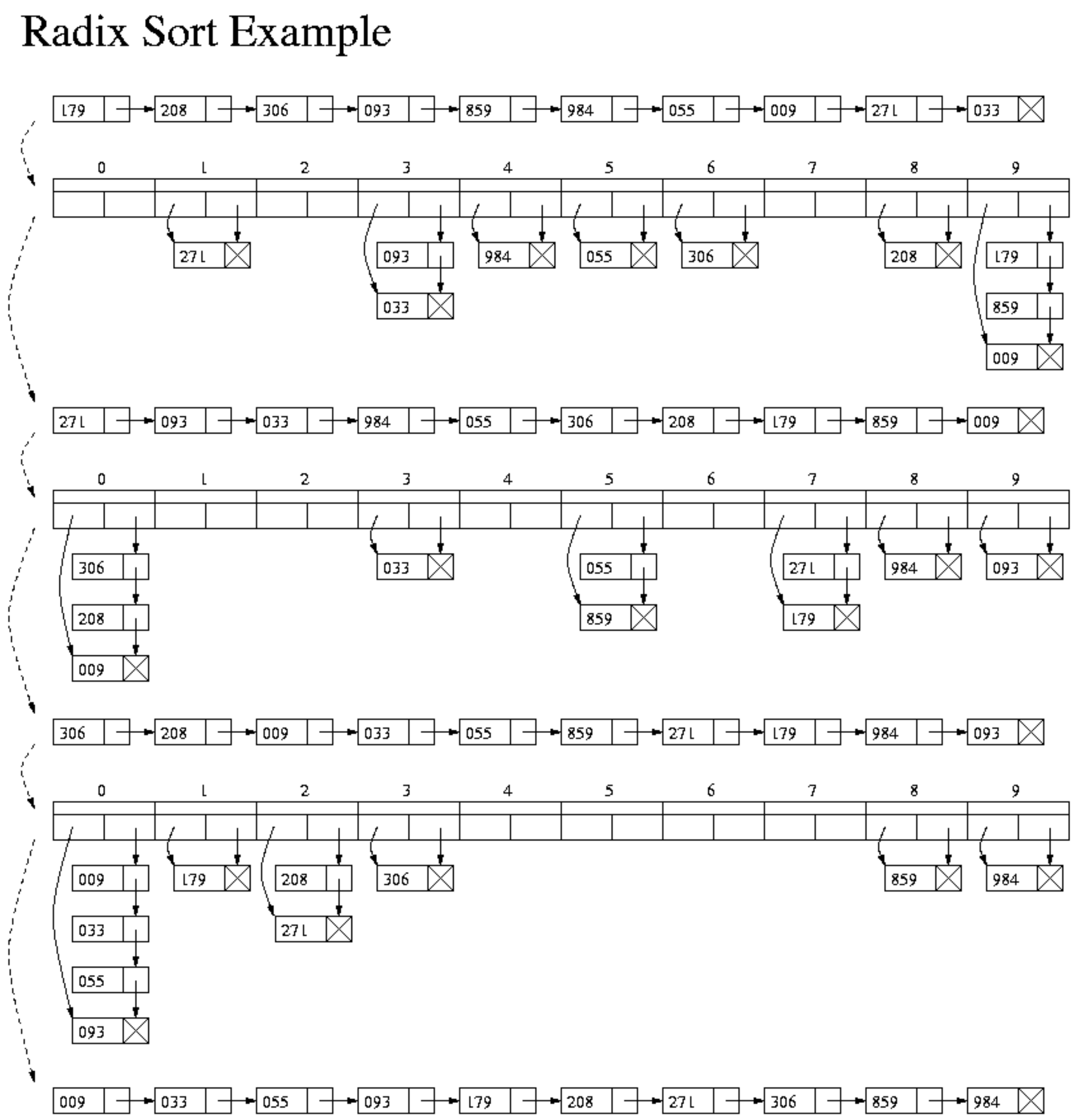
Give recurrences T(n) and S(n) for the time and space complexity of:

```
public static void get_space(int level, int [] A)
{
      int [] B;     int i, n;
      n= A.length;
      if (n==1) return;

      for (i=1; i <= n; i++)
      {
            B= new int[n-1];
            get_space(level+1, B);
      }
}
```

Radix Sort



http://www.cosc.canterbury.ac.nz/tad.takaoka/alg/sorting/radixsort.gif

Radix Sort

Radix sort is a fast algorithm which can be used to sort k-digit integers base r (the radix).

radixSort(L). Input: linked list L.

Action: the cells on L are rearranged so that the list is sorted.

