TRƯỜNG ĐH CNTT & TT THÁI NGUYÊN

KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO THỰC TẬP CƠ SỞ

Đề tài: NGHIÊN CỨU CÁC THUẬT TOÁN SẮP XẾP THÔNG DỤNG VÀ CÀI ĐẶT MÔ PHỎNG THUẬT TOÁN SẮP XẾP NHANH

Giảng viên hướng dẫn: PHẠM THỊ THƯƠNG

Sinh viên thực hiện: DƯƠNG VĂN ĐỊNH **Lớp**: KTPMK17B

Mã SV: DTC1854801030012

Thái Nguyên 4/2021

MỤC LỤC

LỜI CẢM ƠN

Trước tiên với tình cảm sâu sắc và chân thành nhất, cho phép em được bày tỏ lòng biết ơn đến tất cả các Thầy cô và bạn bè đã tạo điều kiện hỗ trợ, giúp đỡ em trong suốt quá trình học tập và thực hiện đề tài này. Trong suốt thời gian từ khi bắt đầu học tập tại trường đến nay, em đã nhận được rất nhiều sự quan tâm, giúp đỡ của quý Thầy Cô và bạn bè.

Với lòng biết ơn sâu sắc nhất, em xin gửi đến quý Thầy Cô ở Khoa Công Nghệ Thông Tin, đặc biệt là cô giáo Phạm Thị Thương người đã trực tiếp hướng dẫn em. Cảm ơn Thầy Cô đã truyền đạt vốn kiến thức quý báu cho em trong suốt thời gian học tập tại trường vừa qua. Nhờ có những lời hướng dẫn, dạy bảo của các Thầy Cô nên đề tài thực tập cơ sở của em mới có thể hoàn thiện tốt đẹp.

Một lần nữa, em xin chân thành cảm ơn Thầy Cô người đã trực tiếp giúp đỡ, quan tâm, hướng dẫn em hoàn thành tốt bài báo cáo này trong thời gian qua.

Bài báo cáo thực tập thực hiện trong khoảng thời gian 8 tuần. Bước đầu đi vào thực tế của em còn hạn chế và còn nhiều bỡ ngỡ nên không tránh khỏi những thiếu sót , em rất mong nhận được những ý kiến đóng góp quý báu của quý Thầy Cô để kiến thức của em trong lĩnh vực này được hoàn thiện hơn đồng thời có điều kiện bổ sung, nâng cao ý thức của mình.

Em xin chân thành cảm ơn!

LỜI MỞ ĐẦU

1. Lý do chọn đề tài

Thuật toán là một trong những kiến thức cực kì cần thiết trong lập trình. Mọi chương trình, sản phẩm phần mềm nếu muốn thành công và có sức ảnh hưởng đều cần phải áp dụng đúng và chính xác các thuật toán.

Thuật toán sắp xếp *(sort algorithm)* là một trong top các thuật toán quan trọng và được sử dụng nhiều nhất. Đây cũng chính là thuật toán cơ bản mà mọi lập trình viên đều phải làm chủ trong bộ kĩ năng của mình. Ý thức được tầm quan trọng của thuật toán em đã chọn thuật

Được sự hướng dẫn của cô Phạm Thị Thương và vận dụng kiến thức đã học em đã thực hiện đề tài "**nghiên cứu các thuật toán sắp xếp thông dụng và cài đặt mô phỏng thuật toán sắp xếp nhanh**".

2. Mục đích nghiên cứu

- Nắm được 6 thuật toán sắp xếp thông dụng
- Xây dựng chương trình mô phỏng thuật toán sắp xếp nhanh (Qick Sort) bằng C#

3. Bố cục đề tài

Bài báo cáo gồm 3 chương:

- Chương I: Tổng quan về thuật toán sắp xếp
- Chương II: Một số thuật toán sắp xếp thông dụng
- Chương III: Cài đặt chương trình mô phỏng thuật toán sắp xếp nhanh bằng C#

CHƯƠNG I: TỔNG QUAN VỀ THUẬT TOÁN SẮP XẾP

1.1 Tầm quan trọng của các thuật toán sắp xếp

Trong khoa học máy tính và trong toán học, bài toán sắp xếp là một bài toán sắp xếp các phần tử của một danh sách theo thứ tự nhất định *(tăng dần, giảm dần).* Đầu vào của bài toán là danh sách chưa sắp xếp, đầu ra là một danh sách đã sắp xếp. Ta thường xét trường hợp các phần tử trong danh sách cần sắp xếp là các số và sắp xếp theo chiều tăng dần.

