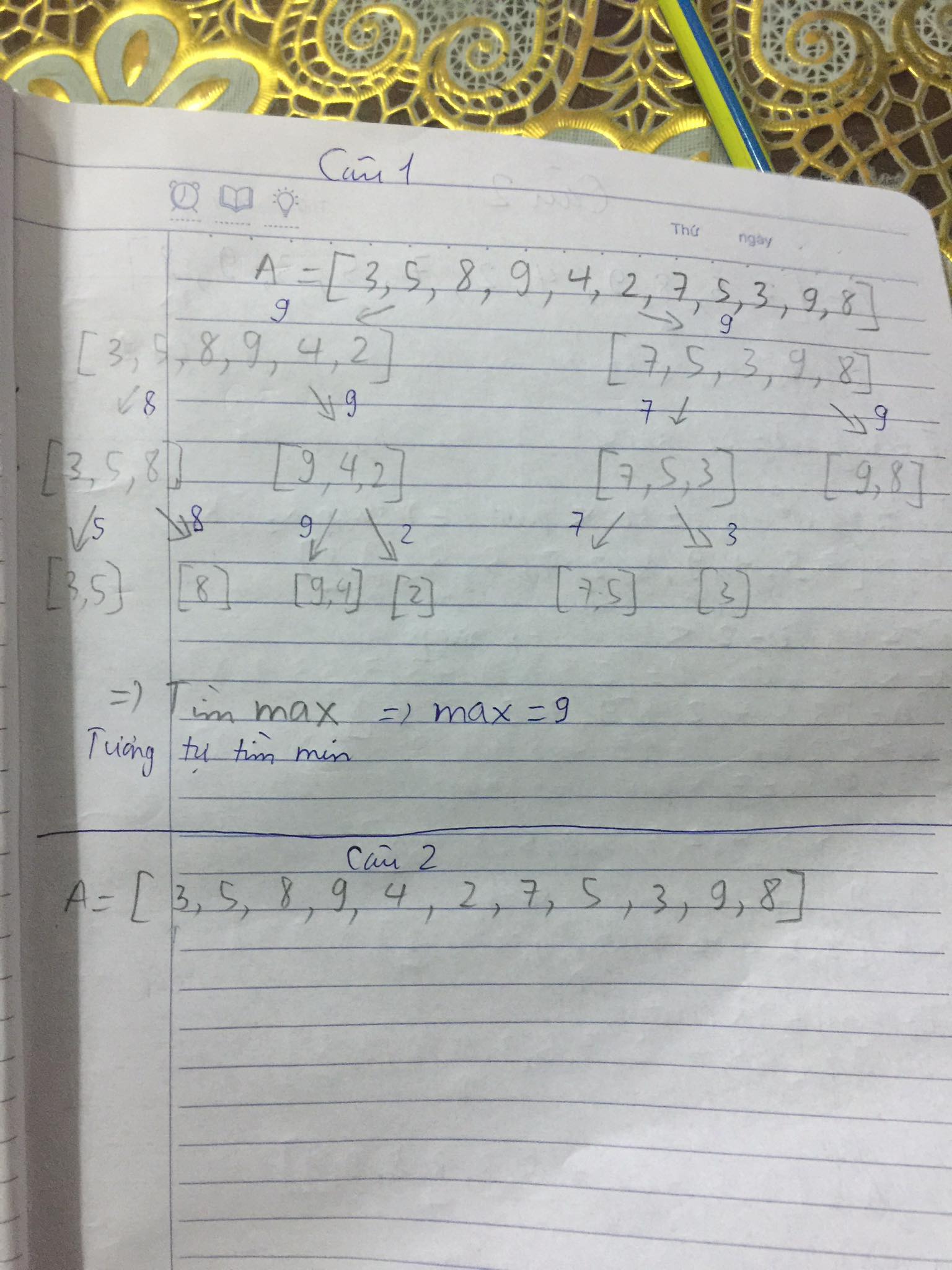
BT giải thuật

Bài 1: Min-Max



#include<stdio.h>

#include<conio.h>

void minmax(int a[], int dau, int cuoi, int &min, int &max){

int min1,min2,max1,max2;

if(dau==cuoi)

{

min=a[dau];

max=a[dau];