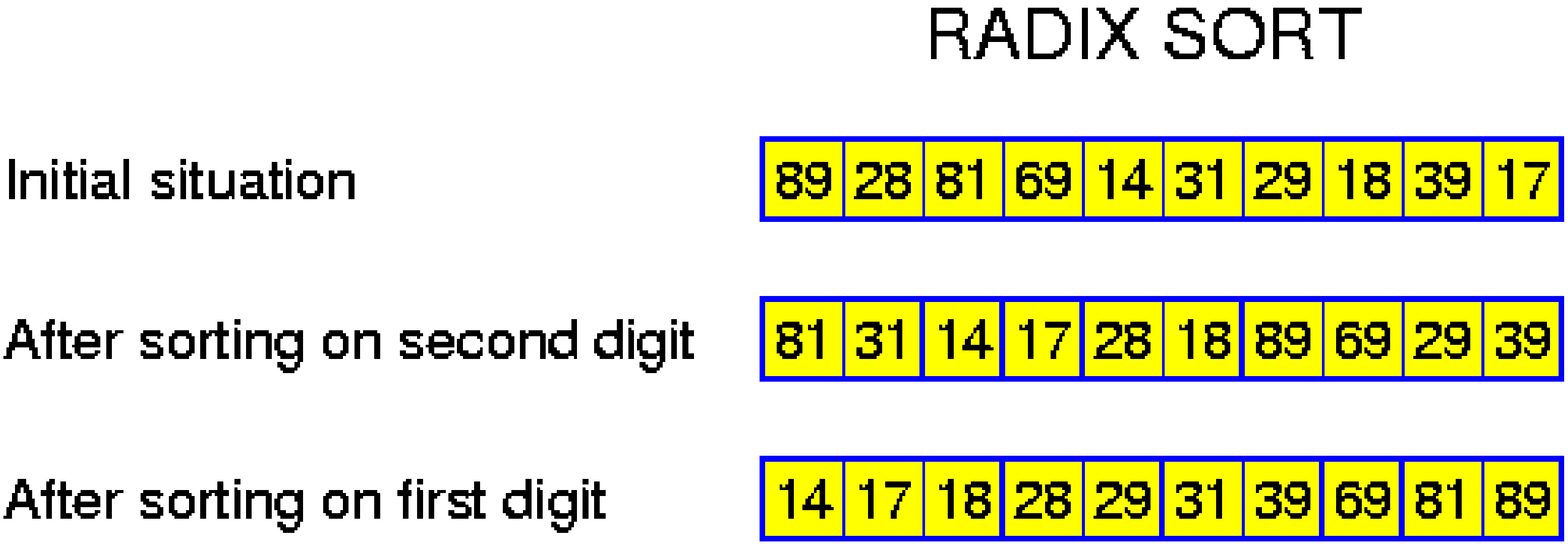
The digits of the integer x are numbered as

$x = d_{k-1}, d_{k-2}, \ldots, d_2, d_1, d_0$.

$x = d_{k-1}, d_{k-2}, \dots, d_2, d_1, d_0.$

Pseudo code

```
for (i=0; i < k; i++)
    for (j=0; j < r; j++)
        Set L_j to be an empty list.
    while (L is not empty) do
        Take the first cell off the front of L.
        Let d be digit i of the key value x stored
        in this cell. Add this cell to the end of
        the list L_d.
    endwhile
    Set L to be an empty list.
    for (j=0; j < r; j++) Append L_j to the end of L.
```



http://users.informatik.uni-halle.de/~jopsi/dinf204/radix_sort.gif

This algorithm works because it is stable: amongst keys with equal value, their relative orders are preserved. The formal proof of correctness applies the following loop invariant.

Loop invariant:
In the outer for loop, just before the iteration with a particular value of i, the integers in L are sorted according to the values induced by their last i digits, $d_{i-1}, \ldots, d_2, d_1, d_0$.

Proof (by induction).
[Basis] This statement implies that before the iteration with i=0, they are not sorted at all. This is trivially true.

Induction step] Assume that just before the iteration with a particular value of i, the integers in L are sorted according to the integers induced by their last i digits. We want to prove that after the iteration with i, the values in L are sorted according to the integers induced by their last i+1 digits,

$d_i, d_{i-1}, \ldots, d_2, d_1, d_0$.

They are placed into the linked lists (L_d's) so that things that are last in the array end up at the end of the lists. Now when you append things together, the integers are ordered according to their ith digit d_i. Amongst those with the same ith digit, they fall into the same order as they were in L and hence by induction, these are sorted by $d_{i-1}, d_{i-2}, \ldots, d_2, d_1, d_0$. So at the end of this iteration, the values are sorted according to $d_i, d_{i-1}, \ldots, d_2, d_1, d_0$.

Note that this same technique could also be used to sort for other data types such as strings.

Suppose for example you wanted to sort strings of length k over the 26 character alphabet {a-z}. You could then use 26 lists, one for each character.

What is the time for radix sort?

If the integers have k digits then it takes time $\Theta(k\ n + kr)$ which is in $\Theta(n)$ if k and r are constants.

This is not a contradiction to the assertion that any comparison model sorting algorithm takes $\Omega(n \log n)$ time:

Radix sort examines individual digits of the data items which is not allowed in the comparison model.