# Selection Sort

## 1. Biên dịch đoạn chương trình nêu trên.

#include <iostream>

using namespace std;

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() {

    int x[10] = {12, 2, 8, 5, 1, 6, 4, 15};

    int n = 8;

    SelectionSort(x, n);

    for (int i = 0; i < n; i++) {

        cout << x[i] << " ";

    }

    cout << endl;

    return 0; //

}

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

- Vì chỉ cần sắp xếp N-1 phần tử thì dãy đã được sắp xếp .

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

#include <stdio.h>

void Swap(int &a, int &b) {

    int c = a;

    a = b;

    b = c;

}

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

    //Ghi chú: tại sao không sử dụng kí hiệu & trong hàm này? Vì hàm này không cần thay đổi giá trị của mảng a mà chỉ cần sắp xếp lại mảng a

    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? để chọn ra phần tử nhỏ nhất trong dãy từ i đến N-1

        min = i;

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

            //Ghi chu: vòng lặp này dùng để làm gì? để tìm ra phần tử nhỏ nhất trong dãy từ i đến N-1

            if (a[j] < a[min]) {

                min = j;

                //Ghi chú : thao tác này dùng để làm gì? để cập nhật lại chỉ số của phần tử nhỏ nhất trong dãy từ i đến N-1

            }

        }

        if (min != i) {

            Swap(a[min], a[i]);

            //Ghi chu: thao tác này dùng để làm gì? để đổi chỗ 2 phần tử a[min] và a[i]

        }

    }

}

void main() {

    int x[10] = {12, 2, 8, 5, 1, 6, 4, 15}; // khởi tạo các giá trị trong mảng

    int n = 8; // số phần tử của mảng

    SelectionSort(x, n);

    for (int i = 0; i < n; i++) {

        cout << x[i];

    }

}

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

**5 1 2 3 8 6 23 10**

**Sau đó dùng thuật toán Selection Sort sắp xếp dãy số nguyên trên tăng dần và ghi kết quả vào file output.txt**

#include <iostream>

#include <fstream>

using namespace std;

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() {

    int x[10];

    int n = 0;

    ifstream fin("input.txt");

    ofstream fout("output.txt");

    while (fin >> x[n]) {

        n++;

    }

    SelectionSort(x, n);

        for (int i = 0; i < n; i++) {

        fout << x[i] << " ";

    }

    fin.close();

    fout.close();

    return 0;

}

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

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]);

        }

    }

}

# Heap Sort

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

void Shift(int a[], int left, int right)

{

    int x;

    int curr;

    int 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? Nghĩa là joint < right để xác định joint có phải là phần tử cuối cùng không .

        {

            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.

- Chương trình trên có chức năng sắp xếp mảng a theo thứ tự giảm dần.

## **3. Viết hàm void CreateHeap(int a[], int N); để chuyển đổi dãy a0, a1, …, aN-1 thành** **heap.  Gợi ý: Sử dụng hàm Shift bên trên với left hiện hành là phần tử ở giữa dãy ((N-1)/2). Lặp lại quá trình trên với left giảm dần về đầu dãy.**

void CreateHeap(int a[], int N)

{

    int left;

    left = (N - 1) / 2;

    while (left >= 0)

    {

        Shift(a, left, N - 1);

        left = left - 1;

    }

}

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

**Gợi ý:**

**Giai đoạn 1: Hiệu chỉnh dãy ban đầu thành heap**

**Giai đoạn 2: Sắp xếp dãy số dựa trên heap.**

**• Xét dãy hiện hành là dãy nhập**

**• Hoán vị phần tử lớn nhất (a0) về vị trí cuối.**

**• Xét dãy hiện hành loại đã trừ phần tử cuối.**

**• Hiệu chỉnh lại dãy hiện hành thành heap**

**• Lặp lại quá trình trên tới hết dãy ban đầu.**

void HeapSort(int a[], int N)

