```cpp
#include "Sort.h"
#include "DataGenerator.h"
#include "AuxiliaryFunction.h"
#include <iostream>

void bubbleSort(int a[], int n) {
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (a[j] > a[j + 1]) Swap(a[j], a[j + 1]);
        }
    }
}

void insertionSort(int a[], int n) {
    for (int i = 1; i < n; i++) {
        int x = a[i], j = i - 1;
        while (j >= 0 && a[j] > x) {
            a[j + 1] = a[j];
            j--;
        }
        a[j + 1] = x;
    }
}

void selectionSort(int a[], int n) {
    for (int i = 0; i < n - 1; i++) {
        int minId = a[i];
        for (int j = i + 1; j < n; j++) {
            if (a[j] < a[minId]) minId = j;
        }
        Swap(a[i], a[minId]);
    }
}

void shakerSort(int a[], int n) {
    int l = 0, r = n - 1, k = 0;
    while (l < r) {
        for (int i = l; i < r; i++) {
            if (a[i] > a[i + 1]) {
                Swap(a[i], a[i + 1]);
                k = i;
            }
        }
        r = k;
        for (int i = r; i > l; i--) {
            if (a[i] < a[i - 1]) {
                Swap(a[i], a[i - 1]);
                k = i;
            }
        }
        l = k;
    }
}

void shellSort(int a[], int n) {
    for (int gap = n / 2; gap > 0; gap /= 2) {
        for (int i = gap; i < n; i++) {
            int j = i - gap, x = a[i];
            while (j >= 0 && a[j] > x) {
                a[j + gap] = a[j];
                j -= gap;
            }
            a[j + gap] = x;
        }
    }
}

void heapSort(int a[], int n) {
    for (int i = (n - 1) / 2; i >= 0; i--) {
        bubbleDown(a, i, n);
    }
    for (int i = n - 1; i > 0; i--) {
        Swap(a[i], a[0]);
        bubbleDown(a, 0, i);
    }
    for (int i = 0; i < n - 1 - i; i++) {
        Swap(a[i], a[n - 1 - i]);
    }
}

void mergeSort(int a[], int n) {
    mergeSortRecursion(a, 0, n - 1);
}

void quickSort(int a[], int n) {
    quickSortRecursion(a, 0, n - 1);
}

void countingSort(int a[], int n) {
    int mx = a[0];
    for (int i = 1; i < n; i++) {
        if (a[i] > mx) mx = a[i];
    }

    int* b = new int[mx + 1];
    for (int i = 0; i <= mx; i++) b[i] = 0;

    for (int i = 0; i < n; i++) b[a[i]]++;
```

```cpp

    for (int i = 1; i <= mx; i++) b[i] += b[i - 1];

    int* c = new int[n];
    for (int i = 0; i < n; i++) {
        c[b[a[i]] - 1] = a[i];
        b[a[i]]--;
    }

    for (int i = 0; i < n; i++) a[i] = c[i];
    delete[] b; delete[] c;
}

void radixSort(int a[], int n) {
    Ref head; // head: Danh sach ban dau
    Ref L[10], T[10];  // L[]: 10 lo nho; T[]: tail cua lo nho

    head = nullptr;
    for (int i = 0; i < 10; i++) L[i] = T[i] = nullptr; // Khoi tao danh sach rong

    arrayToList(head, a, n);

    int k = getMaxDigitLength(a, n);

    for (int i = 1; i <= k; i++) {
        Ref p = head;
        while (p) {
            int j = getDigit(p->val, i);
            if (!L[j]) {
                L[j] = T[j] = p;
            }
            else {
                T[j]->next = p; T[j] = p;
            }
            Ref q = p->next;
            p->next = nullptr; p = q;
        }

        int j = 0;
        while (j < 10 && !L[j]) j++; // j: vi tri dau tien trong L[] ma L[j] != nullptr
        head = L[j];

        Ref prev = T[j];
        for (j = j + 1; j < 10; j++) {
            if (!L[j]) continue;
            prev->next = L[j]; prev = T[j];
