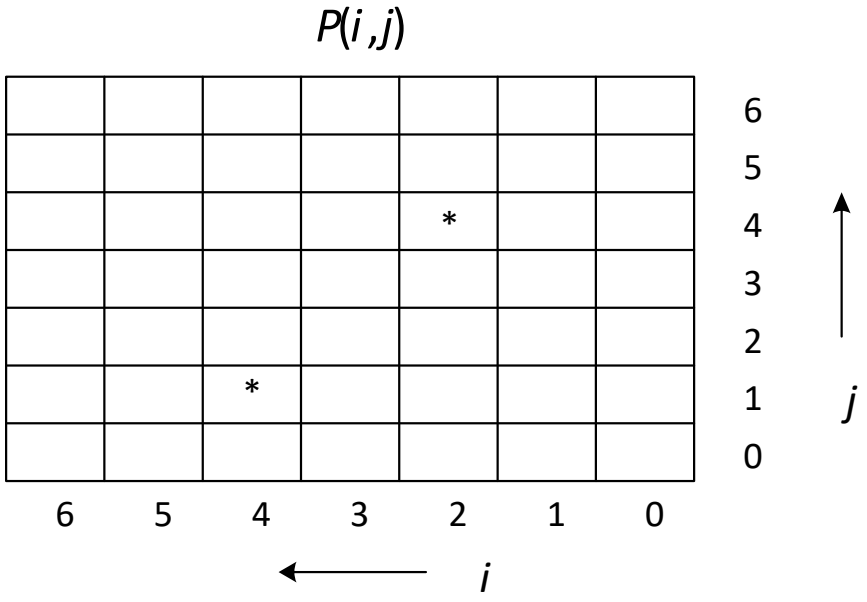
**Problem 5 (10 points).** Consider the following Huffman tree:




(1) Encode the string "BDEGC" to a bit pattern using the Huffman tree.

(2) Decode the bit pattern "010111010011" to the original string using the Huffman tree.

**Problem 6 (10 points).** This question is about the *World Series* problem that we discussed in the class. The following is the probability matrix for the problem.




Calculate the probabilities of $P(4, 1)$ and $P(2, 4)$, which are marked with *. You must show all intermediate steps and calculations.

**Problem 7 (40 points).** The goal of this problem is to give students an opportunity to compare and observe how running times of sorting algorithms grow as the input size (which is the number of elements to be sorted) grows. Since it is not possible to measure an accurate running time of an algorithm, you will use an *elapsed time* as an approximation. How to calculate the elapsed time of an algorithm is described below.

You will use four sorting algorithms for this experiment: insertion-sort, merge-sort, quick-sort and heap-sort. A code of insertion-sort is in page 111 of our textbook. An array-based implementation of merge-sort is shown in pages 537 and 538 of our textbook. An array-based implementation of quick-sort is in page 553 of our textbook. You can use these codes, with some modification if needed, for this assignment. For heap-sort, our textbook does not have a code. You can implement it yourself or you may use any implementation you can find on the internet or any code written by someone else. If you use any material (pseudocode or implementation) that is not written by yourself, you must clearly show the source of the material in your report.

A high-level pseudocode is given below:

```
for n = 10,000, 20,000, . . ., 100,000 (for ten different sizes)
      Create an array of n random integers between 1 and 1,000,000
      Run insertionsort and calculate the elapsed time
      // make sure you use the initial, unsorted array
      Run mergesort and calculate the elapsed time
      // make sure you use the initial, unsorted array
      Run quicksort and calculate the elapsed time
      // make sure you use the initial, unsorted array
      Run heapsort and calculate the elapsed time
```

You can generate *n* random integers between 1 and 1,000,000 in the following way:

```
Random r = new Random( );
for i = 0 to n – 1
      a[i] = r.nextInt(1000000) + 1
```

You can also use the *Math.random*( ) method. Refer to a Java tutorial or reference manual on how to use this method.

Note that it is important that you use the initial, unsorted array for each sorting algorithm. So, you may want to keep the original array and use a copy when you run each sorting algorithm.

