

```
1 #include "Sort.h"
2 #include "DataGenerator.h"
3 #include "AuxiliaryFunction.h"
4 #include <iostream>
5
6 void bubbleSort(int a[], int n) {
7     for (int i = 0; i < n - 1; i++) {
8         for (int j = 0; j < n - i - 1; j++) {
9             if (a[j] > a[j + 1]) Swap(a[j], a[j + 1]);
10        }
11    }
12 }
13
14 void insertionSort(int a[], int n) {
15     for (int i = 1; i < n; i++) {
16         int x = a[i], j = i - 1;
17         while (j >= 0 && a[j] > x) {
18             a[j + 1] = a[j];
19             j--;
20         }
21         a[j + 1] = x;
22     }
23 }
24
25 void selectionSort(int a[], int n) {
26     for (int i = 0; i < n - 1; i++) {
27         int minId = a[i];
28         for (int j = i + 1; j < n; j++) {
29             if (a[j] < a[minId]) minId = j;
30         }
31         Swap(a[i], a[minId]);
32     }
33 }
34
35 void shakerSort(int a[], int n) {
36     int l = 0, r = n - 1, k = 0;
37     while (l < r) {
38         for (int i = l; i < r; i++) {
39             if (a[i] > a[i + 1]) {
40                 Swap(a[i], a[i + 1]);
41                 k = i;
42             }
43         }
44         r = k;
45         for (int i = r; i > l; i--) {
46             if (a[i] < a[i - 1]) {
47                 Swap(a[i], a[i - 1]);
48                 k = i;
49             }
50         }
51         l = k;
52     }
53 }
```

```
50     }
51     l = k;
52 }
53 }
54
55 void shellSort(int a[], int n) {
56     for (int gap = n / 2; gap > 0; gap /= 2) {
57         for (int i = gap; i < n; i++) {
58             int j = i - gap, x = a[i];
59             while (j >= 0 && a[j] > x) {
60                 a[j + gap] = a[j];
61                 j -= gap;
62             }
63             a[j + gap] = x;
64         }
65     }
66 }
67
68 void heapSort(int a[], int n) {
69     for (int i = (n - 1) / 2; i >= 0; i--) {
70         bubbleDown(a, i, n);
71     }
72     for (int i = n - 1; i > 0; i--) {
73         Swap(a[i], a[0]);
74         bubbleDown(a, 0, i);
75     }
76     for (int i = 0; i < n - 1 - i; i++) {
77         Swap(a[i], a[n - 1 - i]);
78     }
79 }
80
81 void mergeSort(int a[], int n) {
82     mergeSortRecursion(a, 0, n - 1);
83 }
84
85 void quickSort(int a[], int n) {
86     quickSortRecursion(a, 0, n - 1);
87 }
88
89 void countingSort(int a[], int n) {
90     int mx = a[0];
91     for (int i = 1; i < n; i++) {
92         if (a[i] > mx) mx = a[i];
93     }
94
95     int* b = new int[mx + 1];
96     for (int i = 0; i <= mx; i++) b[i] = 0;
97
98     for (int i = 0; i < n; i++) b[a[i]]++;
```

```
99
100     for (int i = 1; i <= mx; i++) b[i] += b[i - 1];
101
102     int* c = new int[n];
103     for (int i = 0; i < n; i++) {
104         c[b[a[i]] - 1] = a[i];
105         b[a[i]]--;
106     }
107
108     for (int i = 0; i < n; i++) a[i] = c[i];
109     delete[] b; delete[] c;
110 }
111
112 void radixSort(int a[], int n) {
113     Ref head; // head: Danh sach ban dau
114     Ref L[10], T[10]; // L[]: 10 lo nho; T[]: tail của lo nho
115
116     head = nullptr;
117     for (int i = 0; i < 10; i++) L[i] = T[i] = nullptr; // Khoi tao danh sach rong
118
119     arrayToList(head, a, n);
120
121     int k = getMaxDigitLength(a, n);
122
123     for (int i = 1; i <= k; i++) {
124         Ref p = head;
125         while (p) {
126             int j = getDigit(p->val, i);
127             if (!L[j]) {
128                 L[j] = T[j] = p;
129             }
130             else {
131                 T[j]->next = p; T[j] = p;
132             }
133             Ref q = p->next;
134             p->next = nullptr; p = q;
135         }
136
137         int j = 0;
138         while (j < 10 && !L[j]) j++; // j: vi tri dau tien trong L[] ma L[j] != nullptr
139         head = L[j];
140
141         Ref prev = T[j];
142         for (j = j + 1; j < 10; j++) {
143             if (!L[j]) continue;