{

    int right;

    CreateHeap(a, N);

    right = N - 1;

    while (right > 0)

    {

        Swap(a[0], a[right]);

        right = right - 1;

        Shift(a, 0, right);

    }

}

## 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: 44 55 12 42 94 18 6 67

#include <iostream>

#include <fstream>

using namespace std;

void Swap(int &a, int &b) {

    int c = a;

    a = b;

    b = c;

}

void Shift(int a[], int left, int right)

{

    int x;

    int curr;

    int joint;

    curr = left;

    joint = 2 \* curr + 1;

    x = a[curr];

    while (joint <= right)

    {

        if (joint < right)

        {

            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;

}

void CreateHeap(int a[], int N)

{

    int left;

    left = N / 2 - 1;

    while (left >= 0)

    {

        Shift(a, left, N - 1);

        left = left - 1;

    }

}

void HeapSort(int a[], int N)

{

    int right;

    CreateHeap(a, N);

    right = N - 1;

    while (right > 0)

    {

        Swap(a[0], a[right]);

        right = right - 1;

        Shift(a, 0, right);

    }

}

int main()

{

    int x[10];

    int n = 0;

    ifstream fin("input.txt");

    ofstream fout("output.txt");

    while (fin >> x[n])

    {

        n++;

    }

    HeapSort(x, n);

    for (int i = 0; i < n; i++)

    {

        fout << x[i] << " ";

    }

    fin.close();

    fout.close();

    return 0;

}

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

void HeapSort(int a[], int N)

{

    int right;

    CreateHeap(a, N);

    right = N - 1;

    while (right > 0)

    {

        Swap(a[0], a[right]);

        right = right - 1;

        Shift(a, 0, right);

    }

    for (int i = 0; i < N / 2; i++)

    {

        Swap(a[i], a[N - i - 1]);

    }

}

# Quick Sort

## 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: 42 23 74 11 65 58 94 36 99 87

#include <iostream>

#include <fstream>

using namespace std;

void Swap(int &a, int &b) {

    int c = a;

    a = b;

    b = c;

}

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 fin;

    fin.open("input.txt");

    if (!fin.is\_open())

    {

        cout << "ko mo dc\n";

        return -1;

    }

    ofstream fout("output.txt");

    int a[50];

    int n = 0;

    while(fin >> a[n]) {

        n++;

    }

    QuickSort(a, 0, n-1);

    for(int i = 0; i < n; i++) {

        fout << a[i] << " ";

    }

    fin.close();

    fout.close();

    return 0;

}

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

#include <iostream>

#include <fstream>

using namespace std;

void Swap(int &a, int &b, int &sophepgan) {

    int c = a;

    a = b;

    b = c;

    sophepgan += 3;

}

void QuickSort(int a[], int left, int right, int &sophepgan, int &sophepsosanh)

{

    int i, j, x;

    if (left >= right)

    {

        sophepsosanh++;

        return;

    }

    x = a[(left + right) / 2];

       i = left;

    j = right;

    sophepgan += 3;  // sophepgan = sophepgan + 3; or sophepgan += 2; sophepgan++;

    while (i < j)

    {

        sophepsosanh++;

        while (a[i] < x)

        {

            i++;

            sophepgan++;

            sophepsosanh++;

        }

        sophepsosanh++;

        while (a[j] > x)

        {

            j--;

            sophepgan++;

            sophepsosanh++;

        }

        sophepsosanh++;

        if (i <= j)

        {

            sophepsosanh++;

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

            i++;

            sophepgan++;

            j--;

            sophepgan++;

        }

    }

    QuickSort(a, left, j, sophepgan, sophepsosanh);

    QuickSort(a, i, right, sophepgan, sophepsosanh);

}

int main()

{

    ifstream fin("input.txt");

    if (!fin.is\_open())

    {

        cout << "Ko mo dc file\n";

        return -1;