You can calculate the elapsed time of the execution of a sorting algorithm in the following way:

```
long startTime = System.currentTimeMillis();
call a sorting algorithm
long endTime = System.currentTimeMillis();
long elapsedTime = endTime - startTime;
```

Write a program that implements the above requirements.

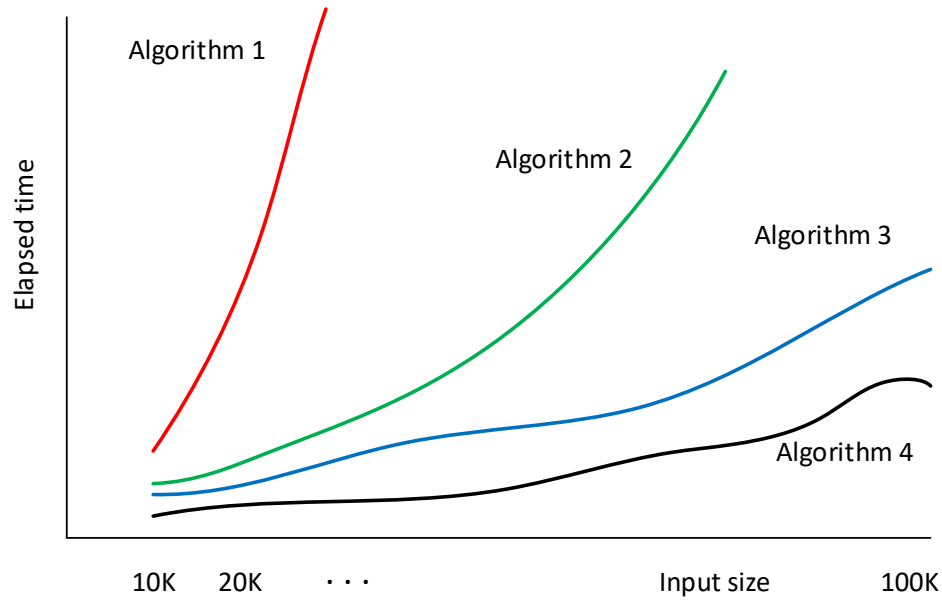
Collect all elapsed times and show the result (1) as a table and (2) as a line graph.

A table should look like:

| *n*<br>*Algorithm* | 10000 | 20000 | 30000 | 40000 | 50000 | 60000 | 70000 | 80000 | 90000 | 100000 |
|---|---|---|---|---|---|---|---|---|---|---|
| insertion | | | | | | | | | | |
| merge | | | | | | | | | | |
| quick | | | | | | | | | | |
| heapsort | | | | | | | | | | |

Entries in the table are elapsed times in milliseconds.

A line graph should show the same information but as a graph with four lines, one for each sorting algorithm. The *x*-axis of the graph is the input size *n* and the *y*-axis of the graph is the elapsed time in milliseconds. Your graph should look like the following example (Note: this is just an example and your graph will look different):




You don't' need to write a Java program to generate the graph. Once you have all elapsed times, you can plot the graph using any other tools, such as a typical spreadsheet software.

Note that, in your result, the running time of an algorithm may not increase as (theoretically) expected. It is possible that the running time of an algorithm may decrease a bit as the input size increases in a part of your graph. As long as the general trend of your graphs are acceptable, there will be no point deduction. So, you should not be too much concerned about that.

Name the program *Hw5_P7.java*.

**Deliverable**

You need to submit the following files:

- *Hw5_P1_P6.pdf*: This file must include:
  - Answers to problems 1 through 5.
  - Discussion/observation of Problem 6: This part must include what you observed and learned from this experiment and it must be "substantive."
- *Hw5_P7.java*
- Other files, if any.

Combine all files into a single archive file and name it *LastName_FirstName_hw5.EXT*, where *EXT* is an appropriate archive file extension, such as *zip* or *rar*.

**Grading**

- Problem 1 through Problem 6:
  - For each problem, up to 6 points will be deducted if your answer is wrong.

- Problem 7:
  - There is no one correct output. As far as your output is consistent with generally expected output, no point will be deducted. Otherwise, up to 30 points will be deducted.
  - If there are no sufficient inline comments, up to 10 points will be deducted.
  - If your conclusion/observation/discussion is not **substantive**, points will be deducted up to 10 points