144             prev->next = L[j]; prev = T[j];
145         }
146     }
```

```
146     for (int i = 0; i < 10; i++) L[i] = T[i] = nullptr;
147 }
148
149 Ref p = head;
150 for (int i = 0; i < n; i++) {
151     a[i] = p->val;
152     p = p->next;
153 }
154 }
155
156 void flashSort(int a[], int n) {
157     int m = 0.43 * n;
158     int* L = new int[m];
159     for (int i = 0; i < m; i++) L[i] = 0;
160
161     int min = a[0], max = a[0];
162     for (int i = 0; i < n; i++) {
163         if (a[i] > max) max = a[i];
164         if (a[i] < min) min = a[i];
165     }
166
167     for (int i = 0; i < n; i++) {
168         int j = (m - 1) * (a[i] - min) / (max - min);
169         L[j]++;
170     }
171     for (int i = 1; i < m; i++) L[i] += L[i - 1];
172     for (int i = 0; i < m; i++) L[i]--;
173
174     int i = 0;
175     while (i < n) {
176         int k = (m - 1) * (a[i] - min) / (max - min);
177         if (i > L[k]) {
178             i++; continue;
179         }
180
181         int val = a[i];
182         while (i <= L[k]) {
183             k = (m - 1) * (val - min) / (max - min);
184             int tmp = a[L[k]];
185             a[L[k]--] = val, val = tmp;
186         }
187     }
188
189     for (int k = 0; k < m - 1; k++) {
190         for (int i = L[k + 1] - 1; i >= L[k] + 1; i--) {
191             int x = a[i], j = i + 1;
192             while (j <= L[k + 1] && a[j] < x) {
193                 a[j - 1] = a[j];
194                 j++;
```

```
195         }
196         a[j - 1] = x;
197     }
198 }
199 }
200
201 void MSDRadixSort(int a[], int n) {
202     int d = getMaxDigitLength(a, n);
203     MSDRadixSortRecurcion(a, 0, n - 1, d);
204 }
205
206 void MSDRadixSortBinary(int a[], int n) {
207     int d = getMaxDigitLengthBinary(a, n);
208     MSDRadixSortBinaryRecurcion(a, 0, n - 1, d);
209 }
210
211 int interpolationSearch(int a[], int l, int r, int k) {
212     if (l > r || k < a[l] || k > a[r]) return -1;
213
214     int x = l + (k - a[l]) * (r - l) / (a[r] - a[l]);
215
216     if (a[x] == k) return x;
217     if (a[x] < k) return interpolationSearch(a, x + 1, r, k);
218     else return interpolationSearch(a, l, x - 1, k);
219 }
220
221 void mergeSortList(int a[], int n) {
222     Ref head = nullptr;
223     arrayToList(head, a, n);
224     head = mergeSortListRecurcion(head);
225
226     Ref p = head;
227     for (int i = 0; i < n; i++) {
228         a[i] = p->val;
229         p = p->next;
230     }
231 }
```

```
1 #include "AuxiliaryFunction.h"
2 #include "DataGenerator.h"
3
4 void bubbleDown(int a[], int k, int n) {
5     while (2 * k + 1 < n) {
6         int x = 2 * k + 1;
7         if (x + 1 < n && a[x + 1] < a[x]) x++;
8         if (a[k] <= a[x]) break;
9         Swap(a[k], a[x]);
10        k = x;
11    }
12 }
13
14 void mergeSortRecursion(int a[], int l, int r) {
15     if (l == r) return;
16     int m = (l + r) / 2;
17     mergeSortRecursion(a, l, m);
18     mergeSortRecursion(a, m + 1, r);
19     int* b = new int[r - l + 1];
20     int il = l, ir = m + 1, ib = 0;
21     while (il <= m && ir <= r) {
22         b[ib++] = a[il] < a[ir] ? a[il++] : a[ir++];
23     }
24     if (il <= m) {
25         while (ib < r - l + 1) b[ib++] = a[il++];
26     }
27     if (ir <= r) {
28         while (ib < r - l + 1) b[ib++] = a[ir++];
29     }
30     for (int i = 0; i < r - l + 1; i++) {
31         a[i + l] = b[i];
32     }
33     delete[] b;
34 }
35
36 int partitionQSort(int a[], int l, int r, int pivotId) {
37     int pivotVal = a[pivotId];
38     Swap(a[pivotId], a[r]);
39     int idx = l;
40     for (int i = l; i < r; i++) {
41         if (a[i] < pivotVal) {
42             Swap(a[i], a[idx]);
43             idx++;
44         }
45     }
46     Swap(a[r], a[idx]);
47     return idx;
48 }
49
```

```
50 void quickSortRecursion(int a[], int l, int r) {
