TRƯỜNG ĐẠI HỌC SÀI GÒN

KHOA CÔNG NGHỆ THÔNG TIN

Bài Tập Sắp Xếp

Sinh viên thực hiện: Huỳnh Minh Hoàng

Bùi Viết Quang Vinh

Nguyễn Tấn Phát

Đặng Đức Thịnh

GIảng viên hướng dẫn: Đỗ Như Tài

Thành phố Hồ Chí Minh, tháng 3 năm 2025

# Mục lục

[Mục lục 2](#_Toc192877515)

[Phân công 3](#_Toc192877516)

[1. Câu hỏi 3](#_Toc192877517)

[2. Bài tập cơ sở 3](#_Toc192877518)

[3. Bài tập ứng dụng 3](#_Toc192877519)

[Nội dung 4](#_Toc192877520)

[Phần I: Câu Hỏi 4](#_Toc192877521)

[Câu 2 5](#_Toc192877522)

[Câu 3 6](#_Toc192877523)

[Câu 4 8](#_Toc192877524)

[BÀI TẬP CƠ SỞ 13](#_Toc192877525)

[Bài 1 (Đức Thịnh): 13](#_Toc192877526)

[Bài 2 (Quang Vinh): 16](#_Toc192877527)

[Bài 3 ( Minh Hoàng): 23](#_Toc192877528)

[Bài 4 ( Tấn Phát): 26](#_Toc192877529)

[File tạo txt: 35](#_Toc192877530)

[Bài Tập Ứng Dụng 37](#_Toc192877531)

[Bài 1 ( Đức Thịnh): 37](#_Toc192877532)

[Bài 2 ( Tấn Phát): 39](#_Toc192877533)

[Bài 3 (Quang Vinh): 40](#_Toc192877534)

[Bài 4 ( Minh Hoàng) : 45](#_Toc192877535)

[Bài 5 ( Đức Thịnh) : 50](#_Toc192877536)

[Bài 6 (Tấn Phát): 52](#_Toc192877537)

[Bài 7 ( Minh Hoàng): 54](#_Toc192877538)

[Bài 8 (Quang Vinh): 56](#_Toc192877539)

[Bài 9: 58](#_Toc192877540)

# Phân công

## 1. Câu hỏi

Câu 1,2: Huỳnh Minh Hoàng – MSSV: 3124411098

Câu 3,4: Nguyễn Tấn Phát – MSSV: 3124411218

## 2. Bài tập cơ sở

Câu 1: Đặng Đữc Thịnh – MSSV: 3124411290

Câu 2: Bùi Viết Quang Vinh – MSSV: 3124411347

Câu 3: Huỳnh Minh Hoàng

Câu 4: Nguyễn Tấn Phát

## 3. Bài tập ứng dụng

Câu 1: Đặng Đức Thịnh

Câu 2: Nguyễn Tấn Phát

Câu 3: Bùi Viết Quang Vinh

Câu 4: Huỳnh Minh Hoàng

Câu 5: Đặng Đức Thịnh

Câu 6: Nguyễn Tấn Phát

Câu 7: Huỳnh Minh Hoàng

Câu 8: Bùi Viết Quang Vinh

Câu 9: Cả nhóm.

# Nội dung

## Phần I: Câu Hỏi

1. InterChange Sort ( Sắp xếp đổi chỗ trực tiếp):

- Tư tưởng: So sánh từng phần tử với các phần tử còn lại đứng sau nó, nếu hai phần tử được so sánh đứng không đúng thứ tự trong mảng thì ta hoán đổi chúng với nhau và tiếp tục với các phần tử còn lại.

2. Bubble Sort ( Sắp xếp nổi bọt):

- Tư tưởng: So sánh từng cặp phần tử với nhau và hoán đổi nếu chúng không đúng vị trí tương ứng .

3. Selection Sort ( Sắp xếp chọn):

- Tư tưởng: Tìm phần tử nhỏ nhất trong mảng và đưa nó lên đầu, sau đó tìm phần tử nhỏ nhất trong mảng còn lại tiếp tục đưa lên đầu và lặp lại tới khi mảng đã được sắp xếp.

4. Heap Sort ( Sắp xếp đống):

- Tư tưởng: Xây dựng một cây nhị phân dạng đống, sau đó trích xuất phần tử lớn nhất hoặc nhỏ nhất tùy dạng sắp xếp ra khỏi đống và tiêp tục sắp xếp cho đến khi hoàn tất.

5. Merge Sort ( Sắp xếp trộn):

- Tư tưởng: Liên tục chia đôi danh sách thành hai phần nhỏ hơn cho tới khi chỉ còn một phần tử thì trộn những danh sách con lại theo thứ tự đúng.

6. Quick Sort ( Sắp xếp nhanh):

- Tư tưởng: Chọn 1 phần tử làm chốt (pivot), sau đó chia danh sách thành hai phần ( một phần nhỏ hơn chốt và một phần lớn hơn), tiếp tục cho tới khi danh sách được sắp xếp hoàn thành.

7. Insertion Sort ( Sắp xếp chèn):

- Tư tưởng: Duyệt qua từng phần tử của danh sách, lấy từng phần tử và chèn vào đúng vị trí trong danh sách con đã được sắp xếp phía trước.

8. Radix Sort ( Sắp xếp cơ số):

- Tư tưởng: Sắp xếp các phần tử theo từng chữ số, từ chữ số ít quan trọng nhất tới quan trọng nhất (đối với só nguyên) hoặc theo ký tự nếu đối với chuỗi)

## Câu 2

Thuật toán thích nhất: Trong các thuật toán sắp xếp, em thích nhất là thuật toán sắp xếp nhanh (Quick sort) vì đây là thuật toán tương đối dễ hiểu, dễ áp dụng với nhiều bài tập và có độ phức tạp tương đối thấp.

Thuật toán không thích nhất: Thuật toán em không thích là Insertion sort vì nó không tối ưu tốc độ xử lý và tương đối phức tạp.

## Câu 3

I. Sắp xếp nội (Internal Sorting):

Là những thuật toán sắp xếp mà toàn bộ dữ liệu được lưu trực tiếp trong bộ nhớ RAM khi thực hiện.

1. Bubble Sort (Sắp xếp nổi bọt)

Thuật toán này lần lượt so sánh từng cặp phần tử liên tiếp, nếu phần tử trước nhỏ hơn phần tử sau thì đổi chỗ. Quá trình lặp lại cho đến khi mảng được sắp xếp hoàn toàn.

Độ phức tạp: O(n²) (Chậm do số lượng so sánh lớn)

Nhận xét: Dễ hiểu, dễ cài đặt. Hiệu suất thấp với mảng lớn.

2. Selection Sort (Sắp xếp chọn)

Thuật toán tìm phần tử lớn nhất trong dắng sách và đặt nó về đầu danh sách. Quá trình lặp lại với phần tử lớn thứ hai, thứ ba...

Độ phức tạp: O(n²)

Nhận xét: Tốt hơn Bubble Sort vì số lượng hoán đổi ít hơn. Vẫn chậm khi xử lý dữ liệu lớn.

3. Quick Sort (Sắp xếp nhanh)

Chọn một phần tử pivot làm chốt, chia mảng thành hai phần: Phần tử lớn hơn pivot đặt bên trái. Phần tử nhỏ hơn pivot đặt bên phải. Gọi đệ quy để sắp xếp hai phần này.

Độ phức tạp: O(n log n)

Nhận xét: Hiệu quả cao, xử lý dữ liệu lớn nhanh. Có thể bị chậm O(n²) nếu chọn pivot không tốt.

II. Sắp xếp ngoại (External Sorting)

- Dùng khi dữ liệu quá lớn, không thể lưu trực tiếp trong RAM.

4. External Merge Sort (Sắp xếp trộn bên ngoài)

Chia dữ liệu lớn thành nhiều tập con, mỗi tập vừa với bộ nhớ RAM. Sắp xếp từng tập con rồi trộn dần.

Độ phức tạp: O(n log n)

Nhận xét: Tốt cho dữ liệu rất lớn. Chậm do cần nhiều thao tác đọc/ghi từ bộ nhớ ngoài.

## Câu 4

1. Interchange Sort (Đổi chỗ trực tiếp)

Ưu điểm:

- Dễ hiểu, dễ cài đặt.

- Hoạt động tốt với danh sách nhỏ.

Nhược điểm:

- Độ phức tạp O(n²) → Chậm khi dữ liệu lớn.

- Không tối ưu được số lần đổi chỗ.

Cách khắc phục:

- Dùng Bubble Sort có tối ưu hoặc Insertion Sort để giảm số lần đổi chỗ.

- Chuyển sang thuật toán O(n log n) như Quick Sort hoặc Merge Sort cho dữ liệu lớn.

2. Bubble Sort (Sắp xếp nổi bọt)

Ưu điểm:

- Cài đặt đơn giản.

- Có thể tối ưu bằng cách dừng sớm nếu dãy đã sắp xếp.

Nhược điểm:

- Chậm, O(n²) trong trường hợp xấu nhất.

Quá nhiều lần đổi chỗ, tốn thời gian.

Cách khắc phục:

Dùng Bubble Sort cải tiến (dừng sớm khi không có đổi chỗ).

Dùng Insertion Sort cho danh sách gần như đã sắp xếp.

Chuyển sang Merge Sort hoặc Quick Sort nếu danh sách lớn.

3. Selection Sort (Sắp xếp chọn trực tiếp)

Ưu điểm:

- Ít đổi chỗ hơn Bubble Sort.

- Dễ cài đặt, không cần bộ nhớ phụ.

Nhược điểm:

- O(n²) → Không phù hợp với dữ liệu lớn.