    }

    Ofstream fout("output.txt");

    int n = 0;

    int a[50];

    while (fin >> a[n]) {

    n++;

}

    int sophepgan = 0;

    int sophepsosanh = 0;

    QuickSort(a, 0, n - 1, sophepgan, sophepsosanh);

    fout << "Sorted array is ";

    for (int i = 0; i < n; ++i)

        fout << a[i] << " ";

    fout << "\nSo phep gan: " << sophepgan;

    fout << "\nSo phep so sanh: " << sophepsosanh;

    return 0;

}

# Merge Sort

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

int b[MAX], c[MAX], nb, nc; // Ghi chú: 2 mảng này dùng để làm gì? dùng để chia mảng a thành 2 mảng con b và c.

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? pa là biến đếm số phần tử đã chia, pb là biến đếm số phần tử đã chia vào mảng b, pc là biến đếm số phần tử đã chia vào mảng 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? vòng lặp này dùng để chia mảng a thành mảng b với số phần tử là k.

            b[pb] = a[pa];

        }

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

        { // Ghi chú: vòng lặp này có ý nghĩa gì? vòng lặp này dùng để chia mảng a thành mảng c với số phần tử là 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? câu lệnh này dùng để ghi phần tử của mảng b vào mảng a

    }

    while (pc < nc)

    {

        a[pa++] = c[pc++]; // Ghi chú: câu lệnh này có ý nghĩa gì? câu lệnh này dùng để ghi phần tử của mảng c vào mảng a

    }

}

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

{

    int rb;

    int 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++];

    }

}

## 2. Cho biết chức năng của từng hàm trên.

- Distribute: chia mảng a thành 2 mảng con b và c.

- Merge: trộn 2 mảng con b và c thành mảng tổng hợp a.

- MergeSubarr: trộn 2 mảng con b và c thành mảng kết quả a.

## **3. 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.** 5 2 9 3 7 2 4 11

#include <iostream>

#include <fstream>

using namespace std;

const int MAX = 100;

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

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

{

    int rb;

    int 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 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 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;

    for (k = 1; k < N; k \*= 2) {

        Distribute(a, N, nb, nc, k);

        Merge(a, nb, nc, k);

    }

}

int main()

{

    int a[] = {5, 2, 9, 3, 7, 2, 4, 11};

    int n = sizeof(a) / sizeof(a[0]);

        MergeSort(a, n);

        for (int i = 0; i < n; i++)

        {

        cout << a[i] << " ";

        }

        return 0;

    }

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

void MergeSort(int a[], int N) {

    int k;

    for (k = 1; k < N; k \*= 2) {

        Distribute(a, N, nb, nc, k);

        Merge(a, nb, nc, k);

    }

    for (int i = 0; i < N / 2; i++)

    {

        swap(a[i], a[N - i - 1]);

    }

}

# Áp Dụng – Nâng Cao

Cho dãy số nguyên A như sau: 12 2 15 -3 8 5 1 -8 6 0 4 15

## 1. Sắp xếp dãy trên tăng dần.

void MergeSort(int a[], int N) {

    int k;

    for (k = 1; k < N; k \*= 2) {

        Distribute(a, N, nb, nc, k);

        Merge(a, nb, nc, k);

    }

}

## 2. Suy ra số lớn thứ 3 trong dãy.

void timsolonthuba(int a[], int n, int &kq) {

    MergeSort(a, n);

    int count = 1;

    int prev = a[n-1];

    kq = -1;

    for(int i = n-2; i >= 0 && count < 3; i--) {

        if(a[i] != prev) {

            count++;

            prev = a[i];

            if(count == 3) {

                kq = a[i];

                return;

            }

        }

    }

}

## 3. Suy ra số lượng phần tử lớn nhất trong dãy.

void SoPTLonNhat(int a[], int n, int &count) {

    MergeSort(a, n);

    count = 1;

    int solonnhat = a[n-1];

    for(int i = n-2; i >= 0; i--) {

        if(a[i] == solonnhat) {

            count++;

