

Lab 1

Mục Lục

Selection Sort	3
1. Biên dịch đoạn chương trình nêu trên.....	3
2. Tại sao trong hàm SelectionSort, vòng lặp thứ nhất có điều kiện là $i < N-1$?	3
3. Trả lời các dòng lệnh có yêu cầu ghi chú.....	3
4. Sửa lại chương trình để nhập dãy số nguyên từ file input.txt có nội dung như sau:.....	4
5. Sửa hàm SelectionSort trên để sắp xếp dãy số nguyên ở câu 4 giảm dần.	5
Heap Sort.....	7
1. Trả lời các dòng lệnh có yêu cầu ghi chú.....	7
2. Cho biết chức năng của đoạn chương trình trên.	7
3. Viết hàm void CreateHeap(int a[], int N); để chuyển đổi dãy a_0, a_1, \dots, a_{N-1} thành heap.	7
4. Viết hàm void HeapSort(int a[], int N); để sắp xếp một dãy số nguyên tăng dần.....	8
5. Bổ sung các hàm trên vào chương trình mẫu (CacThuatToanSapXep) đồng thời thay đổi hàm main và file input để sắp xếp dãy số nguyên sau tăng dần:	8
6. Viết lại thuật toán Heap Sort để sắp xếp dãy số ở câu 3 giảm dần.	9
Quick Sort	12
1. Bổ sung các hàm trên vào chương trình mẫu (CacThuatToanSapXep) đồng thời thay đổi hàm main và file input để sắp xếp dãy số nguyên sau tăng dần:	12
2. Sửa lại chương trình để đếm số phép gán và số phép so sánh sử dụng trong hàm QuickSort.....	13
Merge Sort.....	16
1. Trả lời các dòng lệnh có yêu cầu ghi chú.....	16
2. Cho biết chức năng của từng hàm trên.....	17
Bổ sung các hàm cần thiết vào chương trình mẫu (CacThuatToanSapXep) và viết hàm void MergeSort(int a[], int N); để sắp xếp dãy số nguyên sau tăng dần.	18
4. Sửa lại chương trình để sắp xếp dãy số trên giảm dần.....	20
Áp dụng – Nâng cao.....	22
Bài Tập Thêm.....	30
1. Viết chương trình so sánh các thuật toán Selection Sort, Heap Sort, Quick Sort, Merge Sort.	30
2. Trong thuật toán QuickSort, nếu lấy x là phần tử đầu dãy, hãy viết chương trình và so sánh thời gian chạy thuật toán với khi lấy x là phần tử chính giữa dãy.....	35
3. Insertion Sort.....	37

4. Binary Insertion Sort.....	38
5. Interchange Sort.....	38
6. Bubble Sort.....	39
7. Shaker Sort.....	39
8. Shell Sort.....	40
9. Radix Sort.....	40

Selection Sort

Yêu cầu:

1. Biên dịch đoạn chương trình nêu trên.

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
@Glasspham on E:/Code/Github/DSA-GTVTHCM
# cd "E:/Code/Github/DSA-GTVTHCM/Lab_1" ; if ($?) { g++ Coding.cpp -o Coding } ; if ($?) { .\Coding }
1 2 4 5 6 8 12 15
@Glasspham on E:/Code/Github/DSA-GTVTHCM/Lab_1
#
```

2. Tại sao trong hàm SelectionSort, vòng lặp thứ nhất có điều kiện là $i < N-1$?

Mục đích của vòng lặp thứ nhất là chọn phần tử nhỏ nhất trong phần chưa sắp xếp của mảng và đưa nó về đúng vị trí. Khi đến phần tử cuối cùng ($i = N - 1$), nó **tự động là phần tử lớn nhất** vì tất cả các phần tử trước đó đã được đặt đúng chỗ.

3. Trả lời các dòng lệnh có yêu cầu ghi chú.

```
void SelectionSort(int a[], int N) {  
    // Ghi chú: tại sao không sử dụng kí hiệu & trong hàm này?  
    /* Trả lời: Mảng trong C++ được truyền theo cơ chế tham chiếu (pass by reference) mặc định, nên không  
       cần dùng kí hiệu &. Khi truyền mảng vào hàm, chỉ địa chỉ của phần tử đầu tiên được truyền vào. */  
    int min; // chỉ số phần tử nhỏ nhất trong dãy hiện hành  
    for (int i = 0; i < N - 1; i++) {  
        // Ghi chú: vòng lặp này dùng để làm gì?  
        /* Trả lời: Vòng lặp này duyệt qua từng phần tử của mảng, chọn phần tử nhỏ nhất trong phần còn lại  
           của mảng để đặt vào vị trí i. */  
        min = i;  
        for (int j = i + 1; j < N; j++) {  
            // Ghi chú: vòng lặp này dùng để làm gì?  
            /* Trả lời: Vòng lặp này dùng để tìm chỉ số của phần tử nhỏ nhất trong đoạn mảng chưa được  
               sắp xếp, bắt đầu từ vị trí i + 1 đến cuối mảng. Nó so sánh các phần tử để cập nhật giá  
               trị min nếu tìm thấy phần tử nhỏ hơn. */  
            if (a[j] < a[min]) {  
                min = j;  
            }  
            // Ghi chú: thao tác này dùng để làm gì?  
            /* Trả lời: Thao tác này cập nhật giá trị min thành chỉ số j nếu phần tử của chỉ số j nhỏ hơn
```

phần tử của chỉ số min hiện tại. Điều này giúp ghi nhớ vị trí của phần tử nhỏ nhất tìm thấy trong đoạn mảng đang xét./*

}

}

if (min != i) {

Swap(a[min], a[i]);

// Ghi chú: thao tác này dùng để làm gì?

/ Trả lời: Thao tác này hoán đổi phần tử nhỏ nhất tìm thấy (tại a[min]) với phần tử ở vị trí đầu của đoạn chưa sắp xếp (a[i]). Điều này đảm bảo phần tử nhỏ nhất được đặt vào đúng vị trí trong mảng sau mỗi lần lặp.*/*

}

}

}

4. Sửa lại chương trình để nhập dãy số nguyên từ file input.txt có nội dung như sau:

```
#include <iostream>
```

```
#include <fstream>
```

```
using namespace std;
```

```
const int MAXN = 1e5;
```

```
void Swap(int &a, int &b) { int c = a; a = b; b = c; }
```

```
void SelectionSort(int a[], int N) {
```

```
    int min;
```

```
    for (int i = 0; i < N - 1; i++) {
```

```
        min = i;
```

```
        for (int j = i + 1; j < N; j++)
```

```
            if (a[j] < a[min]) min = j;
```

```
        if (min != i) Swap(a[min], a[i]);
```

```
    }
```

```
}
```

```
int main() {
```

```
    ifstream ifile("input.txt");
```

```

if (!ifile) {
    cout << "Can not file input.txt" << endl;
    return 1;
}
int x[MAXN], n = 0;
while(ifile >> x[n]) ++n;
ifile.close();
SelectionSort(x, n);
ofstream ofile("output.txt");
for (int i = 0; i < n; i++) ofile << x[i] << ' ';
ofile.close();
}

```

The screenshot shows a C++ IDE with tabs for 'main.cpp', 'tmp.cpp', 'output.txt', and 'input.txt'. The 'output.txt' tab is active, showing the content '1 2 3 5 6 8 10 23'.

5. Sửa hàm SelectionSort trên để sắp xếp dãy số nguyên ở câu 4 giảm dần.

```

#include <iostream>
#include <fstream>
using namespace std;
const int MAXN = 1e5;
void Swap(int &a, int &b) { int c = a; a = b; b = c; }
void SelectionSort(int a[], int N) {
    int max;
    for (int i = 0; i < N - 1; i++) {
        max = i;
        for (int j = i + 1; j < N; j++)
            if (a[j] > a[max]) max = j;
        if (max != i) Swap(a[max], a[i]);
    }
}

```

```

}

int main() {
    ifstream ifile("input.txt");
    if (!ifile) {
        cout << "Can not file input.txt" << endl;
        return 1;
    }
    int x[MAXN], n = 0;
    while(ifile >> x[n]) ++n;
    ifile.close();
    SelectionSort(x, n);
    ofstream ofile("output.txt");
    for (int i = 0; i < n; i++) ofile << x[i] << ' ';
    ofile.close();
}

```

The screenshot shows a C++ IDE with four tabs: 'main.cpp', 'tmp.cpp', 'output.txt', and 'input.txt'. The 'output.txt' tab is active, displaying the output of the program. The output is a single line of text: '23 10 8 6 5 3 2 1'. The numbers are separated by spaces and are displayed in a monospaced font. The IDE interface is dark-themed.

Heap Sort

Yêu cầu:

1. Trả lời các dòng lệnh có yêu cầu ghi chú

```
void Shift(int a[], int left, int right) {  
    int x, curr, joint;  
    curr = left;  
    joint = 2 * curr + 1; // a : Phần tử liên đới joint  
    x = a[curr];  
    while (joint <= right) {  
        if (joint < right) {  
            // Ghi chú: điều kiện này có ý nghĩa gì?  
            /* Trả lời: Điều kiện `joint < right` đảm bảo rằng phần tử `joint + 1` (con phải) không  
            vượt quá giới hạn `right`. Nếu vượt quá `right`, tức là không có con phải, nên chỉ cần  
            xét con trái `a[joint]`. */  
            if (a[joint] < a[joint + 1]) joint = joint + 1;  
        }  
        if (a[joint] < x) break; // Thỏa quan hệ liên đới  
        a[curr] = a[joint];  
        curr = joint; // Xét khả năng hiệu chỉnh lan truyền  
        joint = 2 * curr + 1;  
    }  
    a[curr] = x;  
}
```

2. Cho biết chức năng của đoạn chương trình trên.

Hàm này giúp duy trì tính chất của **heap tối đa** bằng cách **hiệu chỉnh lan truyền** từ một nút left đến right trong cây nhị phân.

3. Viết hàm void CreateHeap(int a[], int N); để chuyển đổi dãy a0, a1, ..., aN-1 thành heap.

```
void CreateHeap(int a[], int N){  
    for (int i = N / 2 - 1; i >= 0; i--) Shift(a, i, N - 1);  
}
```

4. Viết hàm void HeapSort(int a[], int N); để sắp xếp một dãy số nguyên tăng dần.

```
void HeapSort(int a[], int N) {  
    CreateHeap(a, N);  
    for (int i = N - 1; i > 0; i--) {  
        Swap(a[0], a[i]);  
        Shift(a, 0, i - 1);  
    }  
}
```

5. Bổ sung các hàm trên vào chương trình mẫu (CacThuatToanSapXep) đồng thời thay đổi hàm main và file input để sắp xếp dãy số nguyên sau tăng dần:

```
#include <iostream>  
#include <fstream>  
using namespace std;  
const int MAXN = 1e5;  
void Swap(int &a, int &b) { int c = a; a = b; b = c; }  
void Shift(int a[], int left, int right) {  
    int x, curr, joint;  
    curr = left;  
    joint = 2 * curr + 1; // Con trái  
    x = a[curr];  
    while (joint <= right) {  
        if (joint < right && a[joint] < a[joint + 1]) joint = joint + 1;  
        if (a[joint] < x) break;  
        a[curr] = a[joint];  
        curr = joint;  
        joint = 2 * curr + 1;  
    }  
    a[curr] = x;  
}  
void CreateHeap(int a[], int N) {
```



```

    for (int left = (N - 1) / 2; left >= 0; left--) Shift(a, left, N - 1);
}

void HeapSort(int a[], int N) {
    CreateHeap(a, N);
    for (int i = N - 1; i > 0; i--) {
        Swap(a[0], a[i]);
        Shift(a, 0, i - 1);
    }
}

int main() {
    ifstream ifile("input.txt");
    if (!ifile) {
        cout << "Can not file input.txt" << endl;
        return 1;
    }
    int x[MAXN], n = 0;
    while(ifile >> x[n]) ++n;
    ifile.close();
    HeapSort(x, n);
    for (int i = 0; i < n; i++) cout << x[i] << ' ';
}

```

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Glasspham on E:/Code/Github/DSA-GTVTHCM
# cd "e:\Code\Github\DSA-GTVTHCM\Lab_1" ; if ($?) { g++ main.cpp -o main } ; if ($?) { .\main }
6 12 18 42 44 55 67 94
Glasspham on E:/Code/Github/DSA-GTVTHCM/Lab_1
#

