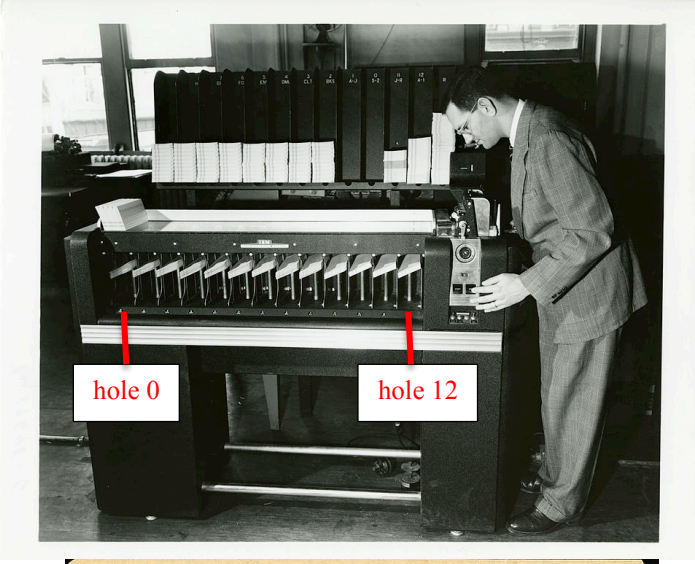
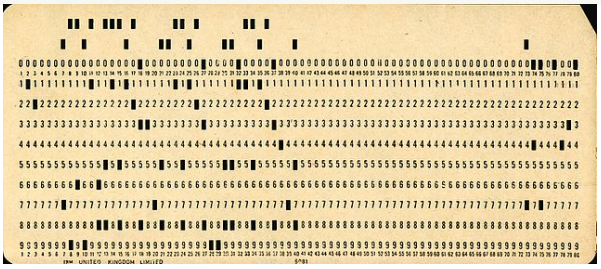
# Pigeonhole sort

The man in the picture[1] to the right is performing a *Pigeonhole sort* using an IBM card sorter. Punch cards are fed into the hopper below his chin. The IBM card sorter's read head has been set to sort based on the holes punched in a particular column of each card. Depending on the holes punched in that column, each card is placed in one of the thirteen baskets, or pigeon holes. (We have labeled the first and last pigeon hole for quick identification.)




Assuming a number from 0 to 9 is punched in the column, all cards with a 0 in the column would be fed into pigeon hole 0, those with a 1 in the column would be fed into pigeon hole 1, and so on. The operator would then gather the cards with those in hole 9 on the bottom, then those in hole 8, etc. with those in hole 0 on the top.

The operator has sorted the cards on that one column using a stable Pigeonhole sort.[2]



To the right is an image of an IBM punched card. Punched cards were in heavy use throughout most of the 20th century. There were different forms of punched cards, but this was one of the most used ones. In academia, programs were punched on such cards from about 1960 to about 1980. One used a key-punch machine to punch the cards.

To the right, we give a version of Pigeonhole sort. Here are points to consider.

1. For simplicity, the method sorts the array elements themselves. See entry "sort" in the JavaHyperText for a version that allows one to sort on any function of the array elements.
2. The method determines how many pigeon holes to create based on the maximum value in the array. The function to compute the max is not given here.
3. The pigeon holes are implemented as `ArrayLists`.
4. The method has three tasks, corresponding to what the operator does. (1) Make sure the pigeon holes are empty. (2) Put the cards into the hopper and have them fed into the pigeon holes based on their value. (3) Pick up the cards out of the pigeon holes in the appropriate order.
5. The space complexity is O(`n` + `m`), where `n` is the size of array `c` and `m` is the maximum value in `c`. The time complexity is also O(`n` + `m`).

```
/** Sort array c. The sort is stable.
  * Precondition: c.length > 0. The c[i] satisfy 0 <= c[i]. */
public static void pigeonholeSort(int[] c) {
   // (1) Create s empty pigeon holes phole[0..s-1].
   int s= 1 + (max value in c); // number of pigeon holes
   ArrayList<Integer>[] phole= new ArrayList[s];
   for (int k= 0; k < s; k= k + 1)
      pholes[k]= new ArrayList<>();

   // (2) Feed the cards (in array c) into the pigeon holes
   for (int t : c) phole[t].add(t);

   // (3) Move the cards from the pigeon holes back to c.
   int k= 0;
   for (ArrayList<Integer> ph : phole) {
      // move cards in pigeon hole ph to c.
      for (int ob : ph) { c[k]= ob;  k= k + 1; }
   }
}
```

---

[1] Picture taken from https://upload.wikimedia.org/wikipedia/commons/thumb/7/7e/SEACComputer_038.jpg/960px-SEACComputer_038.jpg

[2] Suppose columns 1..3 contain a decimal integer. To sort the cards based on that integer, perform a pigeonhole sort on column 3, then on column 2, and finally on column 1. This is known as a radix 10 sort, which is also discussed in JavaHyperText.