Ngay từ đầu của khoa học máy máy tính, bài toán sắp xếp với đã thu hút rất nhiều nghiên cứu của các nhà khoa học. Có thời điểm, đây là chủ đề mà được nhiều sự quan tâm nhất trong lĩnh vực công nghệ. Thuật toán giải quyết bài toán sắp xếp nêu trên được gọi là thuật toán sắp xếp – Sorting Algorithm.

Có rất nhiều lý do để đánh giá đây là một loại thuật toán cực kì quan trọng. Thuật toán sắp xếp là một trong số các thuật toán quan trọng nhất trong lập trình. Là thuật toán hiện tại vẫn được các lập trình viên, các nhà khoa học không ngừng nghiên cứu và phát triển để đưa ra giải thuật tốt nhất. Hiện nay đã có hơn 40 thuật toán sắp xếp được phát minh trên thế giới.

Các thuật toán sắp xếp ứng dụng thực tế rất nhiều trong các chương trình, ứng dụng. Hầu hết các sản phẩm phần mềm đều ứng dụng thuật toán này. Chúng có thể sử dụng để xử lý cơ sở dữ liệu, xây dựng các tính năng, modun cho phần mềm . . .

Một số ví dụ về các tính năng cần sử dụng thuật toán sắp xếp:

- Quản lý, sắp xếp các file, data dựa vào ngày tạo, dung lượng file, tên file
- Các dạng bảng xếp hạng
- Sắp xếp theo giá sản phẩm
- . . .

Đối với các nguồn dữ liệu đầu vào khác nhau thì lại cần phải áp dụng thuật toán sắp xếp phù hợp nhất để chương trình được tối ưu nhất. Thuật toán sắp xếp hoàn toàn có thể ảnh hưởng tới tốc độ của sản phẩm cũng như trải nghiệm người dùng.

Thuật toán sắp xếp cũng cực kì quan trọng đối với các lập trình viên mới bắt đầu học tập. Tìm hiểu thuật toán giúp cải thiện khả năng tư duy về giải thuật của bản thân đồng thời cũng là một kiến thức cần thiết cho công việc sau này.

1.2 Phân loại thuật toán sắp xếp

Thuật toán sắp xếp có thể phân loại theo nhiều cách khác nhau dựa trên tính chất và đặc điểm của thuật toán.

1.2.1 Phân loại theo độ phức tạp của thuật toán

Độ phức tạp tính toán hoặc đơn giản là độ phức tạp của thuật toán là lượng tài nguyên cần thiết để chạy nó. Độ phức tạp thường đánh giá dựa trên thời gian chạy và bộ nhớ sử dụng của thuật toán.

Dựa vào tính chất này, có thể chia thuật toán sắp xếp thành 3 loại:

- Độ phức tạp tốt O($n\ log\ n$): Heap sort, Merge sort, IntroSort . . .
- Độ phức tạp trung bình O($log^2 n$): Bitonic sorter, Sorting network. . .
- Độ phức tạp xấu O (n^2): Bubble sort, Insertion sort, Selection sort

Một số thuật toán có thể có độ phức tạp tốt ở mức O*(n)*. Tuy nhiên trường hợp này chỉ đúng với một số dữ liệu đầu vào và trong các điều kiện phần cứng nhất định. Ở đây ta phân loại dựa theo trung bình của độ phức tạp thuật toán.