```

6. Viết lại thuật toán Heap Sort để sắp xếp dãy số ở câu 3 giảm dần.

```

#include <iostream>

#include <fstream>

```

```

using namespace std;

const int MAXN = 1e5;

void Swap(int &a, int &b) { int c = a; a = b; b = c; }

void ShiftMin(int a[], int left, int right) {
    int x, curr, joint;
    curr = left;
    joint = 2 * curr + 1;
    x = a[curr];
    while (joint <= right) {
        if (joint < right && a[joint] > a[joint + 1]) joint = joint + 1;
        if (a[joint] > x) break;
        a[curr] = a[joint];
        curr = joint;
        joint = 2 * curr + 1;
    }
    a[curr] = x;
}

void CreateMinHeap(int a[], int N) {
    for (int left = (N - 1) / 2; left >= 0; left--) ShiftMin(a, left, N - 1);
}

void HeapSortDescending(int a[], int N) {
    CreateMinHeap(a, N);
    for (int i = N - 1; i > 0; i--) {
        Swap(a[0], a[i]);
        ShiftMin(a, 0, i - 1);
    }
}

int main() {
    ifstream ifile("input.txt");
    if (!ifile) {

```

```
    cout << "Can not file input.txt" << endl;

    return 1;
}

int x[MAXN], n = 0;

while(ifile >> x[n]) ++n;

ifile.close();

HeapSortDescending(x, n);

for (int i = 0; i < n; i++) cout << x[i] << ' ';

}
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

@Glasspham on E:/Code/Github/DSA-GTVTHCM

cd "e:\Code\Github\DSA-GTVTHCM\Lab_1\" ; if (\$?) { g++ main.cpp -o main } ; if (\$?) { .\main }
94 67 55 44 42 18 12 6

@Glasspham on E:/Code/Github/DSA-GTVTHCM/Lab_1

#

Quick Sort

Yêu cầu:

1. Bổ sung các hàm trên vào chương trình mẫu (CacThuatToanSapXep) đồng thời thay đổi hàm main và file input để sắp xếp dãy số nguyên sau tăng dần:

```
#include <iostream>

#include <fstream>

using namespace std;

const int MAXN = 1e5;

void QuickSort(int a[], int left, int right) {

    int i, j, x;

    if (left >= right) return;

    x = a[(left + right) / 2]; // chọn phần tử giữa làm giá trị mốc

    i = left;

    j = right;

    while (i < j) {

        while (a[i] < x) i++;

        while (a[j] > x) j--;

        if (i <= j) {

            Swap(a[i], a[j]);

            i++;

            j--;

        }

    }

    QuickSort(a, left, j);

    QuickSort(a, i, right);

}

int main() {

    ifstream ifile("input.txt");

    if (!ifile) {

        cout << "Can not file input.txt" << endl;
```

```

        return 1;
    }

    int x[MAXN], n = 0;
    while(ifile >> x[n]) ++n;
    ifile.close();
    QuickSort(x, 0, n - 1);
    for (int i = 0; i < n; i++) cout << x[i] << ' ';
}

```

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
• @Glasspham on E:/Code/Github/Dsa-GTVTHCM
# cd "E:\Code\Github\DSA-GTVTHCM\Lab_1" ; if ($?) { g++ main.cpp -o main } ; if ($?) { .\main }
11 23 36 42 58 65 74 87 94 99
• @Glasspham on E:/Code/Github/Dsa-GTVTHCM/Lab_1
#

```

2. Sửa lại chương trình để đếm số phép gán và số phép so sánh sử dụng trong hàm QuickSort.

```

#include <iostream>
#include <fstream>
using namespace std;
const int MAXN = 1e5;
int assignments = 0; // Đếm số phép gán
int comparisons = 0; // Đếm số phép so sánh
void Swap(int &a, int &b) {
    int temp = a;
    a = b;
    b = temp;
    assignments += 3; // 3 phép gán trong Swap
}
void QuickSort(int a[], int left, int right) {
    if (left >= right) return;
    int i, j, x;
    x = a[(left + right) / 2]; // Chọn phần tử mốc

```

```

    assignments++; // Gán giá trị cho x
    i = left;
    j = right;
    assignments += 2; // Gán i và j
    while (i < j) {
        while (a[i] < x) {
            i++;
            comparisons++; // So sánh a[i] < x
        }
        while (a[j] > x) {
            j--;
            comparisons++; // So sánh a[j] > x
        }
        if (i <= j) {
            Swap(a[i], a[j]);
            i++;
            j--;
            assignments += 2; // Gán mới i, j
        }
    }
    QuickSort(a, left, j);
    QuickSort(a, i, right);
}

int main() {
    ifstream ifile("input.txt");
    if (!ifile) {
        cout << "Can not file input.txt" << endl;
        return 1;
    }

    int x[MAXN], n = 0;

```

```
while(ifile >> x[n]) ++n;

ifile.close();

QuickSort(x, 0, n - 1);

for (int i = 0; i < n; i++) cout << x[i] << ' ';

cout << "\nSo phiep hoan doi: " << assignments;

cout << "\nSo phiep so sanh: " << comparisons << endl;

}
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
• @Glasspham on E:/Code/Github/Dsa-GTVTHCM
# cd "e:\Code\Github\DSA-GTVTHCM\Lab_1\" ; if ($?) { g++ main.cpp -o main } ; if ($?) { .\main }
11 23 36 42 58 65 74 87 94 99
So phiep hoan doi: 74
So phiep so sanh: 15
• @Glasspham on E:/Code/Github/Dsa-GTVTHCM/Lab_1
#
```

Merge Sort

Yêu cầu:

1. Trả lời các dòng lệnh có yêu cầu ghi chú.

```
int b[MAXN], c[MAXN], nb, nc; // Ghi chú: 2 mảng này dùng để làm gì?

/* Trả lời: Mảng b[] và c[] được sử dụng để lưu hai nửa của dãy a[] trong quá trình phân tách và trộn (merge) */

void Distribute(int a[], int N, int &nb, int &nc, int k) {

    int i, pa, pb, pc; // Ghi chú: các biến này có ý nghĩa gì?

    /* Trả lời:

        i: Biến đếm trong vòng lặp (kiểm soát số phần tử di chuyển vào b[] hoặc c[]).

        pa: Vị trí hiện tại trong mảng a[] (quét toàn bộ mảng a[]).

        pb: Vị trí hiện tại trong mảng b[] (điểm chèn phần tử tiếp theo vào b[]).

        pc: Vị trí hiện tại trong mảng c[] (điểm chèn phần tử tiếp theo vào c[]).

    */

    pa = pb = pc = 0;

    while (pa < N) {

        for (i = 0; (pa < N) && (i < k); i++, pa++, pb++) {

            // Ghi chú: vòng lặp này có ý nghĩa gì?

            /* Trả lời

                Sao chép k phần tử từ a[] vào b[], bắt đầu từ pa. Nếu còn phần tử trong a[] (pa < N) và chưa đạt k phần tử (i < k), tiếp tục chép vào b[]. Tăng pa và pb sau mỗi lần sao chép để tiếp tục duyệt qua a[] và cập nhật b[].

            */

            b[pb] = a[pa];

        }

        for (i = 0; (pa < N) && (i < k); i++, pa++, pc++) {

            // Ghi chú: vòng lặp này có ý nghĩa gì?

            /* Trả lời:

                Sao chép k phần tử tiếp theo từ a[] vào c[]. Hoạt động tương tự như vòng lặp trước nhưng chép vào c[] thay vì b[]. Giúp phân chia a[] thành hai nhóm con có kích thước tối đa k.

            */


```



```

        */

        c[pc] = a[pa];
    }
}

nb = pb;
nc = pc;
}

void Merge(int a[], int nb, int nc, int k) {
    int pa, pb, pc;
    pa = pb = pc = 0;
    while ((pb < nb) && (pc < nc)) MergeSubarr(a, nb, nc, pa, pb, pc, k);
    while (pb < nb) {
        a[pa++] = b[pb++]; // Ghi chú: câu lệnh này có ý nghĩa gì?
        /* Trả lời: Khi tất cả phần tử trong c[] đã được đưa vào a[], nhưng b[] vẫn còn phần tử chưa
            xét, câu lệnh này sao chép các phần tử còn lại từ b[] vào a[]. pb++ giúp tiếp tục duyệt
            phần tử tiếp theo trong b[]. */
    }
    while (pc < nc) {
        a[pa++] = c[pc++]; // Ghi chú: câu lệnh này có ý nghĩa gì?
        /* Trả lời: Khi tất cả phần tử trong b[] đã được đưa vào a[], nhưng c[] vẫn còn phần tử chưa
            xét, câu lệnh này sao chép các phần tử còn lại từ c[] vào a[]. pc++ giúp tiếp tục duyệt
            phần tử tiếp theo trong c[]. */
    }
}

```

2. Cho biết chức năng của từng hàm trên.

1. Hàm Distribute

Chức năng:

- Phân tách mảng a[] thành hai mảng b[] và c[] theo từng nhóm k phần tử.
- b[] nhận k phần tử đầu tiên, sau đó c[] nhận k phần tử tiếp theo, tiếp tục lặp lại cho đến hết mảng a[].

- nb và nc lần lượt lưu số lượng phần tử trong b[] và c[].

2. Hàm MergeSubarr

Chức năng: Trộn hai nhóm con (subarrays) từ b[] và c[] trở lại vào a[], nhưng chỉ trộn k phần tử từ mỗi nhóm một lần.

3. Hàm Merge

Chức năng: Kết hợp nhiều nhóm con k phần tử trong b[] và c[] trở lại thành mảng a[] đã được sắp xếp.

Bổ sung các hàm cần thiết vào chương trình mẫu (CacThuatToanSapXep) và viết hàm void MergeSort(int a[], int N); để sắp xếp dãy số nguyên sau tăng dần.

```
#include <iostream>

#include <fstream>

using namespace std;

const int MAXN = 1e5;

int b[MAXN], c[MAXN], nb, nc;

void Distribute(int a[], int N, int &nb, int &nc, int k) {

    int i, pa, pb, pc;

    pa = pb = pc = 0;

    while (pa < N) {

        for (i = 0; (pa < N) && (i < k); i++, pa++, pb++) b[pb] = a[pa];

        for (i = 0; (pa < N) && (i < k); i++, pa++, pc++) c[pc] = a[pa];

    }

    nb = pb;

    nc = pc;

}

void MergeSubarr(int a[], int nb, int nc, int &pa, int &pb, int &pc, int k) {

    int rb, rc;

    rb = min(nb, pb + k);

    rc = min(nc, pb + k);

    while ((pb < rb) && (pc < rc)) {

        if (b[pb] < c[pc]) a[pa++] = b[pb++];

        else a[pa++] = c[pc++];

    }

}
```

```

    }

    while (pb < rb) a[pa++] = b[pb++];

    while (pc < rc) a[pa++] = c[pc++];
}

void Merge(int a[], int nb, int nc, int k) {
    int pa, pb, pc;

    pa = pb = pc = 0;

    while ((pb < nb) && (pc < nc)) MergeSubarr(a, nb, nc, pa, pb, pc, k);

    while (pb < nb) a[pa++] = b[pb++];

    while (pc < nc) a[pa++] = c[pc++];
}

void MergeSort(int a[], int N) {
    int k = 1;

    while (k < N) {
        Distribute(a, N, nb, nc, k);

        Merge(a, nb, nc, k);

        k *= 2;
    }
}

int main() {
    int x[] = {5, 2, 9, 3, 7, 2, 4, 11};

    int n = sizeof(x) / sizeof(x[0]);

    MergeSort(x, n);

    for (int i = 0; i < n; i++) cout << x[i] << ' ';
}

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```

● @Glasspham on E:/Code/Github/DSA-GTVTHCM
# cd "e:\Code\Github\DSA-GTVTHCM\Lab_1\" ; if ($?) { g++ main.cpp -o main } ; if ($?) { .\main }
2 2 3 4 5 7 9 11
○ @Glasspham on E:/Code/Github/DSA-GTVTHCM/Lab_1
#