- Không ổn định (nếu có phần tử trùng nhau, thứ tự ban đầu có thể thay đổi).

Cách khắc phục:

- Dùng Heap Sort nếu cần tối ưu hiệu suất.

- Dùng Merge Sort nếu cần thuật toán ổn định.

4. Insertion Sort (Sắp xếp chèn trực tiếp)

Ưu điểm:

- Nhanh với dữ liệu nhỏ hoặc gần như đã sắp xếp (O(n)).

- Ít đổi chỗ hơn Bubble Sort và Selection Sort.

Nhược điểm:

- O(n²) với danh sách lớn và ngẫu nhiên.

- Không hiệu quả khi dữ liệu quá lộn xộn.

Cách khắc phục:

- Dùng Binary Insertion Sort để giảm số lần so sánh.

- Dùng Shell Sort hoặc Merge Sort khi dữ liệu lớn.

5. Quick Sort (Sắp xếp nhanh)

Ưu điểm:

- Nhanh, O(n log n) trong trường hợp trung bình.

- Hiệu suất tốt hơn Merge Sort vì không cần bộ nhớ phụ.

Nhược điểm:

- Trường hợp xấu (chọn pivot không tốt) có thể O(n²).

- Không ổn định.

Cách khắc phục:

- Chọn pivot tốt hơn (median-of-three, random pivot).

- Dùng Merge Sort nếu cần thuật toán ổn định.

6. Merge Sort (Sắp xếp trộn)

Ưu điểm:

- Ổn định, luôn O(n log n).

- Tốt cho dữ liệu lớn, đặc biệt là khi dùng đa luồng (parallel processing).

Nhược điểm:

- Cần bộ nhớ phụ O(n), tốn RAM.

- Chậm hơn Quick Sort khi dữ liệu nhỏ.

Cách khắc phục:

- Chỉ dùng khi cần ổn định hoặc khi dữ liệu quá lớn.

- Nếu bộ nhớ hạn chế, dùng Quick Sort hoặc Heap Sort.

7. Heap Sort (Sắp xếp bằng đống)

Ưu điểm:

- O(n log n) luôn, không bị O(n²) như Quick Sort.

- Không cần bộ nhớ phụ.

Nhược điểm:

- Không ổn định.

- Chạy chậm hơn Quick Sort trên dữ liệu ngẫu nhiên.

Cách khắc phục:

- Dùng Quick Sort khi tốc độ quan trọng hơn tính ổn định.

- Dùng Merge Sort nếu cần ổn định.

# BÀI TẬP CƠ SỞ

## Bài 1 (Đức Thịnh):

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

void bubbleSort(vector<int>& arr) {

    int n = arr.size();

    for (int i = 0; i < n - 1; i++) {

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

            if (arr[j] > arr[j + 1]) {

                swap(arr[j], arr[j + 1]);

            }

        }

    }

}

void selectionSort(vector<int>& arr) {

    int n = arr.size();

    for (int i = 0; i < n - 1; i++) {

        int minIdx = i;

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

            if (arr[j] < arr[minIdx]) {

                minIdx = j;

            }

        }

        swap(arr[i], arr[minIdx]);

    }

}

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;

    }

}

int main() {

    vector<int> arr = {39, 8, 5, 1, 3, 6, 9, 12, 4, 7, 10};

    vector<int> arrCopy;

    // Bubble Sort

    arrCopy = arr; // Copy the original array

    bubbleSort(arrCopy);

    cout << "Bubble Sort: ";

    for (int num : arrCopy) {

        cout << num << " ";

    }

    cout << endl;

    // Selection Sort

    arrCopy = arr; // Copy the original array

    selectionSort(arrCopy);

    cout << "Selection Sort: ";

    for (int num : arrCopy) {

        cout << num << " ";

    }

    cout << endl;

    // Insertion Sort

    arrCopy = arr; // Copy the original array

    insertionSort(arrCopy);

    cout << "Insertion Sort: ";

    for (int num : arrCopy) {

        cout << num << " ";

    }

    cout << endl;

    return 0;

}

Test Key: Input: 25, 14, 7, 3, 19, 11, 8, 16, 2, 5, 13

Output (sắp xếp tăng dần bằng các thuật toán): Interchange Sort, Selection Sort, Insertion Sort, Bubble Sort đều cho kết quả: 2, 3, 5, 7, 8, 11, 13, 14, 16, 19, 25

## Bài 2 (Quang Vinh):

#include<iostream>

#include<ctime>

#include<cstdlib>

#include<time.h>

using namespace std;

//Ham tao mang ngau nhien

void createArray\_Random(int arr[],int size)

{

    for(int i=0; i < size;i++)

    {

        arr[i]=rand()%(size\*10);//so ngau nhien tu 0->size\*10

    }

}

//Heap sort

void Heapify(int arr[],int n,int i)

{

    int left=2\*i+1;

    int right=2\*i+2;

    int largest=i;

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

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

    // Nếu largest thay đổi, hoán đổi và đệ quy

    if (largest != i)

    {

        swap(arr[i], arr[largest]);

        Heapify(arr, n, largest);

    }

}

    //Ham sap xep Heap

void HeapSort(int arr[],int n)

{

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

    //doi cho node dau tien voi node cuoi cung

    for(int i=n-1;i>=0;i--)

    {

        swap(arr[0],arr[i]);

        Heapify(arr,i,0);

    }

}

    //ham

void Merge(int arr[], int low, int mid, int high) {

    int n1 = mid - low + 1;   // Kích thước mảng con bên trái

    int n2 = high - mid;      // Kích thước mảng con bên phải

    int left[n1], right[n2];  // Tạo mảng tạm để lưu hai mảng con

    // Sao chép phần tử vào mảng tạm left[]

    for (int i = 0; i < n1; i++)

        left[i] = arr[low + i];

    // Sao chép phần tử vào mảng tạm right[]

    for (int j = 0; j < n2; j++)

        right[j] = arr[mid + 1 + j];

    int i = 0, j = 0, k = low;  // Chỉ số cho mảng left, right và mảng gốc arr

    // Trộn hai mảng con lại theo thứ tự tăng dần

    while (i < n1 && j < n2) {

        if (left[i] <= right[j]) {  // Chọn phần tử nhỏ hơn để đưa vào arr[k]

            arr[k] = left[i];

            i++;

        } else {

            arr[k] = right[j];

            j++;

        }

        k++;

    }

    // Sao chép các phần tử còn lại của mảng left[] nếu có

    while (i < n1) {

        arr[k] = left[i];

        i++;

        k++;

    }

    // Sao chép các phần tử còn lại của mảng right[] nếu có

    while (j < n2) {

        arr[k] = right[j];

        j++;

        k++;

    }

}

    //Ham sap xep Merge

void MergeSort(int arr[], int low, int high) {

    if (low < high) {

        int mid = (low + high) / 2;

        MergeSort(arr, low, mid);

        MergeSort(arr, mid + 1, high);

        Merge(arr, low, mid, high);

    }

}

//Phan vung mang

int Partition(int arr[], int low, int high)

{

    int pivot = arr[high]; // Chọn phần tử cuối làm pivot

    int i = low - 1; // Chỉ số của phần tử nhỏ hơn pivot

    for (int j = low; j < high; j++)

    {

        if (arr[j] < pivot)

        {

            i++;

            swap(arr[i], arr[j]);

        }

    }

    swap(arr[i + 1], arr[high]); // Đưa pivot về đúng vị trí

    return i + 1; // Trả về vị trí mới của pivot

}

//Ham sap xep QuickSort

void QuickSort(int arr[], int low, int high)

{

    if (low < high)

    {

        int pivotIndex = Partition(arr, low, high);

        QuickSort(arr, low, pivotIndex - 1);

        QuickSort(arr, pivotIndex + 1, high);

    }

}

//Hàm đo thời gian chạy

void MeasureTime(int size)

{

    int\* arr=new int[size];

    createArray\_Random(arr,size);

    clock\_t start, end;

    double  time\_use\_Heap,time\_use\_Merge,time\_use\_Quick;

    //Đo thời gian thực hiện heap sort

    int \*arr\_Heap=new int [size];//cấp phát mảng động

    //Cấp phát mảng ngẫu nhiên

    copy(arr,arr+size,arr\_Heap);// Sao chép nội dung của arr vào arr\_heap để đảm bảo cùng dữ liệu ban đầu

    createArray\_Random(arr\_Heap,size);

    start=clock();//bắt đầu đo thời gian

    HeapSort(arr\_Heap,size);//gọi hàm Heáport để sắp xếp mảng arr\_Heap

    end =clock();//kết thúc đo thời gian

    time\_use\_Heap=(double)(end-start)/CLOCKS\_PER\_SEC;//tính thời gian chạy đổi sang giây

    delete[] arr\_Heap;//giải phóng bộ nhớ

    //Đo thời gian thực hiện merge sort

    int \*arr\_Merge=new int [size];

    copy(arr,arr+size,arr\_Merge);

    createArray\_Random(arr\_Merge,size);

    start=clock();

    MergeSort(arr\_Merge,0,size-1);

    end=clock();

    time\_use\_Merge=(double)(end-start)/CLOCKS\_PER\_SEC;

    delete[] arr\_Merge;

    //Đo thời gian thực hiện quick sort

    int \* arr\_Quick=new int [size];

    copy(arr,arr+size,arr\_Quick);

    createArray\_Random(arr\_Quick,size);

    start=clock();

    QuickSort(arr\_Quick,0,size-1);

    end=clock();

    time\_use\_Quick=(double)(end-start)/CLOCKS\_PER\_SEC;

    delete[] arr\_Quick;

