1. (25%, 5% each) Short questions. For each question you **must** explain your answer. Write neatly. I cannot give you credit if I cannot read your answer.

   a. Why would we use a binary search tree instead of a binary search algorithm using an array?

   b. What advantage does one have when using an array implementation of linked list as opposed to a circular array for implementing a queue? Remember in a queue one only inserts and removes from either end.

   c. Why do we not let the queue get totally full in a circular array implementation of a queue?

   d. Suppose you wanted an ADT that can insert, remove and search quickly. What is the best ADT that we have covered so far.

   e. Suppose you wanted an ADT that can insert, remove and search quickly (like in part d) but the platform that you are working on does not support dynamic memory allocation. How can we implement the ADT from (d) on this platform.
2. (16%, 8% each) a. Find the time complexity in terms of N of the following segment of code. In this case the complexity is the number times the W++ gets executed. This is also the value of W after the code is executed. Note \( \sum_{i=m}^{n} k = k(n - m + 1) \) and \( \sum_{i=1}^{n} i = \frac{n(n + 1)}{2} \).

```c
for (x = 0; x < N*N; x++)
  for (y = x; y > 0; y = y - 4)
    for (z = 0; z < N; z++)
      w++;  
```

b. Find the time complexity in terms of N of the following segment of code. In this case the complexity is the number times the W++ gets executed. This is also the value of W after the code is executed.

```c
x = 1;
While (x < N)
  { 
    for (z = 0; z < N; z++)
      w++;
    x = x * 2;
  }
```
3. (15%) Does the following binary search algorithm work. If not what is wrong with and how can you fix it.

```c
int BinaryFind( int data[N],
    int front,
    int end,
    int entry)
{
    if (end - front < 2)
    {
        if (data[front] == entry)
            return front;
        else if data[end] == entry)
            return end;
        else
            return -1;
    }
    else
    {
        middle = (end + front) / 2;
        if (data[middle] < entry)
            return BinaryFind( data, middle + 1, end, entry);
        else
            return BinaryFind( data, front, middle - 1, entry);
    }
}
```

4. (14%) Write a segment of code to insert a new node after the node pointed to by P. Assume there is a node on both sides of the node you are inserting. The following structure is used.

```c
struct node
{
    int data;
    node *next;
    node *prev;
};

node *P;
```
5. (15%) Trees
   a. Show the following tree after inserting a 7.5, 0.3 and 3.5.

   ![Tree after insertions](image1)

   b. Show the following binary tree after deleting the 2 and the 6.

   ![Binary tree after deletions](image2)

   c. What is the output of the following function when given the tree below.

   ```c
   Int Add(binary_tree_node *T)
   {
     int sum = 0;

     if (T != NULL)
       {
         sum = sum + Add(T->Lchild) * 2;
         sum = sum + Add(T->Rchild) * 10;
         sum = sum + T->data;
         return sum;
       }
     else
       return 0;
   }
   ```

   ![Function call](image3)

   output = ________________
6. (15%) Having seen the array implementation of linked list, the following is an array implementation of a tree.

a. Given the following array, show the tree it represents.

```c
struct node {
    int   Lchild;
    char  str[80];
    int   Rchild;
}

node   Array[8];
```

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>-1</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>7</td>
<td>9</td>
</tr>
<tr>
<td>5</td>
<td>-1</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>7</td>
<td>-1</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>-1</td>
<td>10</td>
</tr>
<tr>
<td>9</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>10</td>
<td>-1</td>
<td>-1</td>
</tr>
</tbody>
</table>

b. Show the array after inserting Joe as the left child of Barbara. Note the empty list still forms a linked list. The Lchild field is ignored and the Rchild field is used as the Next field.