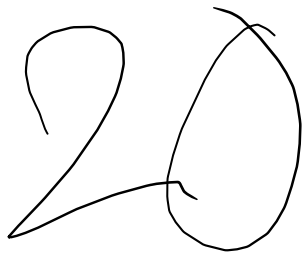
Let’s solve it

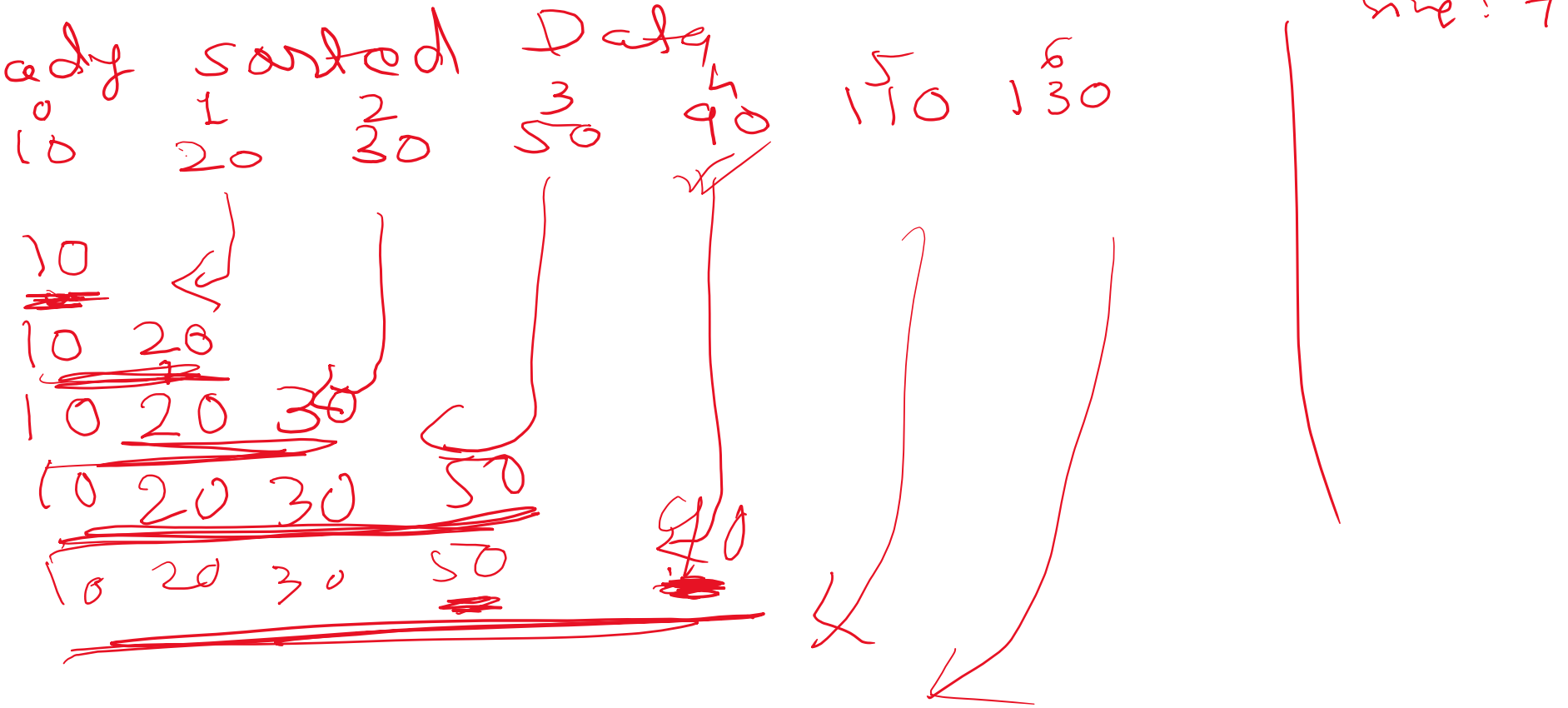


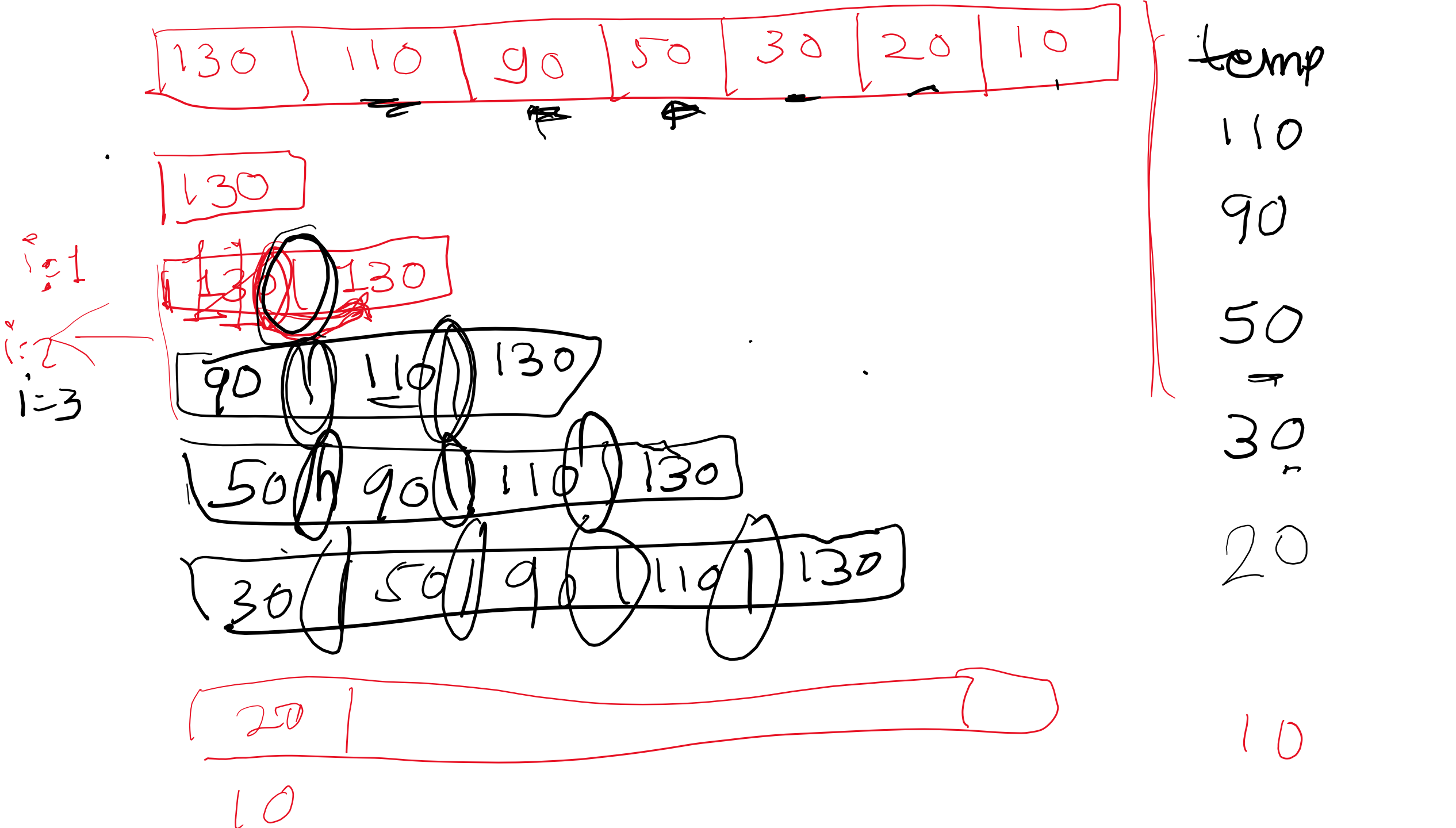
Insertion Sort

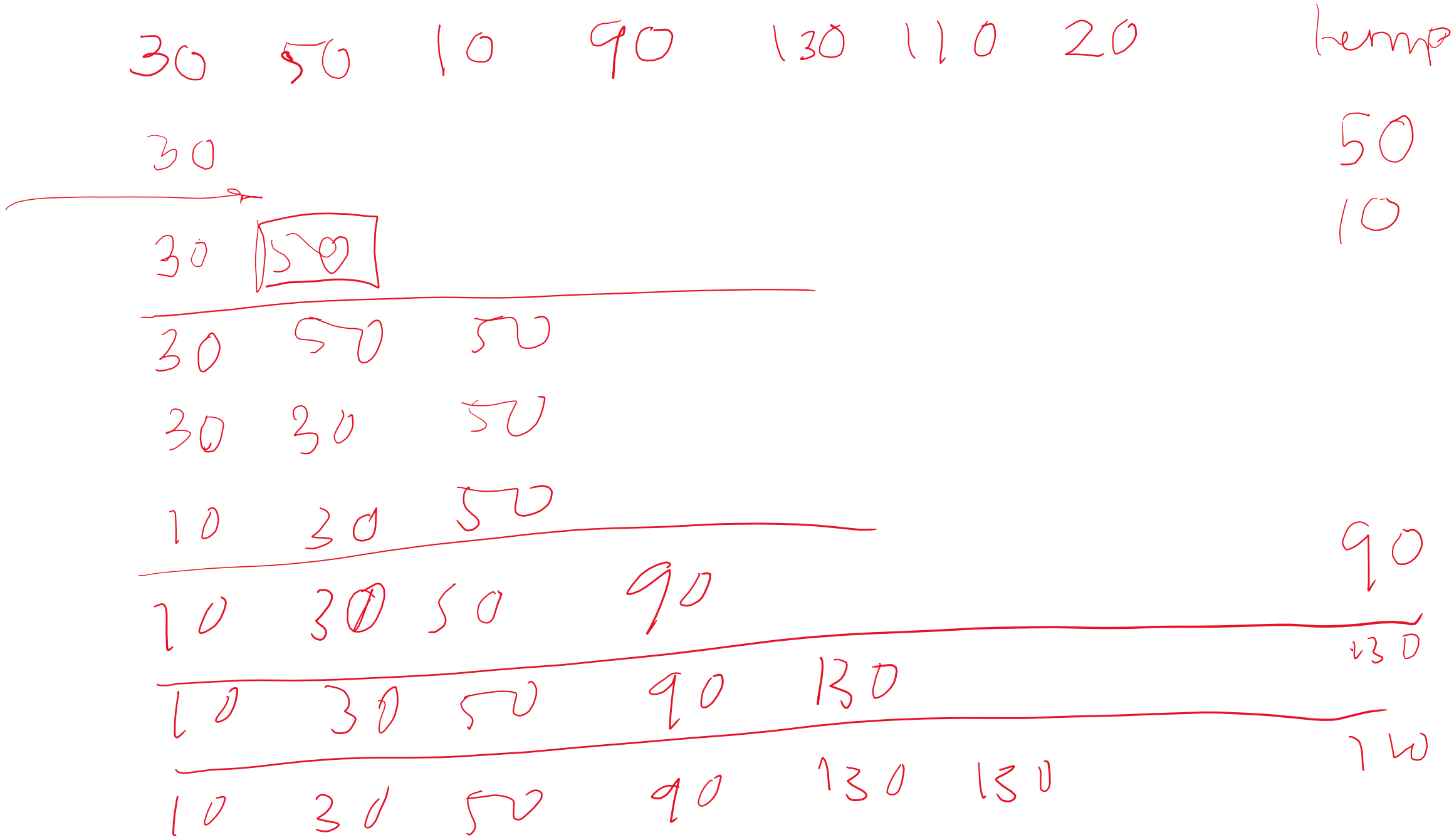
Sort while you insert.

Ascending - A–Z . 1 to 100

Already sorted Data




130 110 90 50 30 20 10
temp
110
90
50
30
20
10
130
i=1
i=2
i=3
130 130
90 110 130
50 90 110 130
30 50 90 110 130
20
10


30 50 10 90 130 110 20
temp
30
50
10
30 50
30 50 50
30 30 50
10 30 50
90
10 30 50 90
130
10 30 50 90 130
10
10 30 50 90 130 150

10 30 50 90 130 150

10 30 50 90 130 130

10 30 50 90 110 130

20

10 30 50 90 110 130 130

10 30 50 90 110 110 130

10 30 50 90 90 110 130

