This exam has 10 questions, for a total of 100 points.

1. **8 points** What is the output of the following Python program?

```python
def dub(x):
    return 2 * x
class A:
    def __init__(self, x):
        self.d = []
        for i in range(0,6):
            self.d.append(x * i)
    def m(self, b, c, f):
        print(b[1])
        print(c[3])
        print(self.d[5])
        print(f(21))
a = A(2)
a.m([1,2], {3: 4, 5: 6}, dub)
```

**Solution:** (2 points each)

```
2
4
10
42
```

2. **15 points** Complete the following implementation of a Stack (Last-In-First-Out) by filling in the blanks.

```python
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None
class Stack:
    def __init__(self):
        self.head = None
    def insert_front(self, x):
        ___(a)___
        self.head = ___(b)___
    def erase_front(self):
        self.head = ___(c)___
    def push(self, x):
        self.insert_front(x)
    def pop(self):
        result = ___(d)___
        ___(e)___
        return result
```

3. 5 points Draw the result of inserting a node with key 7 into the following Binary Search Tree.

```
      16
     /   \
    5     17
   / \   /  \
  1   10 17   21
```

Solution:

```
      16
     /   \
    5     17
   / \   /  \
  1   10 7   21
```

4. 12 points For each of the nodes 5, 10, and 21, list the successor and predecessors of the node (if they exist).

```
      13
     /   \
    5     17
   / \   /  \
  1   10 15   21
```

Solution:

- For 5, 10 is the successor and 1 is the predecessor.
- For 10, 13 is the successor and 5 is the predecessor.
- For 21, there is no successor and 17 is the predecessor.
5. **9 points** Use the Tree walk method below to define a function named `print_tree` that has one parameter, a tree, and prints the tree so that the arithmetic operators are printed using infix notation and so that parentheses surround the arguments to each operator and integer. For example, the tree

```python
tree = Tree(Node('+', Node('+', Node(1), Node(2)), Node(3)))
```

should be printed as

```plaintext
(((1)+(2))+(3))
```

Here are the definitions of the Tree and Node classes.

```python
class Node:
    def __init__(self, key, left=None, right=None):
        self.key = key
        self.left = left
        self.right = right
    def recursive_walk(self, f):
        f('pre', self.key)
        if self.left:
            self.left.recursive_walk(f)
        f('in', self.key)
        if self.right:
            self.right.recursive_walk(f)
        f('post', self.key)
class Tree:
    def __init__(self, root):
        self.root = root
    def walk(self, f):
        if self.root:
            self.root.recursive_walk(f)

```

Hint: you will need to define a second function that you pass as an argument to the `f` parameter of the `walk` method.

**Solution:**

```python
def tree_printer(state, key):
    # (2 points)
    if state == 'pre':
        print('(', end='')
    elif state == 'in':
        print(key, end='')
    else:
        print(')', end='')

def print_tree(T):
    T.walk(tree_printer)  # (3 points)
```

6. **9 points** Given the following AVL tree, delete node 3 and re-balance the tree if necessary using one or more left or right rotations. State which rotations you applied to which nodes.
Solution:
To delete node 3, we replace it with its successor, node 4. (3 points)

But now node 4 is not AVL, so we rotate node 4 to the right. (3 points)

But now node 5 is not AVL, so we rotate 5 to the left. (3 points)
7. **10 points** What is the best DNA sequence alignment for the sequences AGTG and GAGT? When computing the score, use +1 for a match, -1 for a mismatch, and -1 for the gap penalty. Show your work by filling in the below dynamic programming table.

```
<table>
<thead>
<tr>
<th></th>
<th>G</th>
<th>A</th>
<th>G</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>G</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>G</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

**Solution:** One solution is

```
_AG TG
GAGT_
```

```
<table>
<thead>
<tr>
<th></th>
<th>G</th>
<th>A</th>
<th>G</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>←-1</td>
<td>←-2</td>
<td>←-3</td>
<td>←-4</td>
</tr>
<tr>
<td>A</td>
<td>↑-1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>G</td>
<td>↑-2</td>
<td>←-1</td>
<td>↑-1</td>
<td>↑-2</td>
</tr>
<tr>
<td>T</td>
<td>↑-3</td>
<td>↑-1</td>
<td>←-1</td>
<td>↑0</td>
</tr>
<tr>
<td>G</td>
<td>↑-4</td>
<td>←-2</td>
<td>↑-2</td>
<td>↑0</td>
</tr>
</tbody>
</table>
```

(2 points for correct initialization of the table. 5 points for the rest of the table. 3 points for correct result from traceback.)

8. **12 points** Complete the following implementation of `max_heapify`, which takes an instance of Heap `H` that contains an array (the `data` field) that represents a max-heap, except that the node at position `i` breaks the max-heap property. The `max_heapify` function turns the array into a max-heap.

```python
class Heap:
    def __init__(self, array):
        self.data = array
        self.heap_size = len(data)
    def swap(A, i, j):
        tmp = A[i]
        A[j] = tmp
    def max_heapify(H, i):
        l = 2 * i + 1
        r = ____(a)____
        if l < H.heap_size and ____ (b) ____:
            largest = l
        else:
            largest = r
        if H.data[largest] > H.data[i]:
            H.data[largest], H.data[i] = H.data[i], H.data[largest]
```

Page 5 of 7
largest = i
if r < H.heap_size and H.data[largest] < H.data[r]:
    largest = r
if largest != i:
    swap(H.data, ___(c)___, largest)
max_heapify(H, ___(d)___)

Solution: (3 points each)
(a) 2 * (i + 1)
(b) H.data[i] < H.data[1]
(c) i
(d) largest

9. 10 points Show that \( f(n) = 3n^3 - 2n^2 + 4 \in O(n^3) \). Start your answer by giving the definition of big-O notation.

Solution: The definition of big-O is: (2 points)

\[
O(g(n)) = \{ f(n) \mid \exists n_0. \forall n \geq n_0. 0 \leq f(n) \leq cg(n) \}
\]

So we need to show that

\[
3n^3 - 2n^2 + 4 \leq cn^3
\]

for all \( n \) greater than some \( n_0 \) and for some suitable choice of \( c \). Solving for \( c \), we get

\[
3 - \frac{2}{n} + \frac{4}{n^3} \leq c
\]

(2 points for some argument about \( \leq \))

For \( n \geq 1 \) we have

\[
3 - \frac{2}{n} + \frac{4}{n^3} \leq 5
\]

so we choose \( c = 5 \) (3 points) and \( n_0 = 1 \) (3 points).

10. 10 points What is the big-O time complexity of the following function in terms of the input size \( n \)? Explain why your answer is correct.

def bottom_up_cut_rod(p, n):
    r = [-1 for range(0,n+1)]
    r[0] = 0
    for j in range(1, n+1):
        q = -1
        for i in range(1,j+1):
            q = max(q, p[i] + r[j-i])
        r[j] = q
    return r[n]
Solution: The time complexity is $O(n^2)$ (4 points). The inner loop is $O(n)$ (3 points) and the outer loop executes $n$ times (3 points), so the whole thing is $O(n^2)$. 