    //Xuat thoi gian thuc hien

        cout<<"Size: "<<size<<endl;

    cout<<"Thoi gian thuc hien HeappSort: "<<time\_use\_Heap<<"s"<<endl;

    cout<<"Thoi gian thuc hien MergeSort: "<<time\_use\_Merge<<"s"<<endl;

    cout<<"Thoi gian thuc hien QuickSort: "<<time\_use\_Quick<<"s"<<endl;

    delete[] arr;

}

int main()

{

    srand(time(0));

    // taoj kich thuoc cho mang ngau nhien

    int sizes[] = {1000, 5000, 10000, 50000, 100000};

    cout<<"Đang chờ thời gian chạy ..."<<endl;

    for(int i=0;i<5;i++)

    {

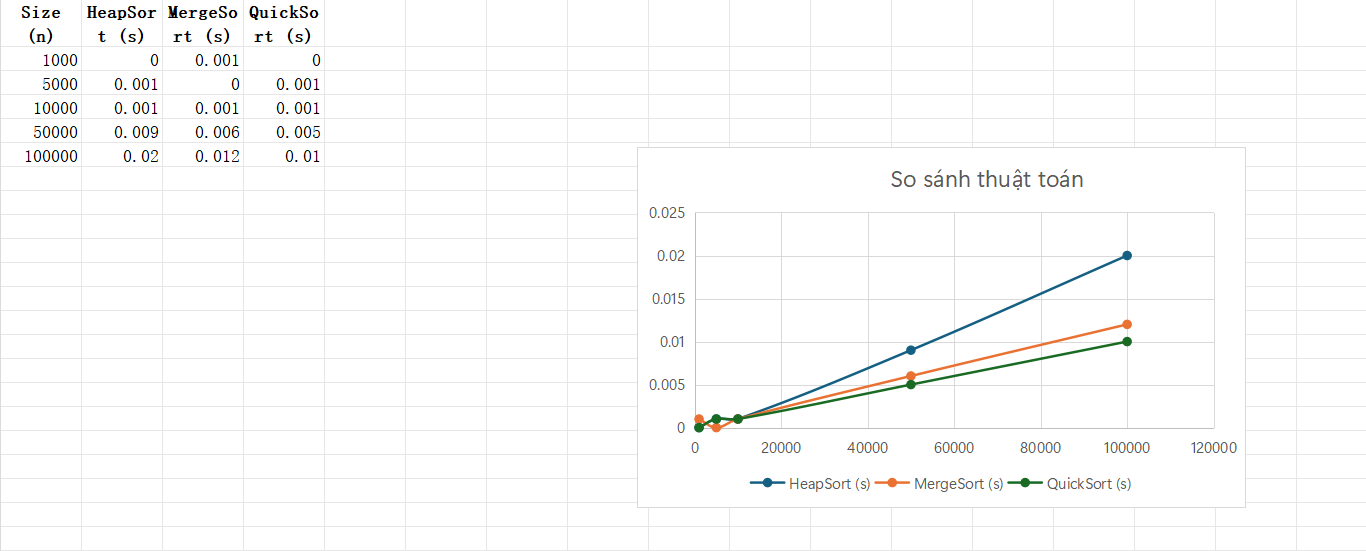
        MeasureTime(sizes[i]);

    }

    return 0;}

Test Key: Input: 17, 9, 4, 12, 6, 15, 1, 10, 8, 3

Output (sắp xếp tăng dần): Quick Sort, Merge Sort, Heap Sort đều cho kết quả: 1, 3, 4, 6, 8, 9, 10, 12, 15, 17



## Bài 3 ( Minh Hoàng):

#include<iostream>

#include<ctime>

#include<cstdlib>

#include<time.h>

using namespace std;

//Ham tao mang ngau nhien

void createArray\_Random(int arr[],int size)

{

    for(int i=0; i < size;i++)

    {

        arr[i]=rand()%(size\*10);//so ngau nhien tu 0->size\*10

    }

}

//Heap sort

void Heapify(int arr[],int n,int i)

{

    int left=2\*i+1; // gia tri cua nut trai

    int right=2\*i+2; // gia tri cua nut phai

    int largest=i; // gia su nut ban dau la lon nhat

    if(left<n && arr[left]>arr[largest]) largest = left; // neu phan tu ben trai lon hon co hieu thi gan gia tri vo no

    if(right<n && arr[right]>arr[largest]) largest =right; // neu phan tu ben phai lon hon co hieu thi gan gia tri vo no

    // Nếu largest thay đổi, hoán đổi và đệ quy

    if (largest != i) // neu phan tu lon nhat khac voi phan tu dang xet

    {

        swap(arr[i], arr[largest]); // hoan doi 2 phan tu do voi nhau

        Heapify(arr, n, largest); //goi lai ham heap sort de tiep tuc sap xep

    }

}

    //Ham sap xep Heap

void HeapSort(int arr[],int n)

{

    //xay dung cay

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

    //doi cho node dau tien voi node cuoi cung

    for(int i=n-1;i>=0;i--)

    {

        swap(arr[0],arr[i]); // dua phan tu lon nhat hien tai ve cuoi mang

        Heapify(arr,i,0); // goi ham de tiep tuc duy tri tinh chat cua cay

    }

}

//Hàm đo thời gian chạy

void MeasureTime(int size)

{

    int\* arr=new int[size];

    createArray\_Random(arr,size);

    clock\_t start, end;

    double  time\_use\_Heap,time\_use\_Merge,time\_use\_Quick;

    //Đo thời gian thực hiện heap sort

    int \*arr\_Heap=new int [size];//cấp phát mảng động

    //Cấp phát mảng ngẫu nhiên

    copy(arr,arr+size,arr\_Heap);// Sao chép nội dung của arr vào arr\_heap để đảm bảo cùng dữ liệu ban đầu

    createArray\_Random(arr\_Heap,size);

    start=clock();//bắt đầu đo thời gian

    HeapSort(arr\_Heap,size);//gọi hàm Heáport để sắp xếp mảng arr\_Heap

    end =clock();//kết thúc đo thời gian

    time\_use\_Heap=(double)(end-start)/CLOCKS\_PER\_SEC;//tính thời gian chạy đổi sang giây

    delete[] arr\_Heap;//giải phóng bộ nhớ

    cout<<”Thoi gian thuc hien HeappSort: “<<time\_use\_Heap<<”s”<<endl;

    delete[] arr;

}

int main()

{

    srand(time(0));

    // taoj kich thuoc cho mang ngau nhien

    int sizes[] = {10, 100, 200, 300, 400, 500, 600, 700, 800, 900,

        1000, 2000, 3000, 4000, 5000, 6000, 7000, 8000, 9000, 10000};

    cout<<”Đang chờ thời gian chạy …”<<endl;

    for(int i=0;i<20;i++)

    {

        MeasureTime(sizes[i]);

    }

    return 0;

}

Test Key: Input: Với n = 10: 31, 88, 14, 62, 27, 95, 41, 73, 6, 50

Output (sắp xếp tăng dần): 6, 14, 27, 31, 41, 50, 62, 73, 88, 95

## Bài 4 ( Tấn Phát):

#include<iostream>

#include<ctime>

#include<cstdlib>

#include<fstream>

using namespace std;

const int size = 30000;

// Hàm đọc mảng từ file

void readArrayFromFile(const char\* filename, int arr[], int &size) {

    ifstream file(filename);

    if (!file) {

        cout << "Không thể mở file!\n";

        exit(1);

    }

    file >> size;  // Đọc kích thước mảng

    for (int i = 0; i < size; i++) {

        file >> arr[i];  // Đọc từng phần tử

    }

    file.close();

}

// Hàm ghi mảng ra file

void writeArrayToFile(const char\* filename, int arr[], int size) {

    ofstream file(filename);

    if (!file) {

        cout << "Không thể mở file để ghi!\n";

        exit(1);

    }

    for (int i = 0; i < size; i++) {

        file << arr[i] << " ";

    }

    file.close();

}

 //1. Interchange Sort

 void InterchangeSort(int arr[], int n){

    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]);

            }

        }

    }

 }

 //2. Selection Sort

 void SelectionSort(int arr[], int n){

    for(int i=0; i < n-1;i++){

        int Min = i; // lưu vị trí nhỏ nhất chưa sắp xếp

        for(int j=i+1; j<n; j++){

            if(arr[j] < arr[Min]){

                Min = j;

            }

        }

        swap(arr[i], arr[Min]);

    }

 }

 //3. Insertion Sort

 void InsertionSort(int arr[], int n){

    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. Bubble Sort

 void BubbleSort(int arr[], int n){

    for(int i=0; i<n-1; i++){

        for(int j= 0; j<n-i-1; j++){

            if(arr[j]>arr[j+1]){

                swap(arr[j], arr[j+1]);

            }

        }

    }

 }

//5.Heap sort

void Heapify(int arr[],int n,int i)

{

    int left=2\*i+1;

    int right=2\*i+2;

    int largest=i;

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

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

    // Nếu largest thay đổi, hoán đổi và đệ quy

    if (largest != i)

    {

        swap(arr[i], arr[largest]);

        Heapify(arr, n, largest);

    }

}

    //6.Ham sap xep Heap

void HeapSort(int arr[],int n)

{

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

    //doi cho node dau tien voi node cuoi cung

    for(int i=n-1;i>=0;i--)

    {

        swap(arr[0],arr[i]);

        Heapify(arr,i,0);

    }

}

    //ham

void Merge(int arr[], int low, int mid, int high) {

    int n1 = mid - low + 1;   // Kích thước mảng con bên trái

    int n2 = high - mid;      // Kích thước mảng con bên phải

    int left[n1], right[n2];  // Tạo mảng tạm để lưu hai mảng con

    // Sao chép phần tử vào mảng tạm left[]

    for (int i = 0; i < n1; i++)

        left[i] = arr[low + i];

