1. (a) $T(n) = 4T(n/2) + 3n^2 - 9n$
$a = 4 \;\; b = 2 \;\; f(n) = 3n^2 - 9n = 3n(n-3)$
$n^{\log_2(4)} = n^2$
$f(n) = \Theta(n^2)$
case 2 is met therefore:
$T(n) = \Theta(n^2 \log n)$

   (b) $T(n) = 4T(n/2) + 2n^3 - 100n^2$
$a = 4 \;\; b = 2 \;\; f(n) = 2n^3 - 100n^2 = 2n^2(n-50)$
$n^{\log_2(4)} = n^2$
$f(n) = \Omega(n^{2+\epsilon})$\{for any positive constant $\epsilon \leq 1$\} AND
$4f(n/2) = 4(2(n/2)^2((n/2) - 50)) = n^2(n-100) \leq$
$c(2n^2(n-50))$\{for any positive constant $1 > c \geq 1/2$\}
case 3 is met therefore:
$T(n) = \Theta(f(n)) = \Theta(n^3)$

   (c) $T(n) = 4T(n/2) + n + 5\log n$
$a = 4 \;\; b = 2 \;\; f(n) = n + 5\log n$
$n^{\log_2(4)} = n^2$
$f(n) = O(n^{2-\epsilon})$\{for any positive constant $\epsilon \leq 1$\}
case 1 is met therefore:
$T(n) = \Theta(n^2)$

   (d) $T(n) = 8T(n/2) + n^2 + n\log n$
$a = 8 \;\; b = 2 \;\; f(n) = n^2 + n\log n$
$n^{\log_2(8)} = n^3$
$f(n) = O(n^{3-\epsilon})$\{for any positive constant $\epsilon \leq 1$\}
case 1 is met therefore:
$T(n) = \Theta(n^3)$

   (e) $T(n) = 8T(n/2) + 4n^3 - 5n^2$
$a = 8 \;\; b = 2 \;\; f(n) = 4n^3 - 5n^2 = n^2(4n-5)$
$n^{log_2(8)} = n^3$
$f(n) = \Theta(n^3)$
case 2 is met therefore:
$T(n) = \Theta(n^3 \log n)$

   (f) $T(n) = 8T(n/2) + 2^{-10}n^4 - 6n^3$
$a = 8 \;\; b = 2 \;\; f(n) = 2^{-10}n^4 - 6n^3 = 2n^3(\frac{1}{2048}n - 3)$
$n^{log_2(8)} = n^3$
$f(n) = \Omega(n^{3+\epsilon})$\{for any positive constant $\epsilon \leq 1$\} AND
$8f(n/2) = 8(2(n/2)^3(\frac{1}{2048}(n/2) - 3)) = 2n^3(\frac{n}{4096} - 3) \leq$
$c(2n^3(\frac{1}{2048}n - 3))$\{for any positive constant $1 > c \geq 1/2$\}
case 3 is met therefore:
$T(n) = \Theta(f(n)) = \Theta(n^4)$

2. (a) $T(n) = 2T(n/2) + \frac{n}{\log n}$
This reccurence cannot be solved using the master theorem because the difference between $\frac{n}{\log n}$ and $n^{\log_2(2)}$ is not polynomial.

(b) $T(n) = 2^n(n/2) + n^n$
This reccurence cannot be solved using the master theorem because a is not constant, therefore the number of sub-problems is not fixed.

3. (a) The algorithm would recursively trace through the left half of the tree, record the sum of the nodes in the left half, trace through and record the sums for the right half, then compare the two and return the reference to the node holding the higher sum.

(b)
```
Node findMaxWeight(Node root) {
if(root != null){
        Node left = findMaxWeight(root.left);
        Node right = findMaxWeight(root.right);
        int leftWeight = sum(left);//helper method
        int rightWeight = sum(right);//helper method
        if(leftWeight>rightWeight){return left;}
        return right;
        }
}
```

(c) $T(n) = 2T(n/2) + 2n$
$a = 2 \;\; b = 2 \;\; f(n) = 2n$
$n^{\log_2(2)} = n$
$f(n) = \Theta(n)$ case 2 is met therefore:
$T(n) = \Theta(n \log n) = O(n \log n)$

4. (a) The pros of BSTs are that they can always guarantee insertion because of the linked-list structure of the tree, the size of the input is only limited by the hardware of the machine itself, and all of the elements are inserted in a specific order such that the call to print all elements only takes O(n) time. The main con is that the creation of all of the nodes in the BST require a considerable amount of memory from the system's resources. The best time to use a BST is when you know that the amount of data being stored will be dynamic, that way you can take advantage of the BST's linked-list structure.

(b) The major pro of a hash table is that the insertion and retrieval functions are theoretically only 1 constant-time operation, making those functions incredibly efficient. The cons are that you must allocate the size of the table before indexing the data, making the data structure static rather than dynamic, you must deal with hash collisions which are highly probable and this could take up to O(n) time, and the elements inserted in the table will be unsorted. The best time to use a hash table would be if you already know the size/amount of data to be indexed and know that there will be no need for the data to be sorted, only inserted, removed, or retrieved.

https://brackece.wordpress.com/2012/09/18/hash-table-vs-binary-search-tree/