        }
        for (int i = 0; i < 10; i++) L[i] = T[i] = nullptr;
    }

    Ref p = head;
    for (int i = 0; i < n; i++) {
        a[i] = p->val;
        p = p->next;
    }
}

void flashSort(int a[], int n) {
    int m = 0.43 * n;
    int* L = new int[m];
    for (int i = 0; i < m; i++) L[i] = 0;

    int min = a[0], max = a[0];
    for (int i = 0; i < n; i++) {
        if (a[i] > max) max = a[i];
        if (a[i] < min) min = a[i];
    }

    for (int i = 0; i < n; i++) {
        int j = (m - 1) * (a[i] - min) / (max - min);
        L[j]++;
    }
    for (int i = 1; i < m; i++) L[i] += L[i - 1];
    for (int i = 0; i < m; i++) L[i]--;

    int i = 0;
    while (i < n) {
        int k = (m - 1) * (a[i] - min) / (max - min);
        if (i > L[k]) {
            i++; continue;
        }

        int val = a[i];
        while (i <= L[k]) {
            k = (m - 1) * (val - min) / (max - min);
            int tmp = a[L[k]];
            a[L[k]--] = val, val = tmp;
        }
    }

    for (int k = 0; k < m - 1; k++) {
        for (int i = L[k + 1] - 1; i >= L[k] + 1; i--) {
            int x = a[i], j = i + 1;
            while (j <= L[k + 1] && a[j] < x) {
                a[j - 1] = a[j];
                j++;
                }
                a[j - 1] = x;
            }
        }
}

void MSDRadixSort(int a[], int n) {
    int d = getMaxDigitLength(a, n);
    MSDRadixSortRecurcion(a, 0, n - 1, d);
}

void MSDRadixSortBinary(int a[], int n) {
    int d = getMaxDigitLengthBinary(a, n);
    MSDRadixSortBinaryRecursion(a, 0, n - 1, d);
}

int interpolationSearch(int a[], int l, int r, int k) {
    if (l > r || k < a[l] || k > a[r]) return -1;

    int x = l + (k - a[l]) * (r - l) / (a[r] - a[l]);

    if (a[x] == k) return x;
    if (a[x] < k) return interpolationSearch(a, x + 1, r, k);
    else return interpolationSearch(a, l, x - 1, k);
}

void mergeSortList(int a[], int n) {
    Ref head = nullptr;
    arrayToList(head, a, n);
    head = mergeSortListRecursion(head);

    Ref p = head;
    for (int i = 0; i < n; i++) {
        a[i] = p->val;
        p = p->next;
    }
}
```

```cpp
#include "AuxiliaryFunction.h"
#include "DataGenerator.h"

void bubbleDown(int a[], int k, int n) {
    while (2 * k + 1 < n) {
        int x = 2 * k + 1;
        if (x + 1 < n && a[x + 1] < a[x]) x++;
        if (a[k] <= a[x]) break;
        Swap(a[k], a[x]);
        k = x;
    }
}

void mergeSortRecursion(int a[], int l, int r) {
    if (l == r) return;
    int m = (l + r) / 2;
    mergeSortRecursion(a, l, m);
    mergeSortRecursion(a, m + 1, r);
    int* b = new int[r - l + 1];
    int il = l, ir = m + 1, ib = 0;
    while (il <= m && ir <= r) {
        b[ib++] = a[il] < a[ir] ? a[il++] : a[ir++];
    }
    if (il <= m) {
        while (ib < r - l + 1) b[ib++] = a[il++];
    }
    if (ir <= r) {
        while (ib < r - l + 1) b[ib++] = a[ir++];
    }
    for (int i = 0; i < r - l + 1; i++) {
        a[i + l] = b[i];
    }
    delete[] b;
}

int partitionQSort(int a[], int l, int r, int pivotId) {
    int pivotVal = a[pivotId];
    Swap(a[pivotId], a[r]);
    int idx = l;
    for (int i = l; i < r; i++) {
        if (a[i] < pivotVal) {
            Swap(a[i], a[idx]);
            idx++;
        }
    }
    Swap(a[r], a[idx]);
    return idx;
}

```

```cpp
void quickSortRecursion(int a[], int l, int r) {
