3. (i) $\Theta(n^{\lg_2 3})$

   (ii) $\Theta(n^{\lg_6 5})$

   (iii) $\Theta(n \lg n)$

   (iv) $\Theta(n^2 \lg n)$

   (v) $\Theta(n^{2.5})$

   (vi) $\Theta(n^2)$

   (vii) $\Theta(2^{n/2})$

   (viii) $\Theta(\lg \lg n)$ Why? Well, ...

$$\begin{aligned} T(n) &= T(n^{1/3}) + 3 \\ &= T(n^{1/9}) + 3 + 3 \\ &= T(n^{1/27}) + 3 + 3 + 3 \\ &\dots \end{aligned}$$

   So $T(n) = 3x + T(c)$, where c is the first integer where the recursive calls would stop, and x is the smallest integer such that $n^{1/3^x} \leq c$. Assume $c = 2$. Then we want $n^{1/3^x} \leq 2$. Take the log base 2 of each side. Then we want $\lg(n^{1/3^x}) = 1/3^x \lg n \leq \lg 2 = 1$. I.e. we want $\lg n \leq 3^x$. What is the smallest x such that $3^x \geq \lg n$? Take the log base 3 of each side, we want $\lg_3(3^x) = x \geq \lg_3(\lg n)$. So x is in $\Theta(\lg(\lg n))$.

4. (i) $P_1 + P_2 = A(F - H) + (A + B)H = AF - AH + AH + BH = AF + BH$

   (ii) $T(n) = 3T(n/2) + 3691 * (n/2) * n/2 = 3T(n/2) + \Theta(n^2)$, so by the master theorem, $T(n)$ is in $\Theta(n^2)$.

   (iii) I don't know how to prove that it is impossible.

   Right now, the best known algorithm for multiplying matrices is a variation of the Coppersmith-Winograd algorithm, taking time $O(n^{2.37\dots})$. So giving an algorithm that runs in $O(n^2)$ would be an exciting breakthrough in algorithms.

5. (i) Mergesort takes 5 key comparisons. Or you could reason like this. Label the items A,B,C,D. Compare A,B, and C,D. Then compare the larger of each of these pairs. So now we know the max of A,B,C,D. Now we want to sort the other three. We already have one comparison made among them. So with 2 more comparisons we can sort these three elements. So we have sorted the four elements with $3 + 2 = 5$ comparisons.

   (ii) There are many ways to argue this. Here is one argument. Suppose A,B is the first comparison made. Then at some point in the algorithm, C must be compared with D (otherwise, there are 2 different permutations that the algorithm cannot distinguish between). So we can assume that A,B and C,D are two comparisons made by the algorithm. Now, in the rest of the algorithm, we must find the max of max(A,B) and max(C,D). That requires one comparison, assume w.l.o.g. that the max is A. Now we need to sort B,C,D, and we have already made one comparison among these 3 (C,D). But sorting 3 elements requires 3 comparisons in the worst case, so we might need 2 more. The total is $1 + 1 + 1 + 2 = 5$.

   Here is another argument. There are $4! = 24$ possible permutations of the 4 keys. Each comparison rules out at most 0.5 of the possible permutations. So the minimum number of comparisons will be the ceiling of the binary log of 24, which is 5.

6. In the best case, the items from the shorter list are removed, so the minimum number of key comparisons will be the minimum of $\{x, y\}$. Once the shorter list is empty, all the remaining items can be placed with no further key comparisons.

7. M3sort takes an unsorted list of $n \geq 1$ elements and works as follows. If $n \leq 5$, use a perfect sorting algorithm and return. Otherwise, recursively M3sort the first $n/3$ elements, copy into sublist A. Then recursively M3sort the second $n/3$ elements, copy into sublist B. Then recursively M3sort the remaining elements, copy into sublist C. Then merge lists A and B, copy into list D. Then merge lists D and C, return this list.

   (i) argue by induction. The base cases are covered, because we use a perfect sorting algorithm wen the list has at most 5 elements. if the list has 6 or more elements, then the recursive calls are correct, so the sublists are sorted, so all we need to do is check that we recombine the solutions properly. but merging sorted sublists leaves a sorted list, so after the first merge B is sorted, so B and C (by induction hypothesis) are sorted, so the final list is sorted.

   (ii) $T(n)$ is a constant if n is at most 5. For $n \geq 6$, $T(n) = T(a) + T(b) + T(c) + a + b - 1 + a + b + c - 1$, where $a + b + c = n$. Notice that this sum is minimized if we make a and b as small as possible. So the best choice for this algorithm is to set a and b each the floor of $n/3$, and let c be the remainder. So the recurrence in this case is, for $n \geq 6$, $T(n) = 2T(\lfloor n/3 \rfloor) + T(n - \lfloor n/3 \rfloor) + 2\lfloor (n/3) \rfloor - 1 + n - 1$.

   (iii) $\Theta(n \lg n)$

   (iv) It is possible to sort 5 keys using 7 key comparisons in the worst case. The best choices for a, b, c are mentioned in (ii). Here is some python code, with output:

```
X = [0,0,1,3,5,7]
for n in range(len(X),37):
  a,b,c = n/3, n/3, n - (n/3+n/3)
  X.append(X[a]+X[b]+X[c] + a+b-1 + a+b+c-1)
for j in range(1,len(X)):
  print '%2s' % j,'%3s' % X[j],"   ",
  if (j%6 == 0): print ''
```

```
 1   0     2   1     3   3     4   5     5   7     6  11
 7  14     8  17     9  22    10  25    11  28    12  33
13  36    14  41    15  44    16  49    17  53    18  61
19  65    20  69    21  75    22  79    23  85    24  89
25  95    26  99    27 109    28 113    29 117    30 123
31 127    32 133    33 137    34 143    35 147    36 157
```

8. In python, you can traverse a list in reverse sorted order like this:

   `for j in reversed(sorted(G)):` .

   So you could modify the dfsList procedure from the webnotes and run the modified program to check your answer. The dfs preorder sequence is `L H K G C D J I E A B F`. The dfs postorder sequence is `D C G K H L B A E I J F`. In preorder, a tree root appears first. In postorder, a tree root appears last. So we can reconstruct the nodes of the trees of the forest: `LHKGC, JIEAB, F`. In this case, we can reconstruct the forest exactly. Do you see how?