Question 1) (20 points) For each function below with input argument n, determine the asymptotic number of “basic operations” that will be executed. Justify your answer for each case.

**Note:** For the recursive functions, you should first write the corresponding recurrence relation. Then solve the recurrence relation to come up with the asymptotic bound.

\theta(1) \ \theta(\log n) \ \theta(n) \ \theta(n \log n) \ \theta(2^n) \ \theta(n \log n^2) \ \theta(n^2) \ \theta(n^3) \ \theta(n!) \ \text{Other? Please specify.}

**a)**
```c
void func(int n) {
    if(n>1)
    {
        func(n-1);
        Perform n basic operations;
    } //endif
}
```

**b)**
```c
void func(int n) {
    if (n > 3)
    {
        func(n/4);
        func(n/4);
        func(n/4);
        func(n/4);
        Perform n basic operations;
    } //endif
}
```

**c)**
```c
void func(int n) {
    if (n > 1)
    {
        func(n/2);
        func(n/2);
        Perform 1 basic operation;
    } //endif
}
```

**d)**
void func(int n) {
    int i=n;
    while (i>0)
    {
        Perform 1 basic operation;
        i=i/4;
    }//endwhile
}

Question 2)

a) (8 points) Explain how heap data structures are different from binary search trees (BSTs). Provide at least two main differences and explain each.

b) (12 points) Apply the Heap Sort algorithm to sort the following list in ascending (non-decreasing) order. In addition to drawing the tree step by step, you should draw the array after each step. Show all your work.

int list[]={5, 6, 9, 8, 2, 1}

Question 3) (20 points) Consider two singly sorted linked lists, L1 and L2, each of which is sorted in ascending (non-decreasing) order. Assume L1 has n entries and L2 has m entries, where n, m>=0. Each entry has two components: a key component of type int and the usual next link component.

a) (15 points) Write a C++ Merge function to merge two given lists L1 and L2 in-place. That means your code should result in a singly merged list sorted in ascending order without creating a new list. Your Merge function should return a pointer to the head node of the merged list.

Here is an example of how merging would work. Assume the first linked list L1 has 4->35->95 and the other linked list L2 has 1->7->20->35, your code will produce 1->4->7->20->35->35->95 without using extra space.

b) (5 points) Analyze the time complexity of your code in part (a) in the worst-case. Justify your analysis.