Problem 1:
Base case: T(2) = 2 = 2log2(2)

Assume:
T(n/2)= (n/2)log2(n/2)

1. T(n) = 2T(n/2) + n
2. T(n) = 2((n/2)log2(n/2)) + n
3. T(n) = nlog2(n/2) + n
4. T(n) = n(log2(n)−1) + n
5. T(n) = nlog2(n)

->proven.

Problem 2:
To sort A[1…n], we recursively sort A[1…n-1] then insert A[n] into the sorted array

T(n) = T(n-1) + A[n]

To insert, we compare A[n] with elements in A[1…n - 1] from n - 1, if the value of A[n] is smaller than all other numbers in the array, then it takes T(n - 1) comparisons to find its position.
Base case: T(1) = 0 since there's only 1 comparison and it takes n - 1 comparisons
=> 1-1 = 0
Worst case running time: T(n) = T(n - 1) + (n - 1) (n > 1, T(1) = 0)

Problem 3:

Pseudocode:

```
sortingNegativePositive(arr) {
   left = 0; //the begin of the array
   right = size of arr - 1;  //the end of the array
   while (left < right) { //stop when left meets right
      if (arr[left] < 0):
         left = left + 1; //move to the right
      else if (arr[right] > 0):
         right = right - 1; //move to the left
      else:
         swap arr[left] to arr[right];
         left = left + 1;
         right = right - 1;
```

};
}

Problem 4:

-In a binary search tree, each element can possibly have maximum two children

=>n = 2^k - 1 nodes total when we have a perfectly balanced tree
=>2^k = n + 1
=>k = log2(n+1)

-The number of nodes at level i only is 2^(i-1)
Sn = sum (i = 1 -> k) i x 2^(i-1)
Sn = (k - 1)2^k + 1
Sn = (k - 1)(n + 1) + 1
Sn = (log2(n + 1) - 1)(n + 1) + 1
Sn = (n + 1)log2(n + 1) - n - 1 + 1
Sn = (n + 1) log2(n + 1) - n

Average = Sn/n = ((n + 1) log2(n + 1) - n) / n
An = log2(n + 1) - 1

Base case is 1
Worst case is floor log2(n) + 1
Average case = log2(n) - 1