1.2.2 Phân loại dựa theo có phải là một phương pháp so sánh hay không

Thuật toán được gọi là có sử dụng phương pháp so sánh nếu chúng chỉ sử dụng toán tử so sánh để kiểm tra giữa hai phần tử.

- Sắp xếp so sánh: Bubble Sort, Quicksort, Tree sort, Timsort, Shell sort . . .
- Sắp xếp không so sánh: Counting sort, Flashsort, Bucke tsort . . .

1.2.3 Phân loại dựa theo khả năng thích ứng

Khả năng thích ứng ở đây chính là thuật toán có bị ảnh hưởng bởi dữ liệu đầu vào hay không. Với mỗi dữ liệu đầu vào khác nhau thì tốc độ giải thuật có thể bị ảnh hưởng hoặc không. Đặc điểm này cũng giúp ta phân loại được thuật toán sắp xếp.

- Sắp xếp thích ứng: Merge sort, Heapsort, Intro sort . . .
- Sắp xếp không thích ứng: Bubble sort, Quicksort, . . .

1.2.4 Phân loại dựa theo tính ổn định của thuật toán

Một thuật toán sắp xếp được gọi là ổn định nếu hai phần tử trong dãy có các giá trị bằng nhau, sau khi sắp xếp thứ tự tương quan của chúng không thay đổi so với đầu vào. Tức phần tử nào đứng trước sau khi sắp xếp vẫn đứng trước.

- Sắp xếp ổn định: Bubble sort, Insertion sort . . .
- Sắp xếp không ổn định: Quicksort, Heapsort . . .

Các thuật toán sắp xếp không ổn định có thể được sửa đổi thành ổn định bằng

1.2.5 Phân loại dựa theo đặc điểm của thuật toán

Dựa vào đặc điểm của giái thuật có thể chia thành hai loại

- Sắp xếp đơn giản: Bubble sort, Selection sort

Sắp xếp nhanh: Quicksort, Merge sort, Heapsort, Counting sort, Radix sort . . .

Ngoài ra còn có thể phân loại theo phương pháp chung của thuật toán: chèn, trao đổi, hợp nhất, lựa chọn, sử dụng đệ quy . . .

CHƯƠNG 2: MỘT SỐ THUẬT TOÁN SẮP XẾP THÔNG DỤNG

2.1 Thuật toán sắp xếp nổi bọt – Bubble sort

2.1.1 Ý tưởng thuật toán

Thuật toán sắp xếp nổi bọt - Bubble Sort là một thuật toán thuộc loại sắp xếp dựa vào so sánh. Nó được đưa ra nghiên cứu lần đầu bởi Iverson vào năm 1962, đây là một thuật toán sắp xếp đơn giản dựa vào phương pháp so sánh. Nó được lấy ý tưởng nổi bọt của các hạt bong bóng, bong bóng lớn sẽ nổi lên trên bong bóng nhỏ.

- Thuật toán thực hiện duyệt từ đầu danh sách tới cuối danh sách. So sánh hai phần tử liền kề nhau, tiến hành đổi chỗ nếu phần từ đứng trước và phần tử đứng sau không theo thứ tự.
- Mỗi lần duyệt mảng sẽ đặt một phần tử vào đúng vị trí của nó, và ta sẽ không xét đến nó trong lần lặp tiếp theo. Số lượng phần tử cần duyệt giảm đi một lần sau mỗi lần lặp. Ở lần lặp thứ i, có i phần tử trong danh sách đã được sắp xếp đúng thứ tự.
- Dừng thuật toán cho đến khi mọi phần tử đều đúng vị trí

Với tư tưởng này, thuật toán sắp xếp nổi bọt có thể thực hiện duyệt mảng theo hai cách:

- Duyệt từ đầu tới cuối, mỗi lần duyệt phần tử lớn nhất còn lại sẽ nổi lên ở cuối danh sách
- Duyệt từ cuối về đầu, mỗi lần duyệt phần tử nhỏ nhất còn lại sẽ ở nổi đầu danh sách

