BSTs & AVL Trees – Q1

Why would you use an AVL tree versus a Binary Search Tree?

- Faster Search/Insert/Delete in a balanced tree versus an unbalanced tree.
- In a balanced tree the Run-time of Search/Insert/Delete is $O(\log n)$
  - but if a branch becomes deep the Run-time approaches $O(n)$. 
Show the state of the AVL tree after deleting node 48 and doing any necessary rebalancing:
Show the state of the AVL tree after deleting node 48 and doing any necessary rebalancing:
Show the state of the AVL tree after deleting node 48 and doing any necessary rebalancing:
What are the PreOrder, InOrder, and PostOrder traversals of the following Binary Tree?

- **PreOrder:** 5,8,7,1,4,3,2,9,6
- **InOrder:** 1,7,4,8,3,5,2,6,9
- **PostOrder:** 1,4,7,3,8,6,9,2,5
What is the height of the following tree?

8
Write a recursive function to free the memory in a Binary Tree:

```c
void FreeBST(node *root) {
    if (root != NULL) {
        FreeBST(root->left);
        FreeBST(root->right);
        free(root);
    }
}
```
What index would 8 be inserted into in the following hash table using Quadratic Probing with the hash function $x^2 + 7 \mod 13$:

<table>
<thead>
<tr>
<th>index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>val</td>
<td></td>
<td></td>
<td></td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

10
What is the purpose of a hash table?

- Very fast search, insert, and delete times: $O(1)$ with a perfect hash function.
What are the two uses for Heaps given in class?

- Priority Queues and Heap Sort.
What is the resulting heap after Deleting the Minimum element from the following heap?
What is the resulting heap after Deleting the Minimum element from the following heap?
What is the resulting heap after Deleting the Minimum element from the following heap?
Using Big-O notation, what is the run-time of:

(a) Inserting 10 items into an initially empty binary heap
(b) Inserting 10 items into a binary heap with \( n \) elements.

- \( O(1) \)
- \( O(\log n) \)
Fill in the table to show the resulting array after each pass in Bubble Sort:

<table>
<thead>
<tr>
<th>Initial</th>
<th>4</th>
<th>2</th>
<th>6</th>
<th>5</th>
<th>7</th>
<th>1</th>
<th>8</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sorted</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
</tbody>
</table>
### Sorting – Q1

- Fill in the table to show the resulting array after each pass in Bubble Sort:

<table>
<thead>
<tr>
<th>Initial</th>
<th>4</th>
<th>2</th>
<th>6</th>
<th>5</th>
<th>7</th>
<th>1</th>
<th>8</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>1</td>
<td>7</td>
<td>3</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>5</td>
<td>1</td>
<td>6</td>
<td>3</td>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>1</td>
<td>5</td>
<td>3</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>4</td>
<td>3</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sorted</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
</tbody>
</table>
Show the result of running Partition on the array below using the leftmost element as the pivot element. Show the array after each swap.

<table>
<thead>
<tr>
<th>Initial</th>
<th>4</th>
<th>2</th>
<th>6</th>
<th>5</th>
<th>7</th>
<th>1</th>
<th>8</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Swap1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Swap2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Partitioned</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Sorting – Q2

Show the result of running Partition on the array below using the leftmost element as the pivot element. Show the array after each swap.

<table>
<thead>
<tr>
<th>Initial</th>
<th>4</th>
<th>2</th>
<th>6</th>
<th>5</th>
<th>7</th>
<th>1</th>
<th>8</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Swap1</td>
<td>4</td>
<td>2</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>Swap2</td>
<td>4</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>7</td>
<td>5</td>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td>Partitioned</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>7</td>
<td>5</td>
<td>8</td>
<td>6</td>
</tr>
</tbody>
</table>
Fill in the table to show the array after each call to the Merge function in Merge Sort.