10 30 50 90 90 110 130

10 30 30 50 90 110 130

10 20 30 50 90 110 130

```
for(i=1; i<7; i++)
{
    tobeInserted = dataAr[i];
    for(j=i-1; j>=0; j--)
    {
        if(temp < ar[j])
        {
            ar[j+1] = ar[j];
        }
        else
            break;
    }
    ar[j+1] = temp;
}
```

```
|| j!=0
   i-j+1 > i
```

```
for(i=1;i<n;i++)
{
  int temp;
  temp = array[i];

  for(j=i-1;j>=0;j--)
  {
   if(temp<array[j])
   {
       array[j+1] = array[j];
   }
   else
     break;
  }

  array[j+1] = temp;

 }
```

```
i=1;
while(i<n)
{
  temp = array[i];

  j=i-1;
  while((j>=0)&&(temp<array[j]))
  {
      array[j+1] = array[j];
      j--;
  }

  array[j+1] = temp;
  i++;

 }
```

Notice that the left hand side part of the array starting single element sorted grows with ordering insert logic into sorted array step by step at every insertion. Hence, while inserting new element into it from right side end we only compare with last most, which in turn the biggest sorted thus far.

Also, notice that this program changes will use same memory and hence it is in-place technique.

If duplicates are present notice the behaviours of their relative location. If the pre sorting order of existence* is maintained then it is stable sorting technique. * amongst duplicates i.e

$\tilde{5_A}\ 5_B\ 5_C \Rightarrow 5_A\ 5_B\ 5_C$

And not $5_B\ 5_A\ 5_C$ like