    if (l >= r) return;
    int m = partitionQSort(a, l, r, (l + r) / 2);
    quickSortRecursion(a, l, m - 1);
    quickSortRecursion(a, m + 1, r);
}

Ref createNode(int k) {
    Ref q = new Node;
    q->val = k, q->next = nullptr;
    return q;
}

void arrayToList(Ref& list, int a[], int n) {
    for (int i = n - 1; i >= 0; i--) {
        Ref q = createNode(a[i]);
        if (!list) list = q;
        else {
            q->next = list, list = q;
        }
    }
}

int getMaxDigitLength(int a[], int n) {
    int mx = a[0];
    for (int i = 1; i < n; i++) {
        if (a[i] > mx) mx = a[i];
    }
    int cnt = 0;
    while (mx > 0) { mx /= 10; cnt++; }
    return cnt;
}

int getDigit(int n, int k) {
    while (k > 1) {
        n /= 10;
        k--;
    }
    return n % 10;
}

int findKth(int a[], int left, int right, int k) {
    int pivotId = (right + left) / 2;
    pivotId = partitionQSort(a, left, right, pivotId);
    if (pivotId == k) return a[pivotId];
    if (pivotId > k) return findKth(a, left, pivotId - 1, k);
    else return findKth(a, pivotId + 1, right, k);
}

```

```
void MSDRadixSortRecurcion(int a[], int l, int r, int d) {
    if (d == 0 || r <= l) return;
    int* L[10]; int size[10] = { 0 };
    for (int i = 0; i < 10; i++) L[i] = new int[r - l + 1];
    for (int i = l; i <= r; i++) {
        int j = getDigit(a[i], d);
        L[j][size[j]++] = a[i];
    }

    int idx = l;
    for (int i = 0; i < 10; i++) {
        for (int j = 0; j < size[i]; j++) {
            a[idx++] = L[i][j];
        }
    }

    for (int i = 0; i < 10; i++) delete[] L[i];
    for (int i = 1; i < 10; i++) size[i] += size[i - 1];

    MSDRadixSortRecurcion(a, l, l + size[0] - 1, d - 1);
    for (int i = 1; i < 10; i++) {
        MSDRadixSortRecurcion(a, l + size[i - 1], l + size[i] - 1, d - 1);
    }
}

int getMaxDigitLengthBinary(int a[], int n) {
    int mx = a[0];
    for (int i = 1; i < n; i++) {
        if (a[i] > mx) mx = a[i];
    }

    int cnt = 0;
    while (mx > 0) { mx /= 2; cnt++; }
    return cnt;
}

int getDigitBinary(int n, int d) {
    while (d > 1) {
        n /= 2; d--;
    }
    return n % 2;
}

void MSDRadixSortBinaryRecursion(int a[], int l, int r, int d) {
    if (d == 0 || r <= l) return;
    int idx = l;
    for (int i = l; i <= r; i++) {
        if (getDigitBinary(a[i], d) == 0) {
            Swap(a[i], a[idx]);
```

```
                idx++;
            }
        }
        MSDRadixSortBinaryRecursion(a, l, idx - 1, d - 1);
        MSDRadixSortBinaryRecursion(a, idx, r, d - 1);
}

Ref findMid(Ref list) {
    Ref slow = list, fast = list;
    while (fast && fast->next && fast->next->next) {
        slow = slow->next;
        fast = fast->next->next;
    }
    return slow;
}

Ref mergeLinkList(Ref list1, Ref list2) {
    if (!list1) return list2;
    if (!list2) return list1;
    if (list1->val < list2->val) {
        list1->next = mergeLinkList(list1->next, list2);
        return list1;
    }
    else {
        list2->next = mergeLinkList(list2->next, list1);
        return list2;
    }
}

Ref mergeSortListRecursion(Ref list) {
    if (!list || !list->next) return list;

    Ref mid = findMid(list);
    Ref list2 = mid->next;
    mid->next = nullptr;

    list = mergeSortListRecursion(list);
    list2 = mergeSortListRecursion(list2);
    return mergeLinkList(list, list2);
}
```