    // Sao chép phần tử vào mảng tạm right[]

    for (int j = 0; j < n2; j++)

        right[j] = arr[mid + 1 + j];

    int i = 0, j = 0, k = low;  // Chỉ số cho mảng left, right và mảng gốc arr

    // Trộn hai mảng con lại theo thứ tự tăng dần

    while (i < n1 && j < n2) {

        if (left[i] <= right[j]) {  // Chọn phần tử nhỏ hơn để đưa vào arr[k]

            arr[k] = left[i];

            i++;

        } else {

            arr[k] = right[j];

            j++;

        }

        k++;

    }

    // Sao chép các phần tử còn lại của mảng left[] nếu có

    while (i < n1) {

        arr[k] = left[i];

        i++;

        k++;

    }

    // Sao chép các phần tử còn lại của mảng right[] nếu có

    while (j < n2) {

        arr[k] = right[j];

        j++;

        k++;

    }

}

    //6.Ham sap xep Merge

void MergeSort(int arr[], int low, int high) {

    if (low < high) {

        int mid = (low + high) / 2;

        MergeSort(arr, low, mid);

        MergeSort(arr, mid + 1, high);

        Merge(arr, low, mid, high);

    }

}

//Phan vung mang

int Partition(int arr[], int low, int high)

{

    int pivot = arr[high]; // Chọn phần tử cuối làm pivot

    int i = low - 1; // Chỉ số của phần tử nhỏ hơn pivot

    for (int j = low; j < high; j++)

    {

        if (arr[j] < pivot)

        {

            i++;

            swap(arr[i], arr[j]);

        }

    }

    swap(arr[i + 1], arr[high]); // Đưa pivot về đúng vị trí

    return i + 1; // Trả về vị trí mới của pivot

}

//7.Ham sap xep QuickSort

void QuickSort(int arr[], int low, int high)

{

    if (low < high)

    {

        int pivotIndex = Partition(arr, low, high);

        QuickSort(arr, low, pivotIndex - 1);

        QuickSort(arr, pivotIndex + 1, high);

    }

}

//Hàm đo thời gian chạy

void MeasureTime(int size)

{

    int\* arr=new int[30000];

    readArrayFromFile("input.txt", arr, size); // Đọc dữ liệu gốc từ file

    clock\_t start, end;

    double  time\_use\_Heap,time\_use\_Merge,time\_use\_Quick,time\_use\_Interchange,time\_use\_Bubble,time\_use\_Selection,time\_use\_Insertion;

    //Đo thời gian thực hiện heap sort

    int \*arr\_Heap=new int [size];//cấp phát mảng động

    //Cấp phát mảng ngẫu nhiên

    copy(arr,arr+size,arr\_Heap);// Sao chép nội dung của arr vào arr\_heap để đảm bảo cùng dữ liệu ban đầu

    start=clock();//bắt đầu đo thời gian

    HeapSort(arr\_Heap,size);//gọi hàm Heáport để sắp xếp mảng arr\_Heap

    end =clock();//kết thúc đo thời gian

    time\_use\_Heap=(double)(end-start)/CLOCKS\_PER\_SEC;//tính thời gian chạy đổi sang giây

    delete[] arr\_Heap;//giải phóng bộ nhớ

    //Đo thời gian thực hiện merge sort

    int \*arr\_Merge=new int [size];

    copy(arr,arr+size,arr\_Merge);

    start=clock();

    MergeSort(arr\_Merge,0,size-1);

    end=clock();

    time\_use\_Merge=(double)(end-start)/CLOCKS\_PER\_SEC;

    delete[] arr\_Merge;

    //Đo thời gian thực hiện quick sort

    int \* arr\_Quick=new int [size];

    copy(arr,arr+size,arr\_Quick);

    start=clock();

    QuickSort(arr\_Quick,0,size-1);

    end=clock();

    time\_use\_Quick=(double)(end-start)/CLOCKS\_PER\_SEC;

    delete[] arr\_Quick;

     //Đo thời gian thực hiện interchange sort

    int \* arr\_Interchange=new int [size];

    copy(arr,arr+size,arr\_Interchange);

    start=clock();

    InterchangeSort(arr\_Interchange,size);

    end=clock();

    time\_use\_Interchange=(double)(end-start)/CLOCKS\_PER\_SEC;

    delete[] arr\_Interchange;

     //Đo thời gian thực hiện bubble sort

    int \* arr\_Bubble=new int [size];

    copy(arr,arr+size,arr\_Bubble);

    start=clock();

    BubbleSort(arr\_Bubble,size);

    end=clock();

    time\_use\_Bubble=(double)(end-start)/CLOCKS\_PER\_SEC;

    delete[] arr\_Bubble;

    //Đo thời gian thực hiện selection sort

    int \* arr\_Selection=new int [size];

    copy(arr,arr+size,arr\_Selection);

    start=clock();

    SelectionSort(arr\_Selection,size);

    end=clock();

    time\_use\_Selection=(double)(end-start)/CLOCKS\_PER\_SEC;

    delete[] arr\_Selection;

    //Đo thời gian thực hiện insertion sort

    int \* arr\_Insertion=new int [size];

    copy(arr,arr+size,arr\_Insertion);

    start=clock();

    InsertionSort(arr\_Insertion,size);

    end=clock();

    time\_use\_Insertion=(double)(end-start)/CLOCKS\_PER\_SEC;

    delete[] arr\_Insertion;

    //Xuat thoi gian thuc hien

        cout<<"Size: "<<size<<endl;

    cout<<"Thoi gian thuc hien HeappSort: "<<time\_use\_Heap<<"s"<<endl;

    cout<<"Thoi gian thuc hien MergeSort: "<<time\_use\_Merge<<"s"<<endl;

    cout<<"Thoi gian thuc hien QuickSort: "<<time\_use\_Quick<<"s"<<endl;

    cout<<"Thoi gian thuc hien InterchangeSort: "<<time\_use\_Interchange<<"s"<<endl;

    cout<<"Thoi gian thuc hien BubbleSort: "<<time\_use\_Bubble<<"s"<<endl;

    cout<<"Thoi gian thuc hien SelectionSort: "<<time\_use\_Selection<<"s"<<endl;

    cout<<"Thoi gian thuc hien InsertionSort: "<<time\_use\_Insertion<<"s"<<endl;

    delete[] arr;

}

int main()

{

    // taoj kich thuoc cho mang ngau nhien

    cout<<"Đang chờ thời gian chạy ..."<<endl;

    int size;

    MeasureTime(size);

    return 0;

}

## File tạo txt:

#include <iostream>

#include <fstream>

#include <cstdlib>

#include <ctime>

using namespace std;

void generateDataFile(const char\* filename, int size) {

    ofstream file(filename);

    if (!file) {

        cout << "Không thể tạo file!\n";

        return;

    }

    file << size << endl;  // Ghi kích thước mảng vào file

    srand(time(0));

    for (int i = 0; i < size; i++) {

        file << rand() % (size \* 10) << " "; // Số ngẫu nhiên từ 0 -> size\*10

    }

    file.close();

    cout << "Đã tạo file " << filename << " thành công!\n";

}

int main() {

    generateDataFile("input.txt", 30000);

    return 0;

}

Test Key: nằm trong file câu 4 trong tập tin bài tập cơ sở

# Bài Tập Ứng Dụng

## Bài 1 ( Đức Thịnh):

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

void bubbleSort(vector<int>& arr) {

    int n = arr.size();

    for (int i = 0; i < n - 1; i++) {

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

            if (arr[j] > arr[j + 1]) {

                swap(arr[j], arr[j + 1]);

            }

        }

    }

}

void selectionSort(vector<int>& arr) {

    int n = arr.size();

    for (int i = 0; i < n - 1; i++) {

        int minIdx = i;

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

            if (arr[j] < arr[minIdx]) {

                minIdx = j;

            }

        }

        swap(arr[i], arr[minIdx]);

    }

}

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;

    }

}

int main() {

    vector<int> arr = {39, 8, 5, 1, 3, 6, 9, 12, 4, 7, 10};

    vector<int> arrCopy;

    // Bubble Sort

    arrCopy = arr; // Copy the original array

    bubbleSort(arrCopy);

    cout << "Bubble Sort: ";

    for (int num : arrCopy) {

        cout << num << " ";

    }

    cout << endl;

    // Selection Sort

    arrCopy = arr; // Copy the original array

    selectionSort(arrCopy);

    cout << "Selection Sort: ";

    for (int num : arrCopy) {

        cout << num << " ";

    }

    cout << endl;

    // Insertion Sort

    arrCopy = arr; // Copy the original array

    insertionSort(arrCopy);

    cout << "Insertion Sort: ";

    for (int num : arrCopy) {

        cout << num << " ";

    }

    cout << endl;

    return 0;

}

Test key Input: 28, 33, 71, 14, 96, 5, 47, 22, 63

Output:

a. Vị trí 3 phần tử lớn nhất (k = 3): 71 (vị trí 2), 96 (vị trí 4), 47 (vị trí 6)

b. Sắp xếp theo tổng chữ số (tăng dần): 5 (5), 14 (1+4=5), 22 (2+2=4), 33 (3+3=6), 71 (7+1=8), 28 (2+8=10), 47 (4+7=11), 63 (6+3=9), 96 (9+6=15)

Kết quả: 22, 5, 14, 33, 71, 63, 28, 47, 96

c. Xóa số nguyên tố (5, 47, 71 là nguyên tố): 28, 33, 14, 96, 22, 63

## Bài 2 ( Tấn Phát):

#include <iostream>

using namespace std;