51     if (l >= r) return;
52     int m = partitionQSort(a, l, r, (l + r) / 2);
53     quickSortRecursion(a, l, m - 1);
54     quickSortRecursion(a, m + 1, r);
55 }
56
57 Ref createNode(int k) {
58     Ref q = new Node;
59     q->val = k, q->next = nullptr;
60     return q;
61 }
62
63 void arrayToList(Ref& list, int a[], int n) {
64     for (int i = n - 1; i >= 0; i--) {
65         Ref q = createNode(a[i]);
66         if (!list) list = q;
67         else {
68             q->next = list, list = q;
69         }
70     }
71 }
72
73 int getMaxDigitLength(int a[], int n) {
74     int mx = a[0];
75     for (int i = 1; i < n; i++) {
76         if (a[i] > mx) mx = a[i];
77     }
78     int cnt = 0;
79     while (mx > 0) { mx /= 10; cnt++; }
80     return cnt;
81 }
82
83 int getDigit(int n, int k) {
84     while (k > 1) {
85         n /= 10;
86         k--;
87     }
88     return n % 10;
89 }
90
91 int findKth(int a[], int left, int right, int k) {
92     int pivotId = (right + left) / 2;
93     pivotId = partitionQSort(a, left, right, pivotId);
94     if (pivotId == k) return a[pivotId];
95     if (pivotId > k) return findKth(a, left, pivotId - 1, k);
96     else return findKth(a, pivotId + 1, right, k);
97 }
98
```

```
99 void MSDRadixSortRecurcion(int a[], int l, int r, int d) {
100     if (d == 0 || r <= l) return;
101     int* L[10]; int size[10] = { 0 };
102     for (int i = 0; i < 10; i++) L[i] = new int[r - l + 1];
103     for (int i = l; i <= r; i++) {
104         int j = getDigit(a[i], d);
105         L[j][size[j]++] = a[i];
106     }
107
108     int idx = l;
109     for (int i = 0; i < 10; i++) {
110         for (int j = 0; j < size[i]; j++) {
111             a[idx++] = L[i][j];
112         }
113     }
114
115     for (int i = 0; i < 10; i++) delete[] L[i];
116     for (int i = 1; i < 10; i++) size[i] += size[i - 1];
117
118     MSDRadixSortRecurcion(a, l, l + size[0] - 1, d - 1);
119     for (int i = 1; i < 10; i++) {
120         MSDRadixSortRecurcion(a, l + size[i - 1], l + size[i] - 1, d - 1);
121     }
122 }
123
124 int getMaxDigitLengthBinary(int a[], int n) {
125     int mx = a[0];
126     for (int i = 1; i < n; i++) {
127         if (a[i] > mx) mx = a[i];
128     }
129
130     int cnt = 0;
131     while (mx > 0) { mx /= 2; cnt++; }
132     return cnt;
133 }
134
135 int getDigitBinary(int n, int d) {
136     while (d > 1) {
137         n /= 2; d--;
138     }
139     return n % 2;
140 }
141
142 void MSDRadixSortBinaryRecursion(int a[], int l, int r, int d) {
143     if (d == 0 || r <= l) return;
144     int idx = l;
145     for (int i = l; i <= r; i++) {
146         if (getDigitBinary(a[i], d) == 0) {
147             Swap(a[i], a[idx]);
```



```
148         idx++;
149     }
150 }
151 MSDRadixSortBinaryRecursion(a, l, idx - 1, d - 1);
152 MSDRadixSortBinaryRecursion(a, idx, r, d - 1);
153 }
154
155 Ref findMid(Ref list) {
156     Ref slow = list, fast = list;
157     while (fast && fast->next && fast->next->next) {
158         slow = slow->next;
159         fast = fast->next->next;
160     }
161     return slow;
162 }
163
164 Ref mergeLinkList(Ref list1, Ref list2) {
165     if (!list1) return list2;
166     if (!list2) return list1;
167     if (list1->val < list2->val) {
168         list1->next = mergeLinkList(list1->next, list2);
169         return list1;
170     }
171     else {
172         list2->next = mergeLinkList(list2->next, list1);
173         return list2;
174     }
175 }
176
177 Ref mergeSortListRecursion(Ref list) {
178     if (!list || !list->next) return list;
179
180     Ref mid = findMid(list);
181     Ref list2 = mid->next;
182     mid->next = nullptr;
183
184     list = mergeSortListRecursion(list);
185     list2 = mergeSortListRecursion(list2);
186     return mergeLinkList(list, list2);
187 }
```