```

4. Sửa lại chương trình để sắp xếp dãy số trên giảm dần.

```
#include <iostream>
#include <fstream>
using namespace std;
const int MAXN = 1e5;
void Distribute(int a[], int N, int &nb, int &nc, int k) {
    int i, pa, pb, pc;
    pa = pb = pc = 0;
    while (pa < N) {
        for (i = 0; (pa < N) && (i < k); i++, pa++, pb++) b[pb] = a[pa];
        for (i = 0; (pa < N) && (i < k); i++, pa++, pc++) c[pc] = a[pa];
    }
    nb = pb;
    nc = pc;
}
void MergeSubarr(int a[], int nb, int nc, int &pa, int &pb, int &pc, int k) {
    int rb = min(nb, pb + k);
    int rc = min(nc, pc + k);
    while ((pb < rb) && (pc < rc)) {
        if (b[pb] > c[pc]) a[pa++] = b[pb++];
        else a[pa++] = c[pc++];
    }
    while (pb < rb) a[pa++] = b[pb++];
    while (pc < rc) a[pa++] = c[pc++];
}
void Merge(int a[], int nb, int nc, int k) {
    int pa, pb, pc;
    pa = pb = pc = 0;
    while ((pb < nb) && (pc < nc)) MergeSubarr(a, nb, nc, pa, pb, pc, k);
    while (pb < nb) a[pa++] = b[pb++];
```

```

    while (pc < nc) a[pa++] = c[pc++];
}

void MergeSort(int a[], int N) {
    int k = 1;
    while (k < N) {
        Distribute(a, N, nb, nc, k);
        Merge(a, nb, nc, k);
        k *= 2;
    }
}

int main() {
    int x[] = {5, 2, 9, 3, 7, 2, 4, 11};
    int n = sizeof(x) / sizeof(x[0]);
    MergeSort(x, n);
    for (int i = 0; i < n; i++) cout << x[i] << ' ';
}

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```

• @Glasspham on E:/Code/Github/DSA-GTVTHCM
# cd "e:\Code\Github\DSA-GTVTHCM\Lab_1\" ; if ($?) { g++ main.cpp -o main } ; if ($?) { .\main }
11 9 7 5 4 3 2 2
o @Glasspham on E:/Code/Github/DSA-GTVTHCM/Lab_1
#

```

Áp dụng – Nâng cao

```
#include <iostream>
#include <vector>
#include <string>
#include <cmath>
using namespace std;
void swap(int &a, int &b) {
    int tmp = a;
    a = b;
    b = tmp;
}
// Hàm in mảng
void PrintArray(int a[], int N) {
    for (int i = 0; i < N; i++)
        cout << a[i] << " ";
    cout << endl;
}
// 1. Sắp xếp tăng dần
void SortAscending(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]);
        }
    }
}
// 2. Tìm số lớn thứ 3
int FindThirdLargest(int a[], int N) {
```

```

SortAscending(a, N);

int max1 = a[N - 1], count = 1;
for (int i = N - 2; i >= 0; i--) {
    if (a[i] < max1) {
        count++;
        max1 = a[i];
        if (count == 3) return max1;
    }
}
return -1;
}

// 3. Đếm số lượng phần tử lớn nhất
int CountMaxOccurrences(int a[], int N) {
    SortAscending(a, N);
    int maxVal = a[N - 1], count = 0;
    for (int i = N - 1; i >= 0 && a[i] == maxVal; i--)
        count++;
    return count;
}

// 4. Sắp xếp theo trị tuyệt đối tăng dần
void SortByAbsoluteValue(int a[], int N) {
    for (int i = 0; i < N - 1; i++) {
        int minIdx = i;
        for (int j = i + 1; j < N; j++)
            if (abs(a[j]) < abs(a[minIdx]))
                minIdx = j;
        swap(a[i], a[minIdx]);
    }
}

```

```
}
```

```
// 5. Sắp xếp số dương giảm dần, số âm tăng dần
```

```
void SortPositiveNegative(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]);
```

```
    }
```

```
    int index = -1;
```

```
    for (int i = 0; i < N; i++) {
```

```
        if (a[i] < 0) {
```

```
            index = i;
```

```
            break;
```

```
        }
```

```
    }
```

```
    if (index != -1) {
```

```
        for (int i = index; i < N - 1; i++) {
```

```
            for (int j = index; j < N - (i - index) - 1; j++)
```

```
                if (a[j] > a[j + 1])
```

```
                    swap(a[j], a[j + 1]);
```

```
            }
```

```
        }
```

```
    }
```

```
// 6. Chẵn tăng dần, lẻ giảm dần
```

```
void SortEvenOdd(int a[], int N) {
```

```
    int j = 0;
```

```
    for (int i = 0; i < N; i++) {
```

```
        if (a[i] % 2 == 0) {
```



```

        swap(a[i], a[j]);

        j++;
    }
}

SortAscending(a, j);

for (int i = j; i < N - 1; i++) {
    for (int k = j; k < N - (i - j) - 1; k++)
        if (a[k] < a[k + 1])
            swap(a[k], a[k + 1]);
    }
}

```

// 7. Sắp xếp theo vị trí chẵn/lẻ ban đầu

```

void SortEvenOddStable(int a[], int N) {
    for (int i = 0; i < N - 1; i++) {
        for (int j = i + 1; j < N; j++)
            if ((a[i] % 2 == 0 && a[j] % 2 == 0 && a[i] > a[j]) ||
                (a[i] % 2 != 0 && a[j] % 2 != 0 && a[i] < a[j]))
                swap(a[i], a[j]);
    }
}

```

// 8. Tạo một cấu trúc dữ liệu để xử lý danh sách trên.

```

class Student {
private:
    int id;

    string name;

    int birthYear;
public:
    // Constructor

```

```

Student(int id, string name, int birthYear) : id(id), name(name), birthYear(birthYear) {}

void printStudent() const {
    cout << id << "t" << name << "t\t" << birthYear << endl;
}

// Hàm hiển thị danh sách sinh viên
static void printStudents(const Student students[], int n) {
    cout << "MSSV\tHo va ten\t\tNam sinh\n";
    for (int i = 0; i < n; ++i)
        students[i].printStudent();
    cout << "-----\n";
}

// 9. Hàm sắp xếp danh sách theo MSSV tăng dần
static void sortByID(Student students[], int n) {
    for (int i = 0; i < n - 1; i++)
        for (int j = 0; j < n - i - 1; j++)
            if (students[j].id > students[j + 1].id)
                swap(students[j], students[j + 1]);
}

// 10. Hàm sắp xếp danh sách theo tên (thứ tự bảng chữ cái), nếu trùng thì theo năm sinh tăng dần
static void sortByNameAndYear(Student students[], int n) {
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            string lastWord1 = getLastWord(students[j].name);
            string lastWord2 = getLastWord(students[j + 1].name);
            if (lastWord1 > lastWord2 ||
                (lastWord1 == lastWord2 && students[j].birthYear > students[j + 1].birthYear)) {
                swap(students[j], students[j + 1]);
            }
        }
    }
}