}

else

{

minmax(a, dau, (dau+cuoi)/2, min1, max1);

minmax(a, cuoi, (dau+cuoi)/2+1, min2, max2);

if(min1 < min2)

min=min1;

else

min=min2;

if(max1 > max2)

max=max1;

else

max=max2;

}

}

int main()

{

int a[11]={3,5,8,9,4,2,7,5,3,9,8};

int min=a[0];

int max=a[10];

minmax(a,0,10,min,max);

printf("\nphan tu nho nhat cua day la : %d",min);

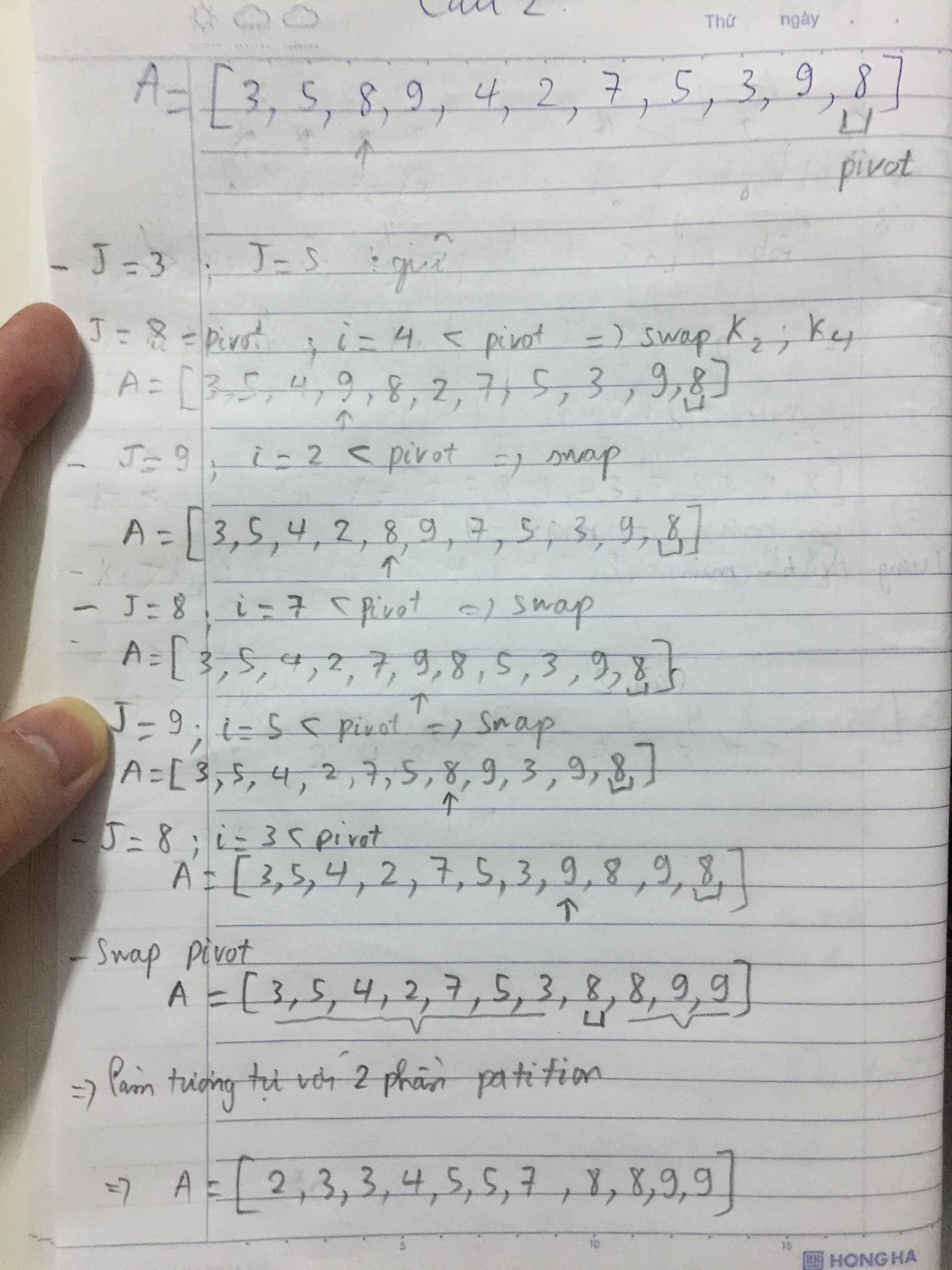
printf("\nphan tu lon nhat trong day la :%d",max);

getch();

return 0 ;