        } else {

            break;

        }

    }

}

## 4. Sắp xếp dãy trên theo thứ tự giá trị tuyệt đối tăng dần.

void MergeSortGTTD(int a[], int N) {

    int k;

    for (k = 1; k < N; k \*= 2) {

        Distribute(a, N, nb, nc, k);

        Merge(a, nb, nc, k);

    }

    for (int i = 0; i < N - 1; i++) {

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

            if (abs(a[j]) > abs(a[j + 1])) {

                swap(a[j], a[j + 1]);

            }

        }

    }

}

## 5. Sắp xếp dãy trên theo quy luật sau:

**• Các số dương (nếu có) ở đầu mảng và có thứ tự giảm dần,**

**• Các số âm (nếu có) ở cuối mảng và có thứ tự tăng dần.**

void SortArray(int a[], int n) {

    vector<int> soduong;

    vector<int> soam;

    for (int i = 0; i < n; i++) {

        if (a[i] >= 0) {

            soduong.push\_back(a[i]);

        } else {

            soam.push\_back(a[i]);

        }

    }

    sort(soduong.begin(), soduong.end(), greater<int>());

    sort(soam.begin(), soam.end());

    int index = 0;

    for (int i = 0; i < soduong.size(); i++) {

        a[index++] = soduong[i];

    }

    for (int i = 0; i < soam.size(); i++) {

        a[index++] = soam[i];

    }

}

## 6. Sắp xếp dãy trên theo quy luật:

**• Các số chẵn (nếu có) ở đầu mảng và có thứ tự tăng dần**

**• Các số lẻ (nếu có) ở cuối mảng và có thứ tự giảm dần.**

void SortArray(int a[], int n) {

    vector<int> chan;

    vector<int> le;

    for (int i = 0; i < n; i++) {

        if (a[i] % 2 == 0) {

            chan.push\_back(a[i]);

        } else {

            le.push\_back(a[i]);

        }

    }

    sort(chan.begin(), chan.end());

    sort(le.begin(), le.end(), greater<int>());

    int index = 0;

    for (int i = 0; i < chan.size(); i++) {

        a[index++] = chan[i];

    }

    for (int i = 0; i < le.size(); i++) {

        a[index++] = le[i];

    }

}

## 7. Sắp xếp dãy trên theo quy luật:

**• Các số chẵn (nếu có) có thứ tự tăng dần,**

**• Các số lẻ (nếu có) có thứ tự giảm dần**

**• Tính chất chẵn/lẻ tại mỗi vị trí trong dãy A không thay đổi sau khi sắp xếp (tức là trước khi sắp xếp, tại vị trí i của dãy A là số chẵn/lẻ thì tại vị trí i của mảng sau khi sắp xếp cũng là số chẵn/lẻ)**

void SortPreserveParity(int a[], int n) {

    vector<int> chan, le;

    vector<int> vitri\_chan, vitri\_le;

    for (int i = 0; i < n; i++) {

        if (a[i] % 2 == 0) {

            chan.push\_back(a[i]);

            vitri\_chan.push\_back(i);

        } else {

            le.push\_back(a[i]);

            vitri\_le.push\_back(i);

        }

    }

 sort(chan.begin(), chan.end());

 sort(le.begin(), le.end(), greater<int>());

        for (int i = 0; i < chan.size(); i++) {

        a[vitri\_chan[i]] = chan[i];

    }

    for (int i = 0; i < le.size(); i++) {

        a[vitri\_le[i]] = le[i];

    }

}

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

struct Student {

    int stt;

    string mssv;

    string hoTen;

    int namSinh;

};

void danhsachsinhvien(const vector<Student>& dsSinhVien) {

    cout << setw(5) << "STT"

         << setw(10) << "MSSV"

         << setw(25) << "Ho va ten"

