Assignment #3

Problem 1.  [30 points]
(a) In the following table, show the worst-case running times for the best algorithms for implementing the
given operations for the dynamic data structure representations listed in the leftmost column.

Notes:
* key: the value of the item
* obj: a pointer to the entry to be removed
* Make sure you have no empty spaces in the middle of your array

<table>
<thead>
<tr>
<th>Data Structure</th>
<th>Search(S,key)</th>
<th>Insert(S,key)</th>
<th>Delete(S,obj)</th>
<th>Min(S)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unsorted Array</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sorted Array</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Unsorted Linked List</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sorted Linked List</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Unsorted Doubly Linked List</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sorted Doubly Linked List</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(b) If you had made any other assumptions while filing up the table above, please state them here.

Problem 2.  [20 points]
Consider the following basic problem.

You’re given an array $A$ consisting of $n$ integers $A[1], A[2], ..., A[n]$. You’d like to output a two-dimensional
$n$-by-$n$ array $B$ in which $B[i, j]$ (for $i < j$) contains the sum of array entries $A[i]$ through $A[j]$ - that is, the
doesn’t matter what is output for these values.)

Here’s a simple algorithm to solve this problem.

```
for $i = 1, 2, \ldots, n$
  for $j = i + 1, i + 2, \ldots, n$
    add up array entries $A[i]$ through $A[j]$
    store the result in $B[i,j]$
  endfor
endfor
```

(a) For some function $f$ that you should choose, give a bound of the form $O(f(n))$ on the running time of this
algorithm on an input of size $n$ (i.e., a bound on the number of operations performed by the algorithm).

(b) Although the algorithm above is a natural way to solve the problem—after all, it just iterates through
the relevant entries of the array $B$, filling in a value for each—it contains some highly unnecessary sources of
inefficiency. Give a different algorithm to solve this problem, with an asymptotically better running time.
Problem 3. [20 points]
Consider the different graph representations discussed in class, and fill out the table below. In the following table, show the worst-case running times for the best algorithms for implementing the given operations. **Note:** Be careful what data structures you will be using for both implementations.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Adjacency Matrix</th>
<th>Adjacency List</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add a vertex</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Remove a vertex</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Find a vertex</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Add an edge</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Remove an edge</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Find an edge</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Problem 4. [15 points]
Remember the Binary Tree data structure? You have studied this in CS230 before, and you might have also worked with it in other courses.

To remind you, a Binary tree is a tree (acyclic graph with a root node), such that each parent node has at most two children nodes. The layout of the tree depends completely on how it was built and the elements in it. The only rule is on the maximum number of children that each node can have.

Given this definition of a Binary Tree, we define the search algorithm as shown below. The input to this algorithm is the root node of the tree, and the element to be found. The output of this algorithm is a boolean indicating if the element has been found in the tree or not.

```
Search(root, element) returns boolean
    if(root.value == element)
        return true
    boolean found = false
    if(root.left != null)
        found = Search(root.left, element)
    end if
    if(found == false && root.right != null)
        found = Search(root.right, element)
    end if
    return found
end of function
```

(a) Compute the worst case running time complexity of the algorithm ($T(n)$). Show your work.
(b) What is the big-Oh of this algorithm? Prove your answer.
(c) What is the big-Omega of this algorithm? Prove your answer.
Problem 5. [15 points]
Now, another similar data structure to a Binary Tree is the Binary Search tree.

To remind you, a Binary Search tree is a tree (acyclic graph with a root node), such that each parent node has at most two children nodes. Moreover, the values of the children on the left have to be less than that of the parent (or equal), and the values of the children on the right have to be more than that of the parent. The layout of the tree still depends completely on how it was built and the elements in it.

Given this definition of a Binary Search tree, we define the search algorithm as shown below. The input to this algorithm is the root node of the tree, and the element to be found. The output of this algorithm is a boolean indicating if the element has been found in the tree or not.

Search(root, element) returns boolean
    if(root.value == element)
        return true
    if(element < root.value)
        if(root.left != null)
            return Search(root.left, element)
        end if
    else if(root.right != null)
        return Search(root.right, element)
    end if
end of function

(a) Compute the worst case running time complexity of the algorithm \( T(n) \). Show your work.

(b) What is the big-Oh of this algorithm? Prove your answer.

(c) What is the big-Omega of this algorithm? Prove your answer.