```

```

    }

}

static void swap(Student& a, Student& b) {
    Student temp = a;
    a = b;
    b = temp;
}

static string getLastWord(const string& str) {
    istringstream iss(str);
    string word, lastWord;
    while (iss >> word) lastWord = word;
    return lastWord;
}

int getID() const { return id; }
};

int main() {
    int A[] = {12, 2, 15, -3, 8, 5, 1, -8, 6, 0, 4, 15};
    int N = sizeof(A) / sizeof(A[0]);

    // 1. Sắp xếp tăng dần
    SortAscending(A, N);
    cout << "1. Tang dan: ";
    PrintArray(A, N);

    // 2. Số lớn thứ 3
    int thirdLargest = FindThirdLargest(A, N);
    cout << "2. So lon thu 3: " << thirdLargest << endl;

    // 3. Số lượng phần tử lớn nhất
    int maxCount = CountMaxOccurrences(A, N);

```

```

cout << "3. So lan xuat hien so lon nhat: " << maxCount << endl;

// 4. Sắp xếp theo giá trị tuyệt đối tăng dần
int B[] = {12, 2, 15, -3, 8, 5, 1, -8, 6, 0, 4, 15};

SortByAbsoluteValue(B, N);

cout << "4. Sap xep theo tri tuyet doi: ";

PrintArray(B, N);

// 5. Sắp xếp số dương giảm dần, số âm tăng dần
int C[] = {12, 2, 15, -3, 8, 5, 1, -8, 6, 0, 4, 15};

SortPositiveNegative(C, N);

cout << "5. So duong giam dan, so am tang dan: ";

PrintArray(C, N);

// 6. Sắp xếp chẵn tăng dần, lẻ giảm dần
int D[] = {12, 2, 15, -3, 8, 5, 1, -8, 6, 0, 4, 15};

SortEvenOdd(D, N);

cout << "6. Chan tang dan, le giam dan: ";

PrintArray(D, N);

// 7. Sắp xếp chẵn tăng dần, lẻ giảm dần nhưng giữ vị trí ban đầu
int E[] = {12, 2, 15, -3, 8, 5, 1, -8, 6, 0, 4, 15};

SortEvenOddStable(E, N);

cout << "7. Giu vi tri chan/le, chan tang dan, le giam dan: ";

PrintArray(E, N);

// 8. Tạo một cấu trúc dữ liệu để xử lý danh sách trên.
Student students[] = {

    Student(1005, "Tran Minh Thanh", 1991),

    Student(1001, "Tran Thi Bich", 1988),

    Student(1003, "Tran Minh Thanh", 1990),

    Student(1000, "Vo Quang Vinh", 1990),

    Student(1008, "Nguyen Van An", 1990)

```

```

};

int n = sizeof(students) / sizeof(students[0]);

cout << "Danh sach ban dau:\n";

Student::printStudents(students, n);

// 9. Sắp xếp theo MSSV tăng dần

Student::sortByID(students, n);

cout << "9. Danh sach sap xep theo MSSV tang dan:\n";

Student::printStudents(students, n);

// 10. Sắp xếp theo tên, nếu trùng tên thì theo năm sinh tăng dần

Student::sortByNameAndYear(students, n);

cout << "10. Danh sach sap xep theo ten va nam sinh:\n";

Student::printStudents(students, n);

return 0;
}

```

```

• Glasspham's DSA-GTVTHCM/
> cd "e:\Code\Github\DSA-GTVTHCM\Lab_1\" ; if ($?) { g++ AP-NC-1.cpp -o AP-NC-1 } ; if ($?) { .\AP-NC-1 }
1. Tang dan: -8 -3 0 1 2 4 5 6 8 12 15 15
2. So lon thu 3: 8
3. So lan xuat hien so lon nhut: 2
4. Sap xep theo tri tuyet doi: 0 1 2 -3 4 5 6 -8 8 12 15 15
5. So duong giam dan, so am tang dan: 15 15 12 8 6 5 4 2 1 0 -8 -3
6. Chan tang dan, le giam dan: -8 0 2 4 6 8 12 15 15 5 1 -3
7. Giu vi tri chan/le, chan tang dan, le giam dan: -8 0 15 15 2 5 1 4 6 8 12 -3
Danh sach ban dau:
MSSV   Ho va ten           Nam sinh
1005   Tran Minh Thanh     1991
1001   Tran Thi Bich       1988
1003   Tran Minh Thanh     1990
1000   Vo Quang Vinh       1990
1008   Nguyen Van An       1990

-----
9. Danh sach sap xep theo MSSV tang dan:
MSSV   Ho va ten           Nam sinh
1000   Vo Quang Vinh       1990
1001   Tran Thi Bich       1988
1003   Tran Minh Thanh     1990
1005   Tran Minh Thanh     1991
1008   Nguyen Van An       1990

-----
10. Danh sach sap xep theo ten va nam sinh:
MSSV   Ho va ten           Nam sinh
1008   Nguyen Van An       1990
1001   Tran Thi Bich       1988
1003   Tran Minh Thanh     1990
1005   Tran Minh Thanh     1991
1000   Vo Quang Vinh       1990