}

Bài 2: Quick sort



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

{

    int pivot = arr[high];    // pivot

    int left = low;

    int right = high - 1;

    while(true){

        while(left <= right && arr[left] < pivot) left++; // Tìm phần tử >= arr[pivot]

        while(right >= left && arr[right] > pivot) right--; // Tìm phần tử <= arr[pivot]

        if (left >= right) break; // Đã duyệt xong thì thoát vòng lặp

        swap(&arr[left], &arr[right]); // Nếu chưa xong, đổi chỗ.

        left++; // Vì left hiện tại đã xét, nên cần tăng

        right--; // Vì right hiện tại đã xét, nên cần giảm

    }

    swap(&arr[left], &arr[high]);

    return left; // Trả về chỉ số sẽ dùng để chia đổi mảng

}

/\* Hàm thực hiện giải thuật quick sort \*/

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

{

    if (low < high)

    {

        /\* pi là chỉ số nơi phần tử này đã đứng đúng vị trí

         và là phần tử chia mảng làm 2 mảng con trái & phải \*/

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

        // Gọi đệ quy sắp xếp 2 mảng con trái và phải

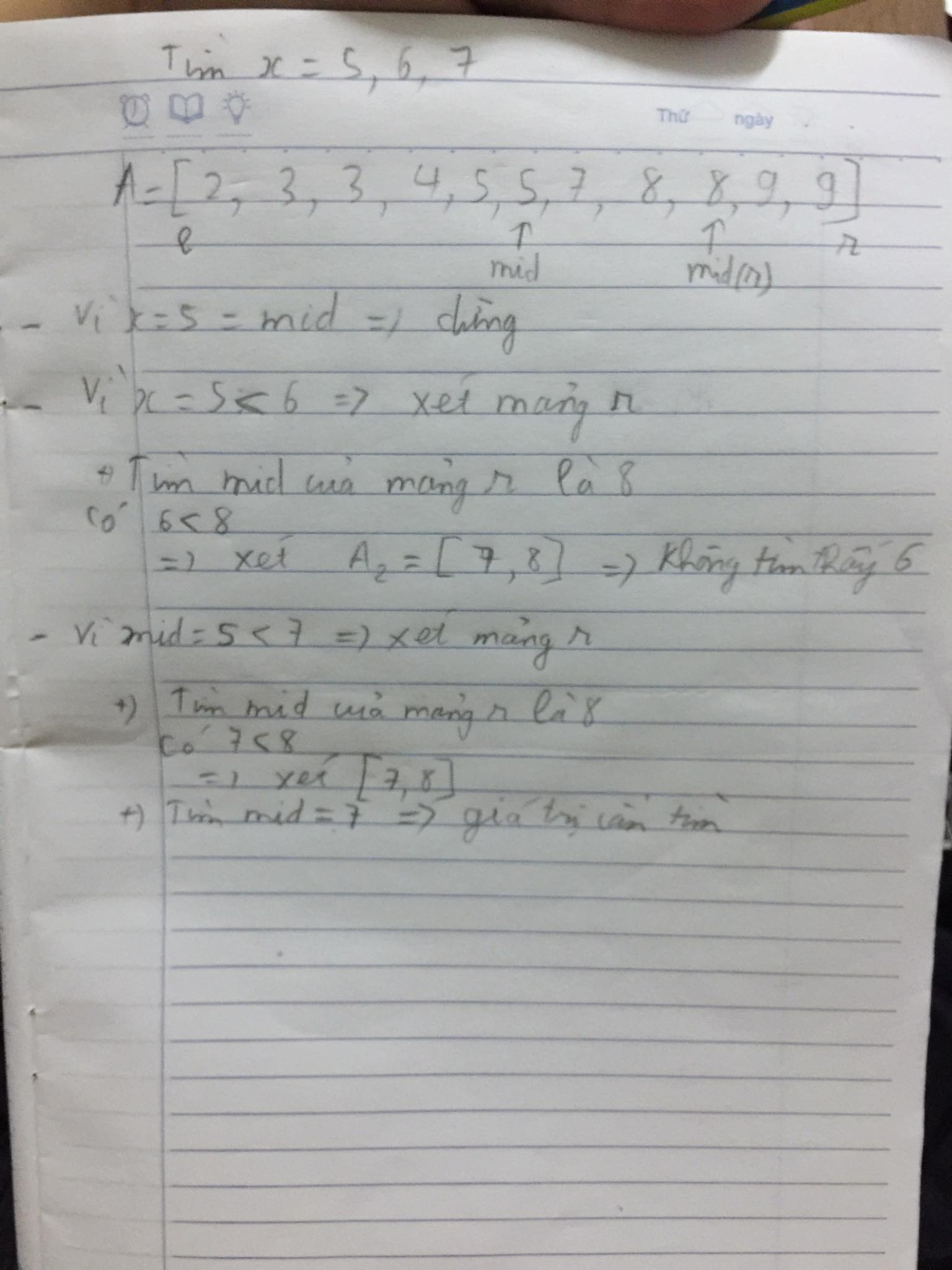
        quickSort(arr, low, pi - 1);

