When asking for designing an algorithm, both English description and pseudo-code are required. When asking for an efficient algorithm, the score will depend on how efficient your algorithm is.

1. (20 points) Suppose the array \( A \) contains \([1, 6, 4, 9, 7]\). (a) Please display the contents of the array whenever its content changes during the execution of the heapsort algorithm. (b) Please do the same for the quicksort algorithm where the first element in the subarray is used as pivot.

Answer: Heapsort:
Create the max-heap: \([1,6,4,9,7] \Rightarrow [1,9,4,6,7] \Rightarrow [9,1,4,6,7] \Rightarrow [9,7,4,6,1]\)
Remove-min from the heap: \([1,7,4,6 \mid 9] \Rightarrow [7,1,4,6 \mid 9] \Rightarrow [7,6,4,1 \mid 9] \Rightarrow [1,6,4 \mid 7,9] \Rightarrow [6,1,4 \mid 7,9] \Rightarrow [4,1,6,7,9] \Rightarrow [1,4,6,7,9] \Rightarrow [1,4,6,7,9].\)

Quicksort: \([1,6,4,9,7] \Rightarrow [1,4,6,9,7] \Rightarrow [1,4,6,7,9].\)

2. (20 points) You are inserting the following elements 1, 6, 4, 9, 7, in the given order into a search tree. Please display the sequence of the trees whenever the tree changes if the tree is (a) Red-black trees; (b) 2-3 trees.

Answer: omitted.

3. (15 points) Given an \( m \times m \) matrix \( A \) and an integer \( n \), provide an efficient algorithm to compute \( A^n \) (\( n \) multiplications of \( A \)) and analyze its complexity.

Answer: We may use quickpower to compute \( A^n \):

```java
quickpower(A, n) {
    if (n == 1) return A;
    if (n is even) { B = quickpower(A, n/2); return B*B; }
    else return A*quickpower(A, n-1);
}
```

The number of * in the above algorithm is \( O(\log n) \). If * is computed using the standard matrix multiplication, the total cost is \( O(m^3\log n) \). If the divide-conquer approach is used, the total cost is \( O(m^{2.81}\log n) \).

4. (15 points) Sometimes we need to merge \( k \) sorted lists into one sorted list. For simplicity, we assume that each of the \( k \) sorted lists has \( n \) elements. Please provide an efficient
merge algorithm to do this job and analyze the complexity of your algorithm in terms of \(k\) and \(n\).

Answer: There are two solutions.
(a) Use the divide-and-conquer approach: If \(k = 2\), we merge the two lists and the cost is \(2n\). If \(k > 2\), we divide \(k\) lists into two groups of \(k/2\) lists and recursively solve each group and then merge the two results into one. Let \(T(k)\) be the cost of merging \(k\) lists. Then \(T(1) = 0\), \(T(2) = 2n\) and \(T(k) = 2T(k/2) + kn\). The solution is \(T(k) = O(nk \log k)\).
(b) Use a priority queue to store the \(k\) lists, the key of each list is the least element in the list. To merge there \(k\) lists, we remove the list with the least element in the queue, put the least element in the result and put the remaining list into the queue if it’s not empty. There are at most \(nk\) insert and remove-min queue operations. Each queue operation takes \(O(\log k)\) time. So the total time is \(O(nk \log k)\).

5. (15 points) Given a binary tree, the length of the longest path connecting any pair of nodes is called the **diameter** of the tree. Please provide an efficient algorithm to compute the diameter of the tree when the root of the tree is given as input. Please also analyze the complexity of your algorithm.

Answer: Let \(P\) be a longest path in the tree and \(x\) be the vertex in \(P\) which is closest to the root. Then the length of \(P\) is the sum of the length from \(x\) to the farthest point in its left branch and to the farthest point in its right branch. We just need to compute the farthest child for each node, which is the depth of the subtree rooted by that node.

```c
int diameter(root) { dia = 0; depth(root); return dia; }

int depth(node x) { // return the distance of a farthest descendent from x.
    int fl = -1, fr = -1;
    if (x.left != nil) fl = depth(x.left);
    if (x.right != nil) fr = depth(x.right);
    if (fl + fr +2< dia) dia = fl+fr+2;
    if (fl > fr) return 1+fl; else return 1+fr;
}
```

The complexity is \(O(n)\) as each node of the tree is visited once.

6. (15 points) Given a list of numbers, an **inversion** is a pair of numbers that are out of order. For example, there are three inversions in the list \([1, 4, 3, 2]\), i.e., \((4, 3), (4, 2),\) and \((3, 2)\). Please provide an efficient algorithm to count the number of inversions of a given list and analyze its complexity.

Answer: We may modify mergesort to compute the number of inversions as follows: mergesort will sort the array as well as return the number of inversions. When the array
is divided into two parts, the number of inversions comes from three places: from the first part, from the second part, and from the inversions where the larger element is in the first part and the smaller element is in the second part. The Java code is given below:

```java
static int mergesortInv(int low, int high) {
    // Check if low is smaller than high, if not then the array is sorted
    if (low < high) {
        // Get the index of the element which is in the middle
        int middle = low + (high - low) / 2;
        // Sort the left side of the array
        int i1 = mergesortInv(low, middle);
        // Sort the right side of the array
        int i2 = mergesortInv(middle + 1, high);
        // Combine them both
        int i3 = mergeInv(low, middle, high);
        return i1+i2+i3;
    }
    return 0;
}

static int mergeInv(int low, int middle, int high) {
    // Copy first part into the arrCopy array
    for (int i = low; i <= middle; i++) arrCopy2[i] = arr[i];

    int s = 0;
    int i = low;
    int j = middle + 1;
    int k = low;

    // Copy the smallest values from either the left or the right side back
    // to the original array
    while (i <= middle && j <= high) {
        if (arrCopy2[i] <= arr[j]) {
            arr[k] = arrCopy2[i];
            i++;
        } else {
            arr[k] = arr[j];
            j++;
            s += middle-i+1; // the number of remaining elements in the left side.
        }
        k++;
    }

    // Copy the rest of the left part of the array into the target array
    while (i <= middle) {
        arr[k] = arrCopy2[i];
        k++;
        i++;
    }
    return s;
}
```