// Định nghĩa struct SoHang để lưu số hạng của đa thức

struct SoHang {

    double heSo; // Hệ số

    int bac;     // Bậc

};

// Hàm hoán vị hai số hạng (đổi chỗ)

void swap(SoHang &a, SoHang &b) {

    SoHang temp = a;

    a = b;

    b = temp;

}

// Hàm sắp xếp theo bậc tăng dần (Interchange Sort)

void interchangeSort(SoHang arr[], int n) {

    for (int i = 0; i < n - 1; i++) {

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

            if (arr[i].bac > arr[j].bac) {

                swap(arr[i], arr[j]);

            }

        }

    }

}

int main() {

    // Mảng chứa các số hạng của đa thức

    SoHang daThuc[] = {

        {3.5, 2}, {1.2, 5}, {4.0, 3}, {2.1, 0}, {5.6, 1}

    };

    int size = sizeof(daThuc) / sizeof(daThuc[0]);

    // Gọi hàm sắp xếp

    interchangeSort(daThuc, size);

    // In kết quả sau khi sắp xếp

    cout << "Da sap xep theo bac:\n";

    for (int i = 0; i < size; i++) {

        cout << daThuc[i].heSo << "x^" << daThuc[i].bac << endl;

    }

    return 0;

}

- Câu a:

- Định nghĩa cấu trúc dữ liệu Để lưu trữ thông tin về mỗi số hạng của dãy thức bậc n trong bộ nhớ máy tính, ta sử dụng một cấu trúc dữ liệu bao gồm:

Hệ số (coefficient): Là một số thực (kiểu double), biểu diễn giá trị của số hạng.

Bậc (exponent): Là một số nguyên (kiểu int), có giá trị từ 0 đến 100, biểu diễn số mũ của biến trong số hạng.

Cấu trúc này có thể được khai báo bằng struct trong C++, giúp quản lý dữ liệu dễ dàng và có thể sử dụng trong các thao tác như sắp xếp, tính toán.

Test Key: Input: 2.7x^4, 5.1x^1, 0.9x^6, 3.2x^3, 1.8x^0 Output: Sắp xếp tăng dần theo bậc: 1.8x^0, 5.1x^1, 3.2x^3, 2.7x^4, 0.9x^6

## Bài 3 (Quang Vinh):

#include<iostream>

using namespace std;

struct PhongThi{

    int SoPhong;//1-200

    char Nha;//A-Z

    int KhaNangChua;//10-250

};

//Ham in thong tin phong thi

void inPhongThi(PhongThi pt){

    cout<<"So phong: "<<pt.SoPhong<<endl;

    cout<<"Nha: "<<pt.Nha<<endl;

    cout<<"Kha nang chua: "<<pt.KhaNangChua<<endl;

}

//Ham heapìy cho SoPhong thi

void HeapifySoPhong (PhongThi arr[],int n,int i)

{

    int left=2\*i+1;

    int right=2\*i+2;

    int largest=i;

    if(left<n && arr[left].SoPhong>arr[largest].SoPhong) largest=left;

    if(right<n && arr[right].SoPhong >arr[largest].SoPhong) largest=right;

    if(largest !=i){

        swap(arr[i],arr[largest]);

        HeapifySoPhong(arr,n,largest);

    }

}

void HeapSortSoPhong(PhongThi arr[],int n)

{

    for(int i=n/2 +1;i>=0;i--) HeapifySoPhong(arr,n,i);

    for(int i=n-1;i>0;i--)

    {

        swap(arr[0],arr[i]);

        HeapifySoPhong(arr,i,0);

    }

}

//Ham heapify cho Nha

void HeapifyNha (PhongThi arr[],int n,int i)

{

    int left=2\*i+1;

    int right=2\*i+2;

    int smallest=i;

    if(left<n && (arr[left].Nha<arr[smallest].Nha || (arr[left].Nha==arr[smallest].Nha && arr[left].SoPhong<arr[smallest].SoPhong))) smallest=left;

    if(right<n && (arr[right].Nha<arr[smallest].Nha || (arr[right].Nha==arr[smallest].Nha && arr[right].SoPhong<arr[smallest].SoPhong))) smallest=right;

    if(smallest !=i)

    {

        swap(arr[i],arr[smallest]);

        HeapifyNha(arr,n,smallest);

    }

}

//Ham heapSort cho Nha

void HeapSortNha(PhongThi arr[],int n)

{

    for(int i=n/2 +1;i>=0;i--)  HeapifyNha(arr,n,i);

    for(int i=n-1;i>0;i--)

    {

        swap(arr[0],arr[i]);

        HeapifyNha(arr,0,i);

    }

}

//Ham heapify cho KhaNangChua

void HeapifyKhaNangChua(PhongThi arr[], int n, int i) {

    int largest = i;

    int left = 2 \* i + 1;

    int right = 2 \* i + 2;

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

        largest = left;

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

        largest = right;

    if (largest != i) {

        swap(arr[i], arr[largest]);

        HeapifyKhaNangChua(arr, n, largest);

    }

}

//Ham heapSort cho KhaNangChua

void HeapSortKhaNangChua(PhongThi arr[], int n) {

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

        HeapifyKhaNangChua(arr, n, i);

    for (int i = n - 1; i > 0; i--) {

        swap(arr[0], arr[i]);

        HeapifyKhaNangChua(arr, i, 0);

    }

}

int main()

{

    // Dữ liệu mẫu

    PhongThi rooms[] = {

        {1, 'A', 50},

        {2, 'B', 100},

        {3, 'A', 75},

        {4, 'C', 25},

        {5, 'B', 150}

    };

    int n = 5;

    // 1. Sắp xếp giảm dần theo khả năng chứa

    cout << "Sap xep giam dan theo kha nang chua:\n";

    HeapSortKhaNangChua(rooms, n);

    for (int i = 0; i < n; i++)

        inPhongThi(rooms[i]);

    cout << endl;

    // Reset mảng về trạng thái ban đầu

    PhongThi rooms2[] = {

        {1, 'A', 50},

        {2, 'B', 100},

        {3, 'A', 75},

        {4, 'C', 25},

        {5, 'B', 150}

    };

    // 2. Sắp xếp tăng dần theo Nhà, rồi Số phòng

    cout << "Sap xep tang dan theo Nha, So phong:\n";

    HeapSortSoPhong(rooms2, n);

    for (int i = 0; i < n; i++)

        inPhongThi(rooms2[i]);

    cout << endl;

    // Reset mảng về trạng thái ban đầu

    PhongThi rooms3[] = {

        {1, 'A', 50},

        {2, 'B', 100},

        {3, 'A', 75},

        {4, 'C', 25},

        {5, 'B', 150}

    };

    // 3. Sắp xếp tăng dần theo Nhà, giảm dần theo Khả năng chứa

    cout << "Sap xep tang dan theo Nha, giam dan theo kha nang chua:\n";

    HeapSortNha(rooms3, n);

    for (int i = 0; i < n; i++)

        inPhongThi(rooms3[i]);

    return 0;

}

Đề yêu cầu thông tin các phòng thi

Gồm:

+ Số phòng thi // co giá trị từ 1-200

+ Nhà // chữ cái in hoa từ A-Z

+ Khả năng chứa // số nguyên từ 0-100

Ý niệm:

- Ta sử dụng cấu trúc dữ liệu struct để lưu các thông tin ở trên

+Số phòng thi và Khả năng chứa có kiểu dữ liệu int

+Nhà có kiểu dữ liệu char

-Đề yêu cầu sắp xếp và in ra màn hình các phòng thi theo thứ tự

+Trong bài này ,em sử dụng thuật toán sắp xếp vun đống (HeapSort ) để tối ưu hóa tốc độ thuật toán

\*\*\*Tạo Heapify cho từng thông tin,sau đó tạo từng hàm HeapSort riêng lẻ để sắp xếp

Sử dụng Test Key chạy thử

// Dữ liệu mẫu

{1, 'A', 50},

{2, 'B', 100},

{3, 'A', 75},

{4, 'C', 25},

{5, 'B', 150}

Test Key: Input: 105, C, 80 201, A, 120 106, C, 30 150, B, 200

Output:

a. Sắp xếp giảm dần theo khả năng chứa: 150, B, 200 201, A, 120 105, C, 80 106, C, 30

b. Sắp xếp tăng dần theo nhà, rồi số phòng: 201, A, 120 150, B, 200 105, C, 80 106, C, 30

c. Sắp xếp tăng dần theo nhà, giảm dần theo khả năng chứa: 201, A, 120 150, B, 200 105, C, 80 106, C, 30

## Bài 4 ( Minh Hoàng) :

#include <iostream>

#include <math.h>

using namespace std;

struct TreeNode{ // dinh nghia cac node

     int data;

    TreeNode\* left; //tao con tro trai co dang TreeNode

    TreeNode\* right; //tao con tro phai co dang TreeNode

};

// Ham kiem tra so nguyen to

bool IsPrime(int n){

    if (n < 2) return false; // neu phan tu nho hon 2 thi khong phai so nguyen to

    for (int i = 2; i <= sqrt(n); i ++ ){ // cho chay tu 2 toi sqrt(n) de tim kiem n co chia het cho so nao hay khong

        if (n % i == 0) return false; //neu chia het thi khong phai so nguyen to

    }

    return true;

}

//ham sap xep mang

void SapXepMang(int temp[], int size) { //sap xep theo thuat toan bubble sort

    for (int i = 0; i < size - 1; i++) { // so sanh tung phan tu voi nhau

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

            if (temp[j] > temp[j + 1]) {

                swap(temp[j], temp[j + 1]); // hoan doi neu tha dieu kien

            }

        }

    }

}

TreeNode\* creatNode(int val){

    TreeNode\* Node = new TreeNode(); // tao ra 1 node moi va cap phat vung nho cho no

