1. Analysis of non-recursive algorithms.
Consider the problem of determining, given an integer array \( x \), whether \( x \) contains any elements that differ by exactly one.

(a) Write Java code that solves this problem without sorting the array.

```java
/** returns true if \( x \) has two elements differing by exactly one, false if \( x \) does not have two such elements. For example, returns true on \{3,9,3,4\} and false on \{3,7,3,9\}. */
boolean DifferenceOfOne(int[] x)
{
    int i,j;
    int n = x.length;
    for(i=0;i<n;i++) for(j=i+1;j<n;j++)
        if(x[i] == x[j] + 1 || x[j] == x[i] + 1) return true;  // two elements differing by one
    return false;  // loop finished, search failed to find two such elements
}
```

(b) Using \( \Theta \) notation, estimate the worst-case running time of your code in part (a) in terms of the number \( n \) of elements in the array.

Two nested loops; \( \Theta(n^2) \).

(c) Write another solution to the problem that begins by sorting the array.

```java
boolean DifferenceOfOne(int[] x)
{
    Arrays.sort(x);
    int n = x.length;
    for(int i=0;i<n-1;i++)
        if(x[i] == x[i+1] + 1 || x[i] == x[i+1]-1) return true;
    return false;
}
```
(d) Estimate the worst-case running time of your code in (c).

\( n \lg n \) for sorting, one loop after that is constant times \( n \); together it is \( \Theta(n \lg n) \)

(e) Which is faster for large \( n \), the code in (a) or the code in (c)? The code in (c).

2. Indicate, for each pair of expressions (A,B) in the table below, whether A is \( O \), or \( \Theta \) of B. Write “Yes” or “No” in each blank box of the table. [You will only receive points if you answer more than 50% correct, since you can get 50% correct by guessing.]

<table>
<thead>
<tr>
<th>( A )</th>
<th>( B )</th>
<th>( O )</th>
<th>( \Theta )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( n^3 )</td>
<td>( 2^n )</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>( \sqrt{n} )</td>
<td>( n\lg n )</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>( \lg n )</td>
<td>( \lg n^2 )</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>( \lg \sqrt{n} )</td>
<td>( \lg n )</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>( n^3 + \sqrt{n} )</td>
<td>( n^3 )</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>( n^2 )</td>
<td>( n\lg n )</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>( 2^n )</td>
<td>( n! )</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>( \lg n )</td>
<td>( n )</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>

3. Linked lists. The class StringList has these members:

```java
{ String key;
    StringList next;
}
```

Write a method boolean find( String needle) that returns true if needle occurs in the list this, and false if not. For example, if this is the list (“cat”, “dog”, “giraffe”), then find(“dog”) returns true, and find(“anteater”) returns false.

```java
boolean find(String needle)
{
    StringList marker;
    for(marker=this;marker != null; marker=marker.next)
        if(marker.key.equals(needle)) return true;
    return false;
}
```
4. Analyzing recursive code using recurrence relations.

Consider the following variant of MergeSort: instead of dividing the array to be sorted into two, we divide it into three, sort each third recursively, and then merge the three sorted arrays in a way similar to the way two sorted arrays are merged in MergeSort.

(a) Write a recurrence for the worst-case running time $T(n)$, where $n$ is the size of the array being sorted.

$$T(n) = 3T(n/3) + O(n)$$

(b) Solve that recurrence relation to give an estimate for $T(n)$. (“Estimate” means that your answer uses big-O notation.)

$$T(n) = O(n \log_3 n) = O(n \log n)$$

because log to the base 3 differs from lg by a constant factor.

(c) What does this tell us about the worst-case running time? (circle one)

-- it’s the same as MergeSort (up to a constant factor)


The picture represents a binary search tree. The numbers shown are arbitrary node labels, not numbers representing the contents of the nodes. **The contents are not shown.** Please draw the tree after the deletion of the root node (node 1 is the root).
Answer: The successor of node 1 is node 10. So the data in node 10 replaces that in node 1, and node 14 becomes the new left child of node 9.