         << setw(10) << "Nam sinh" << endl;

    cout << string(50, '-') << endl;

    for(const auto& sv : dsSinhVien) {

        cout << setw(5) << sv.stt

             << setw(10) << sv.mssv

             << setw(25) << sv.hoTen

             << setw(10) << sv.namSinh << endl;

    }

}

## 9. Sắp xếp danh sách tăng dần theo mã số tăng dần

void radixSortMSSV(vector<Student>& dsSinhVien) {

    vector<Student> output(dsSinhVien.size());

    int max\_mssv = 0;

    for(const auto& sv : dsSinhVien) {

        max\_mssv = max(max\_mssv, stoi(sv.mssv));

    }

        for(int exp = 1; max\_mssv/exp > 0; exp \*= 10) {

        vector<int> count(10, 0);

    for(const auto& sv : dsSinhVien) {

            count[(stoi(sv.mssv)/exp)%10]++;

        }

    for(int i = 1; i < 10; i++) {

            count[i] += count[i-1];

        }

    for(int i = dsSinhVien.size()-1; i >= 0; i--) {

            output[count[(stoi(dsSinhVien[i].mssv)/exp)%10]-1] = dsSinhVien[i];

            count[(stoi(dsSinhVien[i].mssv)/exp)%10]--;

        }

        dsSinhVien = output;

    }

}

## 10. Sắp xếp danh sách tăng dần theo tên (thứ tự bảng chữ cái) và năm sinh (nếu trùng tên thì sắp theo năm sinh tăng dần).

    string LayTen(const string& fullName) {

    size\_t lastSpace = fullName.rfind(' ');

    if (lastSpace != string::npos) {

        return fullName.substr(lastSpace + 1);

    }

    return fullName;

}

void Sapxeptenvanam(vector<Student>& dsSinhVien) {

    sort(dsSinhVien.begin(), dsSinhVien.end(),

        [](const Student& a, const Student& b) {

            string TenA = LayTen(a.hoTen);

            string TenB = LayTen(b.hoTen);

            if (TenA != TenB)

                return TenA < TenB;

            return a.namSinh < b.namSinh;

        });

}

# Bài Tập Thêm

## 1. Viết chương trình so sánh các thuật toán Selection Sort, Heap Sort, Quick Sort, Merge Sort về các mặt sau:

**a. Thời gian chạy.**

**b. Số phép gán.**

**c. Số phép so sánh.**

#include <iostream>

#include <chrono>

#include <vector>

#include <random>

#include <iomanip>

using namespace std;

using namespace std::chrono;

struct Counter {

    long long sophepsosanh = 0;

    long long sophepgan = 0;

};

class SortingAnalyzer {

private:

    // Selection Sort

    static void selectionSort(vector<int>& arr, Counter& counter) {

        int n = arr.size();

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

            int min\_idx = i;

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

                counter.sophepsosanh++;

                if (arr[j] < arr[min\_idx]) {

                    min\_idx = j;

                }

            }

            if (min\_idx != i) {

                swap(arr[i], arr[min\_idx]);

                counter.sophepgan += 3;

            }

        }

    }

    // Heap Sort

    static void heapify(vector<int>& arr, int n, int i, Counter& counter) {

        int largest = i;

        int left = 2 \* i + 1;

        int right = 2 \* i + 2;

        counter.sophepsosanh++;

        if (left < n && arr[left] > arr[largest])

            largest = left;

        counter.sophepsosanh++;

        if (right < n && arr[right] > arr[largest])

            largest = right;

        if (largest != i) {

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

            counter.sophepgan += 3;

            heapify(arr, n, largest, counter);

        }

    }

    static void heapSort(vector<int>& arr, Counter& counter) {

        int n = arr.size();

        for (int i = n / 2 - 1; i >= 0; i--)

            heapify(arr, n, i, counter);

        for (int i = n - 1; i > 0; i--) {

            swap(arr[0], arr[i]);

            counter.sophepgan += 3;

            heapify(arr, i, 0, counter);