    Node ->data=val;

    Node->left=NULL; //con tro tro den ben trai

    Node->right=NULL;//con tro tro den ben phai

    return Node;

}

//Ham nhap cay

TreeNode\* Nhapcay(int a[], int left, int right){

    if (left > right) return NULL; // thoat ham neu vi tri ko hop le

    int mid = (left + right)/2;

    TreeNode\* root = creatNode(a[mid]); //tao 1 con tro co cau truc TreeNode va gan gia tri nut do la a[mid]

    // xay dung cay tu hai node con hai ben

    root -> left = Nhapcay(a, left, mid -1); //goi lai ham nhap cay de xay dung cay nhi phan ben trai

    root -> right = Nhapcay(a, mid + 1, right); //tuong tu xay cay nhi phan ben phai

    return root;

}

// tao ham duyet cay de dam bao cay dang dung thu tu

void DuyetCay(TreeNode\* root){

    if (root == NULL) return; //neu nhu  nut goc khong co gia tri thi thoat ham

    DuyetCay(root->left); //Duyet cac phan tu ben trai

    cout<< root->data<<" ";

    DuyetCay(root->right); // Duyet cac phan tu ben phai

}

//cau a: Tim so nguyen to lon nhat

int TimSNT\_MAX(TreeNode\* root){

    int solonnhat = 0; // cho so lon nhat la 0 va thay doi khi tim thay so lon hon

    while (root != NULL){

        if (IsPrime(root->data)){ //xac dinh coi cac du lieu cua nut co phai so nguyen to khong

            solonnhat = root->data; // neu phai thi gan gia tri so lon nhat bang gia tri cua nut do

        }

        root = root->right; //tiep tuc tim kiem tim kiem ben phai vi cac phan tu sau ben phai luon lon hon nut goc cua cay

    }

    return solonnhat;

}

//cau b: Ham tim dong co so nguyen to

void TimDongCoSNT(int temp[], int& d, TreeNode\* root) {

    if (root == NULL) return;

    if (IsPrime(root->data)) {  //xac dinh coi cac du lieu cua nut co phai so nguyen to khong

        temp[d++] = root->data; //luu du lieu cua nut do vao 1 mang tam

    }

    TimDongCoSNT(temp, d, root->left); //tiep tuc voi nhanh trai

    TimDongCoSNT(temp, d, root->right); // tiep tuc voi nhanh phai

}

void TimDong(int temp[], int m, int n, int a[][3]){

     cout<<"Dong co chua so nguyen to la: "<<endl;

    bool found = false; //dat 1 ham tim kiem de xac dinh co dong nao chua so nguyen to khong

    for (int i = 0; i < m; i++) {

        bool coso = false; // ham tim kiem so nguyen to

        for (int j = 0; j < n; j++) {

            for (int l = 0; l < m \* n; l++) {

                if (temp[l] == a[i][j]) { // Neu phan tu trong mang tam trung voi phan tu trong ma tran

                    coso = true; // tra ve ket qua la tim thay

                    break; //thoat ham de tranh lap lai

                }

            }

            if (coso) break; // thoat vong lap neu phat hien thay so nguyen to

        }

        if (coso) {

            cout << i << " "; // in ket qua la dong co chua so nugyen to

            found = true; // tra co ket qua la co dong co chua so nguyen to

        }

    }

    if (!found) { // neu ket qua tra ve khong thay thi in ra khong co dong nao chua so nguyen to

        cout << "Khong co dong nao chua so nguyen to.";

    }

    cout << endl;

}

//cau c: Tim nhung dong chi chua so nguyen to

void TimDongChiCoSNT(int a[][3], int m, int n){

    cout<<"Cac dong chi chua so nguyen to: "<<endl;

    for (int i = 0; i < m; i++){ // tao dieu kien chay tu 0 toi m-1 de kiem dong chi chua so nguyen to

        bool codong = true; // tao ham dung sai ban dau de xac dinh co dong nao chi chua so nguyen to khong

        for (int j = 0; j < n; j++){

            if (!IsPrime(a[i][j])) // neu cac phan tu cua dong do co phan tu nao khong phai so nguyen to thi thoat ham

                codong = false;

                break;

        }

        if (codong) // neu ham xac dinh dong chi chua so nguyen to tra ve gia tri dung thi in ra dong do

            cout<<i<<" ";

    }

    cout<<endl;

}

int main(){

    int a[2][3] ={

        {1,3,4},

        {2,5,7}

    };

    int m = 2, n= 3;

    int temp[m\*n]; //tao mang tam co kich thuoc = m \* n

    int d = 0;

    for (int i = 0; i < m; i++){ // ham sap xep mang 2 chieu thanh mang 1 chieu

        for (int j = 0; j < n; j ++){

            temp[d++] = a[i][j]; // luu  cac gia tri cua mang 2 chieu tuan tu vao mang 1 chieu

        }

    }

    SapXepMang(temp, m\*n); // goi lai ham sap xep mang de sap xep mang 1 chieu

    Nhapcay(temp,0, m\*n -1); //goi ham nhap cay de nhap cac gia tri tu mang da sap xep vao cac not tren cay

    TreeNode\* root = Nhapcay(temp, 0, m\*n -1);

    //cau a

    int caua = TimSNT\_MAX(root); //gan gia tri cua ham tim so nguyen to lon nhat

   //cau b

   TimDongCoSNT(temp, d, root); // goi ham tim dong co chua so nguyen to

   TimDong(temp,m,n,a); // goi ham tim dong de tim phan tu trong mang tam ung voi phan tu nao trong mang 2 chieu

   if (caua != 0){ // neu ket qua tra ve tim so nguyen to lon nhat khac 0 thi in gia tri do ra

   cout<<"So nguyen to lon nhat trong mang la: "<<caua<<endl;

   }

   else // va neu ham tra ve gia tri 0 thi in ra

   cout<<"Khong co so nguyen to trong mang!"<<endl;

   // ham duyet cay de dam bao cay dung thu tu

   if (root == NULL) {

    cout << "Cây rỗng, không thể duyệt\n"; // neu nut khong co gia tri tuc la cay rong

} else {

    cout << "Bắt đầu duyệt cây...\n"; // neu co gia tri thi in cac gia tri do ra

    DuyetCay(root);

}

   return 0;

}

Test Key Input: Ma trận 3x4: 21 13 29 8 6 17 31 11 5 19 14 23

Output:

a. Số nguyên tố lớn nhất: 31

b. Dòng chứa số nguyên tố: Dòng 0 (13, 29), Dòng 1 (17, 31, 11), Dòng 2 (5, 19, 23)

c. Dòng chỉ chứa số nguyên tố: Không có (mỗi dòng đều chứa số không nguyên tố).

## Bài 5 ( Đức Thịnh) :

#include <iostream>

using namespace std;

const int MAX\_M = 100, MAX\_N = 100;

 // Hàm tính tổng một dòng trong ma trận

int sum\_of\_row(int A[][MAX\_N], int row, int n) {

    int sum = 0;

    for (int j = 0; j < n; j++) {

        sum += A[row][j];

    }

    return sum;

}

void swap\_rows(int A[][MAX\_N], int row1, int row2, int n) {

    for (int j = 0; j < n; j++) {

        swap(A[row1][j], A[row2][j]);

    }

}

// Sắp xếp các dòng theo tổng giảm dần (Insertion Sort)

void sort\_rows\_by\_sum(int A[][MAX\_N], int m, int n) {

    for (int i = 1; i < m; i++) {

        int key\_sum = sum\_of\_row(A, i, n);

        int j = i - 1;

        while (j >= 0 && sum\_of\_row(A, j, n) < key\_sum) {

            swap\_rows(A, j, j + 1, n);

            j--;

        }

    }

}

void print\_matrix(int A[][MAX\_N], int m, int n) {

    for (int i = 0; i < m; i++) {

        for (int j = 0; j < n; j++) {

            cout << A[i][j] << " ";

        }

        cout << endl;

    }

}

int main() {

    int m = 3, n = 4;

    int A[MAX\_M][MAX\_N] = {

        {4, 2, 7, 1},

        {8, 3, 5, 6},

        {1, 9, 2, 3}

    };

    cout << "Ma trận ban đầu:\n";

    print\_matrix(A, m, n);

    // Sắp xếp dòng theo tổng giảm dần (Insertion Sort)

    sort\_rows\_by\_sum(A, m, n);

    cout << "Ma trận sau khi sắp xếp theo tổng dòng giảm dần:\n";

    print\_matrix(A, m, n);

    return 0;

}

Test Key: Input: Ma trận 3x4: 15 25 35 45 10 20 30 40 5 10 15 20

Output:

a. Dòng có tổng lớn nhất: Dòng 0 (15+25+35+45 = 120)

b. Sắp xếp dòng theo tổng giảm dần: 15 25 35 45 (tổng 120) 10 20 30 40 (tổng 100) 5 10 15 20 (tổng 50)

## Bài 6 (Tấn Phát):

#include <iostream>

using namespace std;

// Hàm hoán vị 2 số nguyên

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

    int temp = a;

    a = b;

    b = temp;

}

// Hàm Bubble Sort dùng để sắp xếp tăng hoặc giảm

void BubbleSort(int arr[], int n, bool SoChan) {

    for (int i = 0; i < n - 1; i++) {

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

            if (arr[j] == 0 || arr[j + 1] == 0) continue; // Giữ nguyên số 0

            if (SoChan) { // Sắp xếp số chẵn tăng dần

                if (arr[j] > arr[j + 1]) {

                    Swap(arr[j], arr[j + 1]);