6. **Red-black trees.** Add the minimum possible number of nodes to the following tree to make it a red-black tree. Indicate (for example using letters R and B) how to color the resulting tree so it is a red-black tree. (Ignore the numbers in the nodes, they are just node labels.)

![Diagram of a tree](image)

Answer: The black height of the root must be as small as possible, which would be 3. If we color 3 and 6 red, then we have to add further black nodes as shown.

![Diagram of a red-black tree](image)
If we color 2 and 5 red instead of 3 and 6, then we have to add a lot more black nodes:

But if anyone submits this tree, I will accept it too, as it does show that you knew what a red-black tree is, and that was the point of the problem.

7. **B-tree insertion.** A new key 4 is to be inserted into the following B-tree. This B-tree has 2, 3, or 4 children per internal node.

Draw a similar diagram showing the B-tree after the insertion of 4. Use the one-pass insertion algorithm, i.e. the one given in your textbook.

*Answer:*
8. B-tree deletion. Delete 8 from this B-tree using the textbook’s algorithm. Show the resulting tree. This tree has 2, 3, or 4 children per internal node.

Answer

9. Quick sort. The following array is to be sorted:

\[
4 \ 14 \ 8 \ 1 \ 10 \ 5 \ 15 \ 13 \ 2 \ 16 \ 6 \ 7 \ 3 \ 9 \ 11 \ 12
\]

(a) if \texttt{quickSort} is used to sort this array, what will the array look like just before the first recursive call?

\[
4 \ 8 \ 1 \ 10 \ 5 \ 2 \ 6 \ 7 \ 3 \ 9 \ 11 \ 12 \ 14 \ 16 \ 15 \ 13
\]

(b) What will the array look like just after the return from the first recursive call?

\[
1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8 \ 9 \ 10 \ 11 \ 12 \ 14 \ 16 \ 15 \ 13
\]

10. Heap algorithms. The following array represents a binary tree.

\[
4 \ 2 \ 11 \ 3 \ 5 \ 1 \ 10 \ 8 \ 7 \ 6 \ 9
\]

(a) Using the convention we used for heaps, draw a picture of this tree.

\textit{Everyone will get this right, so I won’t bother to draw a tree.}
(b) After calling BUILD-MIN-HEAP, the resulting array will be a min-heap. Show it in array form:

| 1 | 2 | 4 | 3 | 5 | 11 | 10 | 8 | 7 | 6 | 9 |   |

11. Execute Prim’s algorithm to find a minimal spanning tree in the following graph, starting from vertex 5. List the vertices in the order they come off the queue, and then highlight the minimal spanning tree by drawing it right on the diagram.

Vertices in the order they come off the queue: 5, 6, 9, 7, 8, 4, 2, 1, 3, 10
12. Execute Kruskal’s algorithm to find a minimal spanning tree in the following graph (which is, incidentally, the same graph as in the previous problem).

List the edges in the order they are added to the answer, and also highlight the minimal spanning tree by drawing it right on the diagram. Since the edges have unique weights, you can identify an edge by its weight, e.g. you can write “11” instead of (4,5).

Edges: 12, 18, 19, 22, 27, 30, 32, 40, 56

13. If we run Dijkstra’s algorithm on the graph shown below, starting from node 2,

(a) how many nodes will be removed from the queue before every node gets its correct distance from node 2 calculated?

Answer [just one number]: 8

(b) List the nodes in the order of their removal from the queue.

Answer [a list of numbers]: 2, 4, 6, 5, 1, 9, 3, 7, 8, 10

When 7 comes off the queue, 8 changes value from 134 to 89, and after than nothing changes value again, so the 8 nodes that come off before that last change are 2, 4, 6, 5, 1, 9, 3, 7.

(a) What are the asymptotic time and space requirements of the Needleman-Wunsch algorithm? Express your answers in terms of the maximum length \( n \) of the two input strings, using big-O notation.

\[
\text{Time: } O(n^2)
\]

\[
\text{Space: } O(n^2)
\]

(b) Execute the Needleman-Wunsch algorithm to find the best alignment of PAIL and POL, scoring exact matches as 1 and non-matches as 0, with a gap penalty of 1 (i.e. matching a letter with a gap scores -1). Use the following table to execute the
algorithm, showing the predecessor arrows as was done in class. (i.e., prefer diagonal, then up, then left).