        quickSort(arr, pi + 1, high);

    }

}

Bài 3: tìm kiếm nhị phân



include <stdio.h>

// Hàm tìm kiếm nhị phân sử dụng giải thuật đệ quy

int binarySearch(int arr[], int l, int r, int x) {

  if (r >= l) {

    int mid = (l + r) / 2;

 // Nếu arr[mid] = x, trả về chỉ số và kết thúc.

    if (arr[mid] == x)

      return mid;

 // Nếu arr[mid] > x, thực hiện tìm kiếm nửa trái của mảng

    if (arr[mid] > x)

      return binarySearch(arr, l, mid - 1, x);

  // Nếu arr[mid] < x, thực hiện tìm kiếm nửa phải của mảng

    return binarySearch(arr, mid + 1, r, x);

  }

 // Nếu không tìm thấy

   return -1;

}

int main(void) {

  int arr[] = {2, 3,3,4,5,5,7,8,8,9,9};

// lấy ra độ dài của mảng

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

// giá trị cần tìm

  int x = 5;

// n-1 là vị trí cuối cùng trong mảng

  int result = binarySearch(arr, 0, n - 1, x);

  if (result == -1)

    printf("%d xuat hien tai chi so %d", x, result);

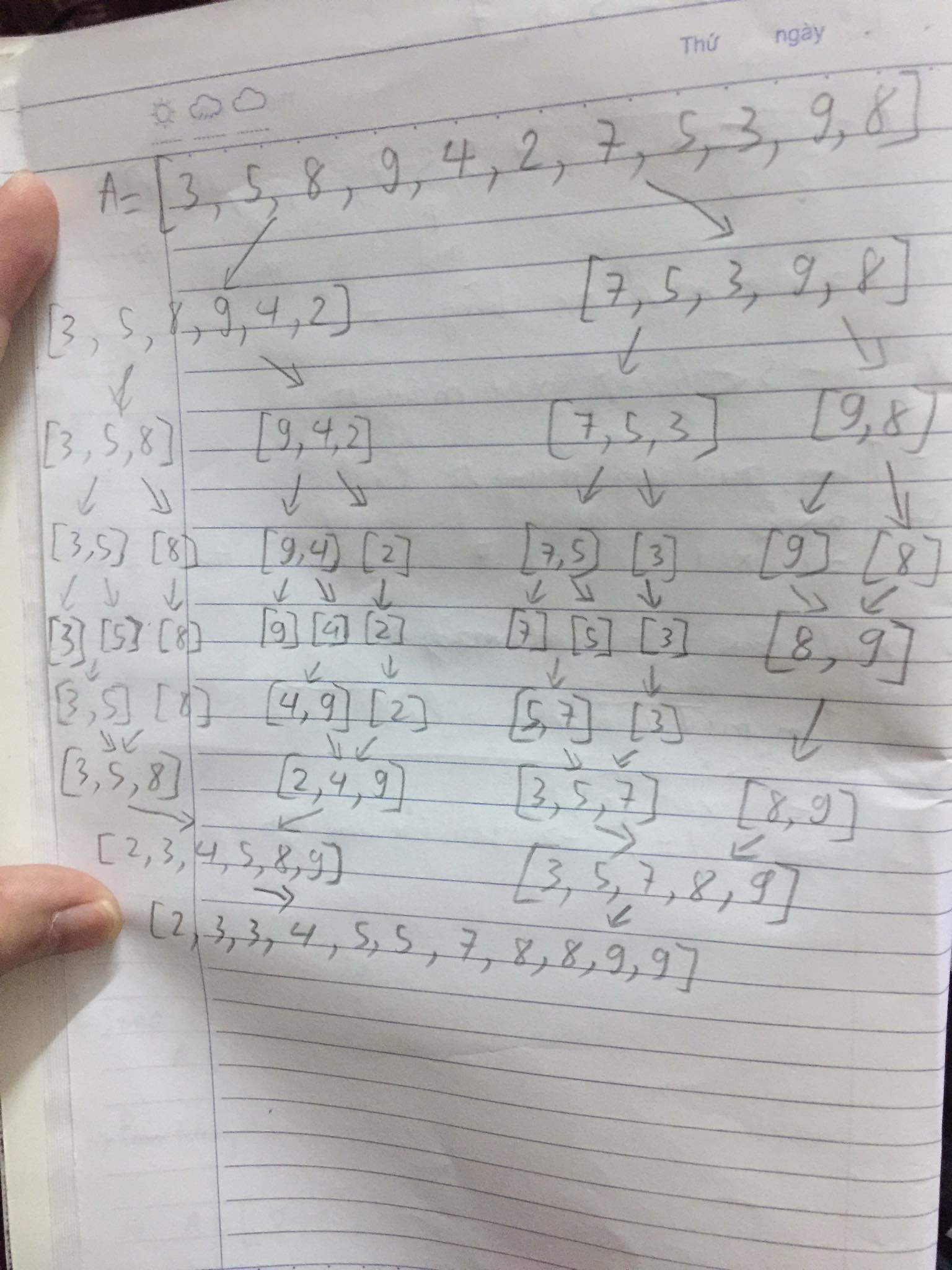
  else

    printf("%d xuat hien tai chi so %d", x, result);

  return 0;

}

Bài 4: merge sort



#include<stdlib.h>

#include<stdio.h>

void merge(int a[], int l, int mid, int r){

int n1 = mid - l + 1; // Số phần tử mảng con trái

// + 1 là do lưu thêm phần tử ở vị trí mid

int n2= r - mid; // Số phần tử mảng con phải