                }

            } else { // Sắp xếp số lẻ giảm dần

                if (arr[j] < arr[j + 1]) {

                    Swap(arr[j], arr[j + 1]);

                }

            }

        }

    }

}

// Hàm sắp xếp chẵn tăng, lẻ giảm, giữ nguyên số 0

void sapXepChanLe(int arr[], int n) {

    int chan[n], le[n]; // Mảng chứa số chẵn và số lẻ

    int viTriChan[n], viTriLe[n]; // Lưu vị trí của số chẵn và số lẻ trong mảng gốc

    int soLuongChan = 0, soLuongLe = 0;

    // Phân loại số chẵn và số lẻ

    for (int i = 0; i < n; i++) {

        if (arr[i] % 2 == 0 && arr[i] != 0) { // Số chẵn

            chan[soLuongChan] = arr[i];

            viTriChan[soLuongChan] = i;

            soLuongChan++;

        } else if (arr[i] % 2 != 0) { // Số lẻ

            le[soLuongLe] = arr[i];

            viTriLe[soLuongLe] = i;

            soLuongLe++;

        }

    }

    // Sắp xếp số chẵn tăng dần

    BubbleSort(chan, soLuongChan, true);

    // Sắp xếp số lẻ giảm dần

    BubbleSort(le, soLuongLe, false);

    // Đưa số chẵn đã sắp xếp về vị trí cũ

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

        arr[viTriChan[i]] = chan[i];

    }

    // Đưa số lẻ đã sắp xếp về vị trí cũ

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

        arr[viTriLe[i]] = le[i];

    }

}

int main() {

    int n;

    cout << "Nhap so luong phan tu: ";

    cin >> n;

    int arr[n];

    cout << "Nhap cac phan tu cho mang: ";

    for (int i = 0; i < n; i++) {

        cin >> arr[i];

    }

    // Gọi hàm sắp xếp

    sapXepChanLe(arr, n);

    // In mảng sau khi sắp xếp

    cout << "Mang sau khi sap xep: ";

    for (int i = 0; i < n; i++) {

        cout << arr[i] << " ";

    }

    cout << endl;

    return 0;

}

Test Key: Input: 9, 0, 14, 3, 22, 7, 0, 18, 5, 12 Output: 9, 0, 12, 7, 14, 5, 0, 18, 3, 22 Ghi chú: Số chẵn tăng (12, 14, 18, 22), số lẻ giảm (9, 7, 5, 3), số 0 giữ nguyên.

## 

## Bài 7 ( Minh Hoàng):

#include <iostream>

using namespace std;

// Ham dua mot node ve vi tri dung trong heap

void XayCay(int a[], int n, int i) {

    int solonnhat = i;      // dat node goc la lon nhat

    int left = 2 \* i + 1; // gia tri node trai

    int right = 2 \* i + 2;// gia tri node phai

    // neu con trai lon hon node goc

    if (left < n && a[left] > a[solonnhat])

        solonnhat = left;

    // neu con phai lon hon node goc

    if (right < n && a[right] > a[solonnhat])

        solonnhat = right;

    // neu solonnhat thay doi, hoan doi va goi de quy

    if (solonnhat != i) {

        swap(a[i], a[solonnhat]);

        XayCay(a, n, solonnhat);

    }

}

// Ham xay cay giam dan

void HeapSort(int arr[], int n) {

    // tao max heap

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

        XayCay(arr, n, i);

    for (int i = n - 1; i > 0; i--) {

        swap(arr[0], arr[i]);  // Dua phan tu lon nhat len cuoi

        XayCay(arr, i, 0);    // Xay dung lai heap

    }

}

// Ham sap xep chan le

void SapXepChanLe(int arr[], int n) {

    int left = 0, right = n - 1; // cho 2 con tro dau va cuoi

    while (left < right) {

        while (left < right && arr[left] % 2 == 0) left++;  // tim so le ben trai

        while (left < right && arr[right] % 2 != 0) right--; // tim so chan ben phai

        if (left < right) swap(arr[left], arr[right]); // doi cho de dua chan ve dau, le ve cuoi

    }

}

int main() {

    int a[] = {12, 7, 9, 10, 5, 2, 8, 1, 3, 4};

    int n = sizeof(a) / sizeof(a[0]);

    // goi ham sap xep chan le

    SapXepChanLe(a, n);

    // tim vi tri dau tien cua so le

    int dem = 0;

    while (dem < n && a[dem] % 2 == 0){

        dem++;

    }

    // sap xep chan le bang heap sort

    HeapSort(a, dem);         // sap xep day so chan

    HeapSort(a + dem, n - dem); // sap xep day so le

    // in ket qua

    cout << "Mang sau khi sap xep: ";

    for (int i = 0; i < n; i++)

        cout << a[i] << " ";

    return 0;

}

Test Key: Input: 11, 24, 7, 16, 33, 8, 19, 10

Output: 24, 16, 8, 10, 11, 7, 33, 19

## Bài 8 (Quang Vinh):

#include <iostream>

using namespace std;

void Merge(int arr[],int B[], int low, int mid, int high) {

    int left\_array = mid - low + 1;   // Kích thước mảng con bên trái

    int right\_array = high - mid;      // Kích thước mảng con bên phải

    int left[left\_array], right[right\_array]; // tạo mảng tạm để lưu hai mảng con

    // Sao chép phần tử vào mảng tạm left[]

    for (int i = 0; i < left\_array; i++)

        left[i] = arr[low + i];

    // Sao chép phần tử vào mảng tạm right[]

    for (int j = 0; j < right\_array; j++)

        right[j] = arr[mid + 1 + j];

    int i = 0, j = 0, k = low;  // Chỉ số cho mảng left, right và mảng gốc arr

    int coutn =0;

    // Trộn hai mảng con lại

    while (i < left\_array && j < right\_array) {

        if (left[i] <= right[j]) {  // Chọn phần tử nhỏ hơn để đưa vào arr[k]

            arr[k] = left[i];

            B[left[i]-1] += coutn;// cập nhật số phần tử trrong phần tử left[i]

            i++;

        } else {

            arr[k] = right[j];

            coutn++;//right[j] nhỏ hơn left[i] ,tăng số phần tử lớn phía trước

            j++;

        }

        k++;

    }

    // Sao chép các phần tử còn lại của mảng left[] nếu có

    while (i < left\_array) {

        arr[k] = left[i];

        B[left[i]-1] += coutn;// cập nhật số phần tử nghịch thế còn lại

        i++;

        k++;

    }

    // Sao chép các phần tử còn lại của mảng right[] nếu có

    while (j < right\_array) {

        arr[k] = right[j];

        B[right[j]-1] += coutn;// cập nhật số phần tử nghịch thế còn lại

        j++;

        k++;

    }

}

void MergeSort(int arr[],int B[], int low, int high) {

    if (low < high) {

        int mid = (low + high) / 2;

        MergeSort(arr,B, low, mid);

        MergeSort(arr,B, mid + 1, high);

        Merge(arr,B, low, mid, high);

    }

}

int main() {

    int arr[1000],B[1000]={0};//khởi tạo mảng B chứa số phần tử nghịch thế n

    int array\_size;

    cout << "Nhap so phan tu cua mang: ";

    cin >> array\_size;

    cout << "Nhap cac phan tu cua mang: ";

    for (int i = 0; i < array\_size; i++) {

        cin >> arr[i];

    }

    // Gọi MergeSort để tìm mảng nghịch thế

    MergeSort(arr,B, 0, array\_size - 1);

    cout << "Mang nghich the B : ";

    for (int i = 0; i < array\_size; i++) {

        cout << B[i] << " ";

    }

    return 0;

}

Test Key: Input: 3, 7, 2, 9, 1, 6, 8, 4, 5

Output:

a. Mảng nghịch thế: 2, 2, 5, 0, 4, 1, 0, 1, 0

b. Từ mảng nghịch thế trên, hoán vị ban đầu: 3, 7, 2, 9, 1, 6, 8, 4, 5

# Bài 9:

#include <iostream>

#include <fstream>

#include <cstring>

using namespace std;

struct SinhVien {

    int ma\_so;

    char ho\_va\_dem[21];

    char ten[41];

    int ngay\_sinh;

    int thang\_sinh;

    int nam\_sinh;

    char phai[4];

    float diem\_tb;

};

void nhap\_danh\_sach\_sinh\_vien(SinhVien danh\_sach\_sv[], int& n, const char\* file\_input) {

    ifstream file(file\_input);

    if (!file) {

        cerr << "Không thể mở file input!" << endl;

        return;

    }

    n = 0;

    while (file >> danh\_sach\_sv[n].ma\_so) {

        file.ignore();

        file.getline(danh\_sach\_sv[n].ho\_va\_dem, 21);

        file.getline(danh\_sach\_sv[n].ten, 41);

        file >> danh\_sach\_sv[n].ngay\_sinh >> danh\_sach\_sv[n].thang\_sinh >> danh\_sach\_sv[n].nam\_sinh;

        file.ignore();

        file.getline(danh\_sach\_sv[n].phai, 4);

        file >> danh\_sach\_sv[n].diem\_tb;

        n++;

    }

    file.close();

}

void luu\_danh\_sach\_sinh\_vien(SinhVien danh\_sach\_sv[], int n, const char\* file\_name) {

    ofstream file(file\_name, ios::binary);

    file.write(reinterpret\_cast<const char\*>(danh\_sach\_sv), n \* sizeof(SinhVien));

    file.close();

}

void doc\_danh\_sach\_sinh\_vien(SinhVien danh\_sach\_sv[], int& n, const char\* file\_name) {

    ifstream file(file\_name, ios::binary);

    file.read(reinterpret\_cast<char\*>(danh\_sach\_sv), n \* sizeof(SinhVien));

    n = file.gcount() / sizeof(SinhVien);

    file.close();