        }

    }

    // Quick Sort

    static int partition(vector<int>& arr, int low, int high, Counter& counter) {

        int pivot = arr[high];

        int i = low - 1;

        for (int j = low; j <= high - 1; j++) {

            counter.sophepsosanh++;

            if (arr[j] < pivot) {

                i++;

                swap(arr[i], arr[j]);

                counter.sophepgan += 3;

            }

        }

        swap(arr[i + 1], arr[high]);

        counter.sophepgan += 3;

        return i + 1;

    }

    static void quickSort(vector<int>& arr, int low, int high, Counter& counter) {

        if (low < high) {

            int pi = partition(arr, low, high, counter);

            quickSort(arr, low, pi - 1, counter);

            quickSort(arr, pi + 1, high, counter);

        }

    }

    // Merge Sort

    static void merge(vector<int>& arr, int left, int mid, int right, Counter& counter) {

        vector<int> temp(right - left + 1);

        int i = left, j = mid + 1, k = 0;

        while (i <= mid && j <= right) {

            counter.sophepsosanh++;

            if (arr[i] <= arr[j]) {

                temp[k++] = arr[i++];

            } else {

                temp[k++] = arr[j++];

            }

            counter.sophepgan++;

        }

        while (i <= mid) {

            temp[k++] = arr[i++];

            counter.sophepgan++;

        }

        while (j <= right) {

            temp[k++] = arr[j++];

            counter.sophepgan++;

        }

        for (i = 0; i < k; i++) {

            arr[left + i] = temp[i];

            counter.sophepgan++;

        }

    }

    static void mergeSort(vector<int>& arr, int left, int right, Counter& counter) {

        if (left < right) {

            int mid = left + (right - left) / 2;

            mergeSort(arr, left, mid, counter);

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

            merge(arr, left, mid, right, counter);

        }

    }

public:

    static void compareThuattoan(int size) {

        vector<vector<int>> arrays(4);

        // Tao random data

        random\_device rd;

        mt19937 gen(rd());

        uniform\_int\_distribution<> dis(1, 10000);

        // Initialize arrays with same random data

        vector<int> original(size);

        for (int i = 0; i < size; i++) {

            original[i] = dis(gen);

        }

        for (int i = 0; i < 4; i++) {

            arrays[i] = original;

        }

        vector<Counter> counters(4);

        vector<double> times(4);

        // Selection Sort

        auto start = high\_resolution\_clock::now();

        selectionSort(arrays[0], counters[0]);

        auto stop = high\_resolution\_clock::now();

        times[0] = duration\_cast<microseconds>(stop - start).count() / 1000.0;

        // Heap Sort

        start = high\_resolution\_clock::now();

        heapSort(arrays[1], counters[1]);

        stop = high\_resolution\_clock::now();

        times[1] = duration\_cast<microseconds>(stop - start).count() / 1000.0;

        // Quick Sort

        start = high\_resolution\_clock::now();

        quickSort(arrays[2], 0, size - 1, counters[2]);

        stop = high\_resolution\_clock::now();

        times[2] = duration\_cast<microseconds>(stop - start).count() / 1000.0;

        // Merge Sort

        start = high\_resolution\_clock::now();

        mergeSort(arrays[3], 0, size - 1, counters[3]);

        stop = high\_resolution\_clock::now();

        times[3] = duration\_cast<microseconds>(stop - start).count() / 1000.0;

        cout << "\nArray size: " << size << endl;

        cout << setw(15) << "Thuattoan" << setw(15) << "Thoigian(ms)"

             << setw(15) << "Sophepsosanh" << setw(15) << "Sophepgan" << endl;

        cout << string(60, '-') << endl;

        vector<string> names = {"Selection Sort", "Heap Sort", "Quick Sort", "Merge Sort"};

        for (int i = 0; i < 4; i++) {

            cout << setw(15) << names[i]