int left[n1]; int right[n2]; // Khai báo hai mảng trung gian

// Copy giữ liệu từ mảng chính ra hai mảng con

for(int x=0; x<n1; x++) left[x] = a[l+x];

for(int y=0; y<n2; y++) right[y] = a[mid+1+y];

int i=0, j=0; // Khai báo hai biến chạy để duyệt mảng con

int k=l; // Lưu k làm vị trí bắt đầu của mảng chính,

while(i<n1 && j<n2){ // Trong khi cả hai mảng con chưa hết phần tử

if(left[i]>=right[j]){ // Nếu phần tử mảng con trái >= mảng con phải

a[k]=right[j]; // Điển phần tử mảng con phải vào mảng chính

j++; // xét phần tử tiếp theo của mảng right

}

else{ // Ngược lại tức là left[i] < right[j]

a[k]=left[i];

i++;

}

k++; // Tăng index của mảng chính, mỗi lần lặp sẽ tìm được 1 phần tử thích hợp

}

// Sau vòng lặp trên, 1 trong 2 mảng đã duyệt hết phần tử, hoặc cả hai cùng hết

while(j<n2){ // Nếu mảng right chưa hết, mảng left đã hết

a[k]=right[j]; // Cho toàn bộ các phần tử còn lại vào mảng chính

k++;

j++;

}

while( i<n1 ){ // Nếu mảng left chưa hết, mảng right hết

a[k]= left[i];

k++;

i++;

}

}

void mergeSort(int a[], int l, int r){

int t; // biến để lưu vị trí chia đôi mảng

if(l<r){ // Nếu mảng còn ít nhất 1 phần tử

t=(l+r)/2; // Chia đôi mảng

mergeSort(a,l,t); // Đệ quy mảng trái

mergeSort(a,t+1,r); // Đệ quy mảng phải

merge(a,l,t,r); // Trộn hai mảng lại

}

else // Mảng < 1 phần tử sẽ dừng đệ quy

return;

}