}

void in\_danh\_sach\_sinh\_vien(SinhVien danh\_sach\_sv[], int n) {

    for (int i = 0; i < n; ++i) {

        cout << "Mã số: " << danh\_sach\_sv[i].ma\_so << ", Họ và đệm: " << danh\_sach\_sv[i].ho\_va\_dem << ", Tên: " << danh\_sach\_sv[i].ten

             << ", Ngày sinh: " << danh\_sach\_sv[i].ngay\_sinh << "/" << danh\_sach\_sv[i].thang\_sinh << "/" << danh\_sach\_sv[i].nam\_sinh

             << ", Phái: " << danh\_sach\_sv[i].phai << ", Điểm trung bình: " << danh\_sach\_sv[i].diem\_tb << endl;

    }

}

void sap\_xep\_theo\_ma\_so(SinhVien danh\_sach\_sv[], int n) {

    for (int i = 0; i < n - 1; ++i) {

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

            if (danh\_sach\_sv[i].ma\_so > danh\_sach\_sv[j].ma\_so) {

                swap(danh\_sach\_sv[i], danh\_sach\_sv[j]);

            }

        }

    }

}

void sap\_xep\_theo\_ten(SinhVien danh\_sach\_sv[], int n) {

    for (int i = 0; i < n - 1; ++i) {

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

            if (strcmp(danh\_sach\_sv[i].ten, danh\_sach\_sv[j].ten) > 0 ||

                (strcmp(danh\_sach\_sv[i].ten, danh\_sach\_sv[j].ten) == 0 && strcmp(danh\_sach\_sv[i].ho\_va\_dem, danh\_sach\_sv[j].ho\_va\_dem) > 0)) {

                swap(danh\_sach\_sv[i], danh\_sach\_sv[j]);

            }

        }

    }

}

void sap\_xep\_theo\_diem\_tb(SinhVien danh\_sach\_sv[], int n) {

    for (int i = 0; i < n - 1; ++i) {

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

            if (danh\_sach\_sv[i].diem\_tb < danh\_sach\_sv[j].diem\_tb) {

                swap(danh\_sach\_sv[i], danh\_sach\_sv[j]);

            }

        }

    }

}

void luu\_chi\_muc(SinhVien danh\_sach\_sv[], int n, const char\* file\_name) {

    ofstream file(file\_name, ios::binary);

    file.write(reinterpret\_cast<const char\*>(danh\_sach\_sv), n \* sizeof(SinhVien));

    file.close();

}

int main() {

    const int max\_sv = 100;

    SinhVien danh\_sach\_sv[max\_sv];

    int n;

    // Nhập danh sách sinh viên từ file input

    nhap\_danh\_sach\_sinh\_vien(danh\_sach\_sv, n, "input.txt");

    luu\_danh\_sach\_sinh\_vien(danh\_sach\_sv, n, "SINHVIEN.DAT");

    // Sắp xếp theo Mã sinh viên tăng dần và lưu chỉ mục

    sap\_xep\_theo\_ma\_so(danh\_sach\_sv, n);

    luu\_chi\_muc(danh\_sach\_sv, n, "SVMASO.IDX");

    // Sắp xếp theo Tên sinh viên từ A → Z và lưu chỉ mục

    sap\_xep\_theo\_ten(danh\_sach\_sv, n);

    luu\_chi\_muc(danh\_sach\_sv, n, "SVTH.IDX");

    // Sắp xếp theo Điểm trung bình giảm dần và lưu chỉ mục

    sap\_xep\_theo\_diem\_tb(danh\_sach\_sv, n);

    luu\_chi\_muc(danh\_sach\_sv, n, "SVDTB.IDX");

    // Đọc và in danh sách sinh viên theo từng chỉ mục

    cout << "Danh sách theo Mã sinh viên:" << endl;

    doc\_danh\_sach\_sinh\_vien(danh\_sach\_sv, n, "SVMASO.IDX");

    in\_danh\_sach\_sinh\_vien(danh\_sach\_sv, n);

    cout << "\nDanh sách theo Tên sinh viên:" << endl;

    doc\_danh\_sach\_sinh\_vien(danh\_sach\_sv, n, "SVTH.IDX");

    in\_danh\_sach\_sinh\_vien(danh\_sach\_sv, n);

    cout << "\nDanh sách theo Điểm trung bình:" << endl;

    doc\_danh\_sach\_sinh\_vien(danh\_sach\_sv, n, "SVDTB.IDX");

    in\_danh\_sach\_sinh\_vien(danh\_sach\_sv, n);

    return 0;

}

Test Key: Danh sách theo Mã sinh viên:

Mã số: 1, Họ và đệm: Nguyen Van, Tên: A, Ngày sinh: 1/1/2000, Phái: Nam, Điểm trung bình: 8.5

Mã số: 2, Họ và đệm: Tran Thi, Tên: B, Ngày sinh: 2/2/2001, Phái: Nu, Điểm trung bình: 9

Mã số: 3, Họ và đệm: Le Van, Tên: C, Ngày sinh: 3/3/2002, Phái: Nam, Điểm trung bình: 7.5

Mã số: 4, Họ và đệm: Pham Thi, Tên: D, Ngày sinh: 4/4/2003, Phái: Nu, Điểm trung bình: 8

Mã số: 5, Họ và đệm: Hoang Van, Tên: E, Ngày sinh: 5/5/2004, Phái: Nam, Điểm trung bình: 6.5

Mã số: 6, Họ và đệm: Nguyen Thi, Tên: F, Ngày sinh: 6/6/2005, Phái: Nu, Điểm trung bình: 9.5

Mã số: 7, Họ và đệm: Tran Van, Tên: G, Ngày sinh: 7/7/2006, Phái: Nam, Điểm trung bình: 7

Mã số: 8, Họ và đệm: Le Thi, Tên: H, Ngày sinh: 8/8/2007, Phái: Nu, Điểm trung bình: 8.8

Mã số: 9, Họ và đệm: Pham Van, Tên: I, Ngày sinh: 9/9/2008, Phái: Nam, Điểm trung bình: 6

Mã số: 10, Họ và đệm: Hoang Thi, Tên: J, Ngày sinh: 10/10/2009, Phái: Nu, Điểm trung bình: 7.8

Danh sách theo Tên sinh viên:

Mã số: 1, Họ và đệm: Nguyen Van, Tên: A, Ngày sinh: 1/1/2000, Phái: Nam, Điểm trung bình: 8.5

Mã số: 2, Họ và đệm: Tran Thi, Tên: B, Ngày sinh: 2/2/2001, Phái: Nu, Điểm trung bình: 9

Mã số: 3, Họ và đệm: Le Van, Tên: C, Ngày sinh: 3/3/2002, Phái: Nam, Điểm trung bình: 7.5

Mã số: 4, Họ và đệm: Pham Thi, Tên: D, Ngày sinh: 4/4/2003, Phái: Nu, Điểm trung bình: 8

Mã số: 5, Họ và đệm: Hoang Van, Tên: E, Ngày sinh: 5/5/2004, Phái: Nam, Điểm trung bình: 6.5

Mã số: 6, Họ và đệm: Nguyen Thi, Tên: F, Ngày sinh: 6/6/2005, Phái: Nu, Điểm trung bình: 9.5

Mã số: 7, Họ và đệm: Tran Van, Tên: G, Ngày sinh: 7/7/2006, Phái: Nam, Điểm trung bình: 7

Mã số: 8, Họ và đệm: Le Thi, Tên: H, Ngày sinh: 8/8/2007, Phái: Nu, Điểm trung bình: 8.8

Mã số: 9, Họ và đệm: Pham Van, Tên: I, Ngày sinh: 9/9/2008, Phái: Nam, Điểm trung bình: 6

Mã số: 10, Họ và đệm: Hoang Thi, Tên: J, Ngày sinh: 10/10/2009, Phái: Nu, Điểm trung bình: 7.8

Danh sách theo Điểm trung bình:

Mã số: 6, Họ và đệm: Nguyen Thi, Tên: F, Ngày sinh: 6/6/2005, Phái: Nu, Điểm trung bình: 9.5

Mã số: 2, Họ và đệm: Tran Thi, Tên: B, Ngày sinh: 2/2/2001, Phái: Nu, Điểm trung bình: 9

Mã số: 8, Họ và đệm: Le Thi, Tên: H, Ngày sinh: 8/8/2007, Phái: Nu, Điểm trung bình: 8.8

Mã số: 1, Họ và đệm: Nguyen Van, Tên: A, Ngày sinh: 1/1/2000, Phái: Nam, Điểm trung bình: 8.5

Mã số: 4, Họ và đệm: Pham Thi, Tên: D, Ngày sinh: 4/4/2003, Phái: Nu, Điểm trung bình: 8

Mã số: 10, Họ và đệm: Hoang Thi, Tên: J, Ngày sinh: 10/10/2009, Phái: Nu, Điểm trung bình: 7.8

Mã số: 3, Họ và đệm: Le Van, Tên: C, Ngày sinh: 3/3/2002, Phái: Nam, Điểm trung bình: 7.5

Mã số: 7, Họ và đệm: Tran Van, Tên: G, Ngày sinh: 7/7/2006, Phái: Nam, Điểm trung bình: 7

Mã số: 5, Họ và đệm: Hoang Van, Tên: E, Ngày sinh: 5/5/2004, Phái: Nam, Điểm trung bình: 6.5

Mã số: 9, Họ và đệm: Pham Van, Tên: I, Ngày sinh: 9/9/2008, Phái: Nam, Điểm trung bình: 6.