<table>
<thead>
<tr>
<th>Initial</th>
<th>5</th>
<th>2</th>
<th>6</th>
<th>4</th>
<th>7</th>
<th>1</th>
<th>8</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sorted</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
</tbody>
</table>
Fill in the table to show the array after each call to the Merge function in Merge Sort.

<table>
<thead>
<tr>
<th>Initial</th>
<th>5</th>
<th>2</th>
<th>6</th>
<th>4</th>
<th>7</th>
<th>1</th>
<th>8</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2</td>
<td>5</td>
<td>6</td>
<td>4</td>
<td>7</td>
<td>1</td>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>5</td>
<td>4</td>
<td>6</td>
<td>7</td>
<td>1</td>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>1</td>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>1</td>
<td>7</td>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>1</td>
<td>7</td>
<td>3</td>
<td>8</td>
</tr>
<tr>
<td>Sorted</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
</tbody>
</table>
What is the Worst Case run-time of Insertion Sort, Selection Sort, and Bubble Sort respectively?

What is the Best Case of each?
- $O(n^2)$, $O(n^2)$, $O(n^2)$
- $O(n)$, $O(n^2)$, $O(n^2)$
What is the Best Case and Worst Case for finding the kth smallest integer out of an unsorted array of n integers. (k <= n)

- Best Case: O(n) , Worst Case: O(n^2)
Stacks & Queues – Q1

- What is the acronym for describing the push and pop rules for Stacks and what does it stand for?
  - LIFO – Last In, First Out.
Show the final contents of the Array-Implemented Queue, the index of front, and numElements – after running this code:

```
enqueue(Q1, 8);
enqueue(Q1, 3);
dequeue(Q1);
enqueue(Q1, 6);
enqueue(Q1, 7);
dequeue(Q1);
enqueue(Q1, 9);
```

<table>
<thead>
<tr>
<th>Q1: elements:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

front:  
numElements:  

Show the final contents of the Array-Implemented Queue, the index of front, and numElements – after running this code:

```java
enqueue(Q1, 8);
enqueue(Q1, 3);
dequeue(Q1);
enqueue(Q1, 6);
enqueue(Q1, 7);
dequeue(Q1);
enqueue(Q1, 9);
```

**FRONT:** -1 0 1 2

**NUMELEMENETS:** 0 1 2 3
What two implementations of Queue’s were used in HW #4? What was each one used for?

- Array implementation – Router
- Linked List implementation – each device’s request queue.
What are the run-times of the following operations:

- Stacks: Push and Pop
- Queues: Enqueue and Dequeue

O(1) for all
Convert the following infix expression to postfix:

\[( A / ( B - C ) + D ) * ( E - F ) + G * H \]

\[A \ B \ C - \ / \ D + \ E \ F - \ * \ G \ H \ * \ +\]
Algorithm Analysis – Q1

- What is the Big-O run-time of deleting one node from an AVL tree with \( n \) nodes?
- What is the Big-O run-time of deleting one node from an AVL tree with height \( h \)?

- \( O(\log n) \) and \( O(h) \)
What is the Big-O solution to the following recurrence relation?

\[ T(n) = 2T(n/2) + n, \text{ assume } T(1) = 1 \]

\[ O(n \log n) \]
Determine a simplified closed-form solution for the following summation in terms of n:

\[
\sum_{i=1}^{3n} \sum_{j=n+1}^{5n} (5i + 3j)
\]
Determine a simplified closed-form solution for the following summation in terms of \( n \):

\[
\sum_{i=1}^{3n} \sum_{j=n+1}^{5n} (5i + 3j)
\]
What is the Big-O running time of the following segment of code, it terms of $n$.

```c
int a = 1, b = n, sum = 0;
while (a < b) {
    sum++;
    a = a*2;
    b = b/2;
}
```
What is the Big-O running time of the following segment of code, in terms of \( n \).

```c
int a = 1, b = n, sum = 0;
while (a < b) {
    sum++;
    a = a*2;
    b = b/2;
}
```

Consider the ratio \( b/a \).

