1.

a) Lines (3) through (6) take $O(1) + T(n-2)$ time. Lines (2) through (6) and (1) through (6) do as well.

```
              if
            (1)–(6)
            /     \
        retn        if
        (1)       (2)–(6)
                  /     \
              retn       block
              (2)       (3)–(6)
                      /   /   \    \
                   (3)  (4)  (5)  retn
                                   (6)
```

b) Lines (3) through (4) take $O(n) + 2T(n/2)$ time. Lines (2) through (4) and (1) through (4) do as well.

```
              if
            (1)–(6)
            /     \
        retn        if
        (1)       (2)–(6)
                  /     \
              retn       block
              (2)       (3)–(4)
                        /    \
                      (3)    retn
                              (4)
```

2.

```
LIST split(LIST list, int n)
{
    LIST rest;

    if (n == 0) {
        rest = list->next;
        list->next = NULL;
        return rest;
    }
    else return split(list->next, n-1);
}
```

```
LIST kmergesort(LIST list, int k)
{
    if (k < 2) return NULL;
    else if (list == NULL) return NULL;
    else if (list->next == NULL) return list;
    else return kmerge(list, length(list), k, k-1);
}

LIST kmerge(LIST list, int len, int k, int n)
{
    LIST SecondList;

    if (list == NULL) return NULL;
    else if (list->next == NULL) return list;
    else {
        SecondList = split(list, n*len/k);
        return merge(kmergesort(SecondList, k),
                     kmerge(list, len, k, --n));
    }
}
```

a) The running time of `merge` is $O(n)$. The running time of the `split` procedure different from the book takes $O(n)$ time. Lines (1) through (4) each take $O(1)$ and line (5) takes $O(n)$. Thus `split` takes $O(1) + O(n)$ time which is $O(n)$.

For `kmerge`, lines (1) and (2) each take $O(1)$ time. Line (3) calls `split` with a length proportional to the length of `list`. This takes $O(n)$ time. Line (4) takes $O(n)$ time for the call to `merge`. Since I wrote `kmergesort` and `kmerge` in terms of each other, I do not know what to do next. Certainly `kmerge` is at least $O(k)$ because the size of `kmerge` reduces by 1 starting from $k-1$. Each call to `kmerge` calls `merge`, so it is at least $O(kn)$. But `kmerge` calls `kmergesort`, which also calls `kmerge`.