                 << setw(15) << fixed << setprecision(3) << times[i]

                 << setw(15) << counters[i].sophepsosanh

                 << setw(15) << counters[i].sophepgan << endl;

        }

    }

};

int main() {

    vector<int> sizes = {1000, 5000};

    for (int size : sizes) {

        SortingAnalyzer::compareThuattoan(size);

    }

    return 0;

}

## 2. Trong thuật toán QuickSort, nếu lấy x là phần tử dầu 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 <random>

#include <iomanip>

using namespace std;

using namespace std::chrono;

class QuickSortAnalyzer {

private:

    static int partitionDautien(vector<int>& arr, int low, int high) {

        int pivot = arr[low];

        int i = low + 1;

        int j = high;

        while (true) {

            while (i <= j && arr[i] <= pivot) i++;

            while (i <= j && arr[j] > pivot) j--;

            if (i >= j) break;

            swap(arr[i], arr[j]);

        }

        swap(arr[low], arr[j]);

        return j;

    }

    static int partitionGiua(vector<int>& arr, int low, int high) {

        int mid = low + (high - low) / 2;

        swap(arr[mid], arr[low]);

        return partitionDautien(arr, low, high);

    }

    static void quickSortDautien(vector<int>& arr, int low, int high) {

        if (low < high) {

            int pi = partitionDautien(arr, low, high);

            quickSortDautien(arr, low, pi - 1);

            quickSortDautien(arr, pi + 1, high);

        }

    }

    static void quickSortGiua(vector<int>& arr, int low, int high) {

        if (low < high) {

            int pi = partitionGiua(arr, low, high);

            quickSortGiua(arr, low, pi - 1);

            quickSortGiua(arr, pi + 1, high);

        }

    }

public:

    static void SoSanhQuickSorts(int size) {

        vector<vector<int>> arrays(2);

        random\_device rd;

        mt19937 gen(rd());

        uniform\_int\_distribution<> dis(1, 10000);

        vector<int> original(size);

        for (int i = 0; i < size; i++) {

            original[i] = dis(gen);

        }

        arrays[0] = original;

        arrays[1] = original;

        auto start = high\_resolution\_clock::now();

        quickSortDautien(arrays[0], 0, size - 1);

        auto end = high\_resolution\_clock::now();

        double timeFirst = duration\_cast<microseconds>(end - start).count() / 1000.0;

        start = high\_resolution\_clock::now();

        quickSortGiua(arrays[1], 0, size - 1);

        end = high\_resolution\_clock::now();

        double timeMiddle = duration\_cast<microseconds>(end - start).count() / 1000.0;

        cout << "\nArray size: " << size << endl;

        cout << setw(25) << "Phan tu" << setw(15) << "Thoigian(ms)" << endl;

        cout << string(40, '-') << endl;

        cout << setw(25) << "Phan tu dau" << setw(15) << fixed << setprecision(3) << timeFirst << endl;

        cout << setw(25) << "Phan tu giua" << setw(15) << timeMiddle << endl;

    }

};

int main() {

    vector<int> sizes = {1000, 5000, 10000, 50000, 100000}; //Chinh size tuy muon

    cout << "So sanh phan tu dau va giua" << endl;

    for (int size : sizes) {

        QuickSortAnalyzer::SoSanhQuickSorts(size);

    }

    return 0;

}

## 3. Tìm hiểu và cài đặt thuật toán Insertion Sort.

- Sắp xếp chèn là một thuật toán sắp xếp đơn giản hoạt động tương tự như cách bạn sắp xếp các thẻ chơi trong tay. Mảng hầu như được chia thành một phần được sắp xếp và một phần chưa được sắp xếp. Các giá trị từ phần chưa được sắp xếp được chọn và đặt ở vị trí chính xác trong phần được sắp xếp.