The loop stops when this ratio is 1. For each loop iteration the ratio decreases by a factor of 4. Let \( k \) be the number of loop iterations total. Then \( 1 = n/4^k \). Solving we get \( k = \log_4 n \). \( \Rightarrow \) \( O(\log n) \)
Algorithm Analysis – Q5

- If an $O(n^2)$ algorithm takes 40 ms to complete with an input size of $n = 20,000$, how much time will it take to complete on an input size of $n = 50,000$?

  - $c \cdot n^2 = 40\text{ms}$, $c = \frac{40}{20,000^2} = \frac{40}{400,000}$
  
  - $40 \div 400,000 \times (50,000^2) = \frac{40}{400,000} \times (2,500,000)$
  
  - $= 10 \times 25 = 250\text{ ms}$
Fill in the blanks of the following recursive sorting function, which of the sorting algorithms that we have seen so far does this resemble:

```c
void sort(int *values, int length) {
    if (length > 1) {
        int maxIndex = 0;
        int i;
        for (i=1; i<length; i++)
            if (________(1)________)
                maxIndex = i;
        int temp = values[length-1];
        values[length-1] = ____ (2) ____;
        __________ (3) __________ = temp;
        ______________ (4) ______________;
    }
```
Fill in the blanks of the following recursive sorting function, which of the sorting algorithms that we have seen so far does this resemble?

- Selection sort.

```c
void sort(int *values, int length) {
    if (length > 1) {
        int maxIndex = 0;
        int i;
        for (i=1; i<length; i++)
            if ( values[i] > values[maxIndex] )
                maxIndex = i ;
        int temp = values[length-1];
        values[length-1] = values[maxIndex];
        values[maxIndex] = temp ;
        sort(values, length - 1);
    }
}
```
In a binary search of the array below, which elements in the array are checked (and in what order) when a search is conducted for the number 17?

47, 9, 22
Briefly explain what the function does AND what its return value means. (Using the typical tree node struct)

```c
int mystery(struct node *root) {
    int retVal;
    if(root == NULL)
        return 0;
    retVal = mystery(root->left) + mystery(root->right);
    if(root->data % 2 == 1) {
        root->data -= 1;
        retVal ++;
    }
    return retVal;
}
```
The function subtracts 1 from all nodes containing odd values.

The function returns the number of nodes altered by the function (number of odd nodes).

```c
int mystery(struct node *root) {
    int retVal;
    if (root == NULL)
        return 0;
    retVal = mystery(root->left) +
             mystery(root->right);
    if (root->data % 2 == 1) {
        root->data -= 1;
        retVal ++;
    }
    return retVal;
}
```
Imagine using a linked list of digits to store an integer. For example, a list containing 3, 6, 2, and 1, in that order stores the number 3621. Write an iterative function which accepts a linear linked list `num` that stores a number in this fashion and returns the value of the number. You may assume the list stores digits only and contains 9 or fewer nodes.

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

int getValue(struct node* num) {
    // Fill in code
}
```
Imagine using a linked list of digits to store an integer. For example, a list containing 3, 6, 2, and 1, in that order stores the number 3621. Write an iterative function which accepts a linear linked list `num` that stores a number in this fashion and returns the value of the number.

```c
int getValue(struct node* num) {
    int sum = 0;
    while (num != NULL) {
        sum = 10*sum + num->data;
        num = num->next;
    }
    return sum;
}
```
What is the Big-O running time of the following segment of code, in terms of $n$.

```c
int i;
for (i=0; i<n; i+=2) {
    for (j=i; j>0; j--)
        printf("%d", j);
    printf("\n");
}
```

- The inner loop will run $0+2+4+...+n$ times
- Since we know $0+1+2+3+...+n = n(n+1)/2 = O(n^2)$
- We would have about $\frac{1}{2}$ of $O(n^2) = O(n^2)$