```

Bài Tập Thêm

1. Viết chương trình so sánh các thuật toán Selection Sort, Heap Sort, Quick Sort, Merge Sort.

```
#include <iostream>
#include <vector>
#include <chrono>
#include <cstdlib>
#include <algorithm>
using namespace std;
using namespace chrono;
struct SortStats {
    long long comparisons = 0;
    long long assignments = 0;
};
// Selection Sort
void selectionSort(vector<int>& arr, SortStats& stats) {
    int n = arr.size();
    for (int i = 0; i < n - 1; i++) {
        int minIdx = i;
        for (int j = i + 1; j < n; j++) {
            stats.comparisons++; // So sánh
            if (arr[j] < arr[minIdx])
                minIdx = j;
        }
        swap(arr[i], arr[minIdx]);
        stats.assignments += 3; // Swap có 3 phép gán
    }
}
// Heap Sort
void heapify(vector<int>& arr, int n, int i, SortStats& stats) {
    int largest = i, left = 2 * i + 1, right = 2 * i + 2;
```

```

    if (left < n) {
        stats.comparisons++;
        if (arr[left] > arr[largest]) largest = left;
    }
    if (right < n) {
        stats.comparisons++;
        if (arr[right] > arr[largest]) largest = right;
    }
    if (largest != i) {
        swap(arr[i], arr[largest]);
        stats.assignments += 3;
        heapify(arr, n, largest, stats);
    }
}

void heapSort(vector<int>& arr, SortStats& stats) {
    int n = arr.size();
    for (int i = n / 2 - 1; i >= 0; i--) heapify(arr, n, i, stats);
    for (int i = n - 1; i > 0; i--) {
        swap(arr[0], arr[i]);
        stats.assignments += 3;
        heapify(arr, i, 0, stats);
    }
}

// Quick Sort
int partition(vector<int>& arr, int low, int high, SortStats& stats) {
    int pivot = arr[high], i = low - 1;
    for (int j = low; j < high; j++) {
        stats.comparisons++;
        if (arr[j] < pivot) {
            i++;

```

```

        swap(arr[i], arr[j]);

        stats.assignments += 3;
    }
}

swap(arr[i + 1], arr[high]);
stats.assignments += 3;
return i + 1;
}

void quickSort(vector<int>& arr, int low, int high, SortStats& stats) {
    if (low < high) {
        int pi = partition(arr, low, high, stats);
        quickSort(arr, low, pi - 1, stats);
        quickSort(arr, pi + 1, high, stats);
    }
}

// Merge Sort
void merge(vector<int>& arr, int left, int mid, int right, SortStats& stats) {
    int n1 = mid - left + 1, n2 = right - mid;
    vector<int> L(n1), R(n2);
    for (int i = 0; i < n1; i++) {
        L[i] = arr[left + i];
        stats.assignments++;
    }
    for (int i = 0; i < n2; i++) {
        R[i] = arr[mid + 1 + i];
        stats.assignments++;
    }
    int i = 0, j = 0, k = left;
    while (i < n1 && j < n2) {
        stats.comparisons++;

```



```

        if (L[i] <= R[j]) arr[k++] = L[i++];

        else arr[k++] = R[j++];

        stats.assignments++;
    }

    while (i < n1) {
        arr[k++] = L[i++];

        stats.assignments++;
    }

    while (j < n2) {
        arr[k++] = R[j++];

        stats.assignments++;
    }
}

void mergeSort(vector<int>& arr, int left, int right, SortStats& stats) {
    if (left < right) {
        int mid = left + (right - left) / 2;

        mergeSort(arr, left, mid, stats);

        mergeSort(arr, mid + 1, right, stats);

        merge(arr, left, mid, right, stats);
    }
}

int main() {
    const int SIZE = 10000; // Kích thước mảng
    vector<int> originalArr(SIZE);

    for (int& num : originalArr)
        num = rand() % 100000; // Sinh số ngẫu nhiên từ 0 đến 99999

    cout << "So sanh cac thuat toan sap xep voi " << SIZE << " phan tu:\n\n";

    vector<int> selectionArr = originalArr;

    SortStats selectionStats;

    auto start = high_resolution_clock::now();

```

```

selectionSort(selectionArr, selectionStats);

auto stop = high_resolution_clock::now();

double selectionDuration = duration_cast<microseconds>(stop - start).count() / 1000.0;

cout << "Selection Sort:\n";

cout << " - Thời gian: " << selectionDuration << " ms\n";

cout << " - Số phép so sánh: " << selectionStats.comparisons << "\n";

cout << " - Số phép gán: " << selectionStats.assignments << "\n\n";

vector<int> heapArr = originalArr;

SortStats heapStats;

start = high_resolution_clock::now();

heapSort(heapArr, heapStats);

stop = high_resolution_clock::now();

double heapDuration = duration_cast<microseconds>(stop - start).count() / 1000.0;

cout << "Heap Sort:\n";

cout << " - Thời gian: " << heapDuration << " ms\n";

cout << " - Số phép so sánh: " << heapStats.comparisons << "\n";

cout << " - Số phép gán: " << heapStats.assignments << "\n\n";

vector<int> quickArr = originalArr;

SortStats quickStats;

start = high_resolution_clock::now();

quickSort(quickArr, 0, quickArr.size() - 1, quickStats);

stop = high_resolution_clock::now();

double quickDuration = duration_cast<microseconds>(stop - start).count() / 1000.0;

cout << "Quick Sort:\n";

cout << " - Thời gian: " << quickDuration << " ms\n";

cout << " - Số phép so sánh: " << quickStats.comparisons << "\n";

cout << " - Số phép gán: " << quickStats.assignments << "\n\n";

vector<int> mergeArr = originalArr;

SortStats mergeStats;

```

```

start = high_resolution_clock::now();

mergeSort(mergeArr, 0, mergeArr.size() - 1, mergeStats);

stop = high_resolution_clock::now();

double mergeDuration = duration_cast<microseconds>(stop - start).count() / 1000.0;

cout << "Merge Sort:\n";

cout << " - Thời gian: " << mergeDuration << " ms\n";

cout << " - Số phép so sánh: " << mergeStats.comparisons << "\n";

cout << " - Số phép gán: " << mergeStats.assignments << "\n\n";

return 0;
}

```

2. Trong thuật toán QuickSort, nếu lấy x là phần tử đầu dãy, hãy viết chương trình và so sánh thời gian chạy thuật toán với khi lấy x là phần tử chính giữa dãy.

```

#include <iostream>
#include <vector>
#include <chrono>
#include <cstdlib>
#include <algorithm>
using namespace std;
using namespace chrono;

struct SortStats {
    long long comparisons = 0;
    long long assignments = 0;
};

// QuickSort - Pivot là phần tử đầu dãy
int partitionFirst(vector<int>& arr, int low, int high, SortStats& stats) {
    int pivot = arr[low];
    int i = low + 1, j = high;

    while (i <= j) {
        while (i <= j && arr[i] <= pivot) { i++; stats.comparisons++; }
        while (i <= j && arr[j] > pivot) { j--; stats.comparisons++; }
    }
}

