Algorithm 1 Merger Sort

Input: $Array$, n
Output: Reverse order array

```
1: function MergerSort(Array, left, right)
2:     result ← 0
3:     if left < right then
4:         middle ← (left + right)/2
5:         result ← result+ MergerSort(Array, left, middle)
6:         result ← result+ MergerSort(Array, middle, right)
7:         result ← result+ Merger(Array, left, middle, right)
8:     end if
9:     return result
10: end function
11:
12: function Merger(Array, left, middle, right)
13:     i ← left
14:     j ← middle
15:     k ← 0
16:     result ← 0
17:     while i < middle and j < right do
18:         if Array[i] < Array[j] then
19:             B[k++] ← Array[i++]
20:         else
21:             B[k++] ← Array[j++]
22:             result ← result + (middle − i)
23:         end if
24:     end while
25:     while i < middle do
26:         B[k++] ← Array[i++]
27:     end while
28:     while j < right do
29:         B[k++] ← Array[j++]
30:     end while
31:     for i = 0 → k − 1 do
32:         Array[left + i] ← B[i]
33:     end for
34:     return result
35: end function
```

Algorithm 2 Framework of ensemble learning for our system.

Input: The set of positive samples for current batch, P_n; The set of unlabelled samples for current batch, U_n; Ensemble of classifiers on former batches, E_{n-1};

Output: Ensemble of classifiers on the current batch, E_n;

1: Extracting the set of reliable negative and/or positive samples T_n from U_n with help of P_n;
2: Training ensemble of classifiers E on $T_n \cup P_n$, with help of data in former batches;
3: $E_n = E_{n-1} cup E$;
4: Classifying samples in $U_n - T_n$ by E_n;
5: Deleting some weak classifiers in E_n so as to keep the capacity of E_n;
6: **return** E_n;

Algorithm 3 An example for format For & While Loop in Algorithm

1: **for** each $i \in [1, 9]$ **do**
2: initialize a tree T_i with only a leaf (the root);
3: $T = T \cup T_i$;
4: **end for**
5: **for all** c such that $c \in RecentMBatch(E_{n-1})$ **do**
6: $T = T \cup PosSample(c)$;
7: **end for**;
8: **for** $i = 1$; $i < n$; $i++$ **do**
9: // Your source here;
10: **end for**
11: **for** $i = 1$ to n **do**
12: // Your source here;
13: **end for**
14: // Reusing recent base classifiers.
15: **while** $(|E_n| \leq L_1) and (D \neq \phi)$ **do**
16: Selecting the most recent classifier c_i from D;
17: $D = D - c_i$;
18: $E_n = E_n + c_i$;
19: **end while**

Algorithm 4 Conjugate Gradient Algorithm with Dynamic Step-Size Control

Input: $f(x)$: objective funtion; x_0: initial solution; s: step size;
Output: optimal x^*

initial $g_0 = 0$ and $d_0 = 0$;
repeat
    compute gradient directions $g_k = \bigtriangledown f(x_k)$;
    compute Polak-Ribiere parameter $\beta_k = \frac{g_k^T(g_k - g_{k-1})}{\|g_{k-1}\|^2}$;
    compute the conjugate directions $d_k = -g_k + \beta_k d_{k-1}$;
    compute the step size $\alpha_k = s/ \parallel d_k \parallel_2$;
until $(f(x_k) > f(x_{k-1}))$

Algorithm 5 My algorithm

procedure MyProcedure
    stringlen ← length of *string*
    $i \leftarrow$ *patlen*
top:
    if $i >$ *stringlen* **then return** false
    end if
    $j \leftarrow$ *patlen*
loop:
    if *string*(i) = *path*(j) **then**
        $j \leftarrow j - 1$.
        $i \leftarrow i - 1$.
        goto *loop*.
        close;
    end if
    $i \leftarrow i + \max(delta_1(string(i)), delta_2(j))$.
    goto *top*.
end procedure