## 4. Tìm hiểu và cài đặt thuật toán Binary Insertion Sort.

- Binary Insertion Sort sử dụng tìm kiếm nhị phân để tìm vị trí thích hợp để chèn mục đã chọn ở mỗi lần lặp. Trong trường hợp chèn thông thường, việc sắp xếp lấy O (i) (ở lần lặp thứ i). Chúng ta có thể giảm nó thành O (logi) bằng cách sử dụng tìm kiếm nhị phân. Nói chung, thuật toán vẫn có thời gian chạy trong trường hợp xấu nhất là O (n2) vì chuỗi hoán đổi cần thiết cho mỗi lần chèn.

## 5. Tìm hiểu và cài đặt thuật toán Interchange Sort.

- Thuật toán Interchange sort chia mảng ra làm 2 mảng con, mảng bên trái đã được sắp xếp và bên phải chưa được sắp xếp. Thuật toán Interchange sort sắp xếp một mảng bằng cách đổi chỗ liên tục phần tử ở đầu đoạn chưa được sắp xếp(không phải đầu mảng) với phần tử nhỏ hơn nó cho tới khi duyệt tới cuối mảng, thực hiện lặp cho tới khi mảng hoàn toàn được sắp xếp.

## 6. Tìm hiểu và cài đặt thuật toán Bubble Sort.

- Thuật toán sắp xếp bubble sort thực hiện sắp xếp dãy số bằng cách lặp lại công việc đổi chỗ 2 số liên tiếp nhau nếu chúng đứng sai thứ tự cho đến khi dãy số được sắp xếp.

## 7. Tìm hiểu và cài đặt thuật toán Shaker Sort.

- Shaker Short hay còn được gọi là thuật toán sắp xếp Cocktail là một phiên bản nâng cấp của giải thuật Bubble Sort. Thay vì đưa lần lượt từng phần tử nhỏ nhất (hoặc lớn nhất) về 1 phía của mảng như Bubble Sort thì Shaker Sort sau khi đưa phần tử nhỏ/lớn nhất lên đầu dãy, sau đấy lại đưa phần tử lớn/nhỏ nhất về cuối dãy. Như vậy, trong một lần duyệt, Shaker sort sẽ đưa ít nhất hai số về đúng với vị trí của nó.

## 8. Tìm hiểu và cài đặt thuật toán Shell Sort.

- Shell Sort là một giải thuật sắp xếp mang lại hiệu quả cao dựa trên giải thuật sắp xếp chèn (Insertion Sort). Giải thuật này tránh các trường hợp phải tráo đổi vị trí của hai phần tử xa nhau trong giải thuật sắp xếp chọn (nếu như phần tử nhỏ hơn ở vị trí bên phải khá xa so với phần tử lớn hơn bên trái).

- Đầu tiên, giải thuật này sử dụng giải thuật sắp xếp chọn trên các phần tử có khoảng cách xa nhau, sau đó sắp xếp các phần tử có khoảng cách hẹp hơn. Khoảng cách này còn được gọi là khoảng (interval) – là số vị trí từ phần tử này tới phần tử khác. Khoảng này được tính dựa vào công thức Knuth như sau:

h = h \* 3 + 1

*trong đó:*

h là Khoảng (interval) với giá trị ban đâu là 1

- Giải thuật này khá hiệu quả với các tập dữ liệu có kích cỡ trung bình khi mà độ phức tạp trường hợp xấu nhất và trường hợp trung bình là O(n), với n là số phần tử.

## 9. Tìm hiểu và cài đặt thuật toán Radix Sort.

- Radix Sort là một thuật toán sắp xếp đặc biệt, dựa trên việc chia từng phần tử của dãy số thành các hạng riêng biệt, theo từng vị trí của chữ số. Mỗi lần chia theo một vị trí (chữ số hàng đơn vị, hàng chục, hàng trăm…), thuật toán sẽ tạo ra các “hạng con” chứa các phần tử có cùng giá trị ở vị trí đó.