```

```

        if (i < j) {
            swap(arr[i], arr[j]);
            stats.assignments += 3;
        }
    }
    swap(arr[low], arr[j]); // Đưa pivot về đúng vị trí
    stats.assignments += 3;
    return j;
}

void quickSortFirst(vector<int>& arr, int low, int high, SortStats& stats) {
    if (low < high) {
        int pivotIndex = partitionFirst(arr, low, high, stats);
        quickSortFirst(arr, low, pivotIndex - 1, stats);
        quickSortFirst(arr, pivotIndex + 1, high, stats);
    }
}

// QuickSort - Pivot là phần tử giữa dãy
int partitionMiddle(vector<int>& arr, int low, int high, SortStats& stats) {
    int mid = (low + high) / 2;
    swap(arr[low], arr[mid]); // Đưa phần tử giữa về đầu
    stats.assignments += 3;
    return partitionFirst(arr, low, high, stats); // Sử dụng lại partitionFirst
}

void quickSortMiddle(vector<int>& arr, int low, int high, SortStats& stats) {
    if (low < high) {
        int pivotIndex = partitionMiddle(arr, low, high, stats);
        quickSortMiddle(arr, low, pivotIndex - 1, stats);
        quickSortMiddle(arr, pivotIndex + 1, high, stats);
    }
}

```

// Hàm chạy benchmark và đo thời gian

```
void benchmarkQuickSort(vector<int> arr, bool useMiddlePivot) {
    SortStats stats;

    auto start = high_resolution_clock::now();

    if (useMiddlePivot) quickSortMiddle(arr, 0, arr.size() - 1, stats);
    else quickSortFirst(arr, 0, arr.size() - 1, stats);

    auto end = high_resolution_clock::now();

    auto duration = duration_cast<milliseconds>(end - start);

    cout << (useMiddlePivot ? "Pivot giữa:" : "Pivot đầu:") << "\n - Thời gian: "
        << duration.count() << "ms\n - Số phép so sánh: " << stats.comparisons
        << "\n - Số phép gán: " << stats.assignments << "\n";
}

int main() {
    int N = 100000;

    vector<int> arr(N);

    for (int i = 0; i < N; i++) arr[i] = rand() % 1000000;

    cout << "So sánh QuickSort với các cách chọn pivot:\n\n";

    benchmarkQuickSort(arr, false); // Pivot là phần tử đầu

    cout << endl;

    benchmarkQuickSort(arr, true); // Pivot là phần tử giữa

    return 0;
}
```

3. Insertion Sort.

```
void insertionSort(vector<int>& arr) {

    int n = arr.size();

    for (int i = 1; i < n; i++) {

        int key = arr[i];

        int j = i - 1;

        while (j >= 0 && arr[j] > key) {

            arr[j + 1] = arr[j];
```

```

        j--;
    }

    arr[j + 1] = key;
}
}

```

4. Binary Insertion Sort.

```

int binarySearch(vector<int>& arr, int item, int low, int high) {
    while (low <= high) {
        int mid = (low + high) / 2;
        if (arr[mid] > item) high = mid - 1;
        else low = mid + 1;
    }
    return low;
}

void binaryInsertionSort(vector<int>& arr) {
    int n = arr.size();
    for (int i = 1; i < n; i++) {
        int key = arr[i];
        int pos = binarySearch(arr, key, 0, i - 1);
        for (int j = i; j > pos; j--) arr[j] = arr[j - 1];
        arr[pos] = key;
    }
}

```

5. Interchange Sort.

```

void interchangeSort(vector<int>& arr) {
    int n = arr.size();
    for (int i = 0; i < n - 1; i++) {
        for (int j = i + 1; j < n; j++) {
            if (arr[i] > arr[j])
                swap(arr[i], arr[j]);
        }
    }
}

```

```
    }  
}  
}
```

6. Bubble Sort.

```
void bubbleSort(vector<int>& arr) {  
    int n = arr.size();  
    for (int i = 0; i < n - 1; i++) {  
        bool swapped = false;  
        for (int j = 0; j < n - i - 1; j++) {  
            if (arr[j] > arr[j + 1]) {  
                swap(arr[j], arr[j + 1]);  
                swapped = true;  
            }  
        }  
        if (!swapped) break;  
    }  
}
```

7. Shaker Sort.

```
void shakerSort(vector<int>& arr) {  
    int left = 0, right = arr.size() - 1;  
    while (left < right) {  
        for (int i = left; i < right; i++) {  
            if (arr[i] > arr[i + 1])  
                swap(arr[i], arr[i + 1]);  
        }  
        right--;  
        for (int i = right; i > left; i--) {  
            if (arr[i] < arr[i - 1])  
                swap(arr[i], arr[i - 1]);  
        }  
    }  
}
```

```
    left++;  
}  
}
```

8. Shell Sort.

```
void shellSort(vector<int>& arr) {  
    int n = arr.size();  
    for (int gap = n / 2; gap > 0; gap /= 2) {  
        for (int i = gap; i < n; i++) {  
            int key = arr[i], j = i;  
            while (j >= gap && arr[j - gap] > key) {  
                arr[j] = arr[j - gap];  
                j -= gap;  
            }  
            arr[j] = key;  
        }  
    }  
}
```

9. Radix Sort

```
void countingSort(vector<int>& arr, int exp) {  
    int n = arr.size();  
    vector<int> output(n);  
    int count[10] = {0};  
    for (int i = 0; i < n; i++)  
        count[(arr[i] / exp) % 10]++;  
    for (int i = 1; i < 10; i++)  
        count[i] += count[i - 1];  
    for (int i = n - 1; i >= 0; i--) {  
        output[count[(arr[i] / exp) % 10] - 1] = arr[i];  
        count[(arr[i] / exp) % 10]--;  
    }  
}
```



```
    for (int i = 0; i < n; i++)  
        arr[i] = output[i];  
}  
void radixSort(vector<int>& arr) {  
    int maxVal = *max_element(arr.begin(), arr.end());  
    for (int exp = 1; maxVal / exp > 0; exp *= 10)  
        countingSort(arr, exp);  
}
```