*Here I used U = up, D = diagonal, L = left because it’s too hard to draw arrows in Word.*

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>-1</th>
<th>-2</th>
<th>-3</th>
<th>-4</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>1D</td>
<td>0L</td>
<td>-1L</td>
<td>-2L</td>
<td></td>
</tr>
<tr>
<td>-2</td>
<td>0U</td>
<td>1D</td>
<td>0D</td>
<td>-1D</td>
<td></td>
</tr>
<tr>
<td>-3</td>
<td>-1U</td>
<td>0D</td>
<td>1D</td>
<td>1D</td>
<td></td>
</tr>
</tbody>
</table>

The best alignment is PAIL

P-OL

and the score of that alignment is 1.

15. Basic properties of some algorithms we studied

(a) Which runs faster, depth-first search or breadth-first search, or do they take the same time?  *The same time*

(b) What algorithm that we studied for graph search has three nested for-loops, each one looping over all vertices?  *Floyd-Warshall*

(c) Could you have used Floyd-Warshall to solve the MazePath homework problem (if you had known that algorithm at the time)?

*No, because it is too slow. For an n by n maze, the number of vertices is n^2, and the cube of that is n^6, so for n even as small as 100 we get a trillion steps, at least 20 minutes.*

(d) Which graph-search algorithm works by repeatedly relaxing all the edges of the graph, as many times as there are vertices?  *Bellman-Ford*

(e) What algorithm did you use to solve the MazePath homework problem?  *Dijkstra’s*

16. Graphs and their representations. In order to ship goods efficiently, the system of “containers” has been developed, so that it costs comparatively little to transfer a shipment from ship to train to truck. Suppose that you are starting a company that will try to improve the efficiency of shipping, by finding more efficient ways to route goods
from the factory to their destinations. (You will charge a fraction of the savings as your fee.) Describe a weighted graph that your engineers would use:

(a) The nodes would be: Factories, destinations, and points where containers can be transferred from one carrier to another (such as ports).

(b) The edges would be: available shipping routes, e.g. an edge between Kobe, Japan and San Francisco because ships run there, and edges for highways on which trucks run from one place where containers can be exchanged to another, and railroad tracks similarly.

(c) The weights would be: The cost per container to ship a container along that edge. One might also want to label edges with the time it takes to ship a container along that edge. (This problem did not ask anything about algorithms to solve any specific problems, but one can imagine that the shipping time is a concern sometimes, as well as the cost.)

17. Huffman codes. Given this table of frequencies:

<table>
<thead>
<tr>
<th>y</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>2</td>
</tr>
<tr>
<td>e</td>
<td>4</td>
</tr>
<tr>
<td>d</td>
<td>5</td>
</tr>
<tr>
<td>a</td>
<td>6</td>
</tr>
<tr>
<td>t</td>
<td>8</td>
</tr>
<tr>
<td>i</td>
<td>9</td>
</tr>
</tbody>
</table>

Construct a Huffman code, and use it to decode the following message:

0000001000110110101101001

Answer: yesididit, or more legibly, “yes i did it”

Here’s my computer-checked work (which I also did by hand)

1: y  2: s  4: e  5: d  6: a  8: t  9: i
3: sy  4: e  5: d  6: a  8: t  9: i
5: d  6: a  7: esy  8: t  9: i
7: esy  8: t  9: i  11: ad
9: i  11: ad  15: esty
15: esty  20: adi
35: adeisty

That yields this tree (written sideways by computer, with indentation showing the level in the tree)

adeisty: 35
esty: 15
esy: 7
sy: 3
So the code is

\[
\begin{align*}
y & = 0000 \\
s & = 0001 \\
e & = 001 \\
t & = 01 \\
i & = 10 \\
d & = 110 \\
a & = 111
\end{align*}
\]

and the given string decodes as

yesididit