2.1.2 Giải thuật Bubble sort

Thuật toán sử dụng cách duyệt từ đầu danh sách:

Input: Mảng số nguyên a có n phần tử

Output: Mảng a sắp xếp có thứ tự

Bước 1: i = 0; *// Đếm số lần duyệt*

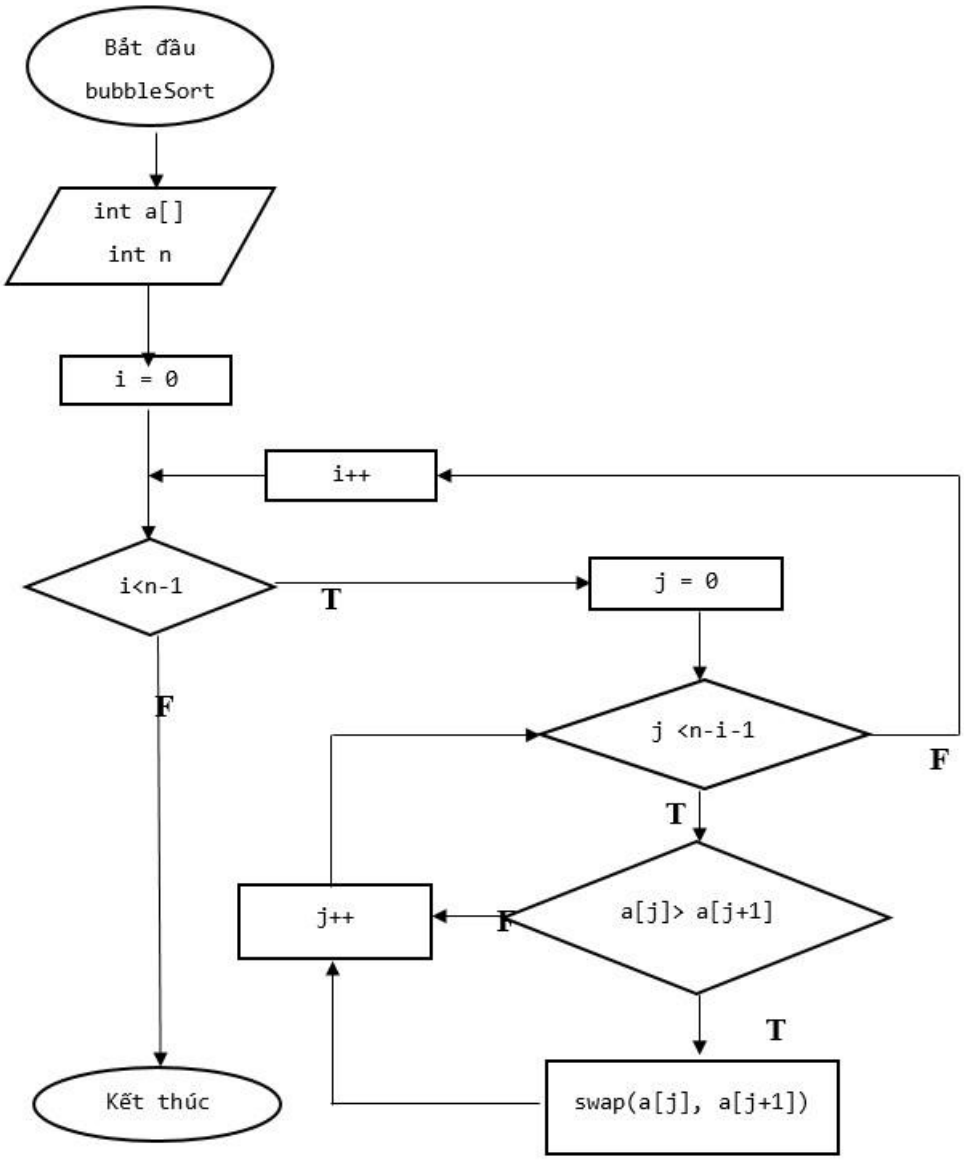
Bước 2: j = 0; *//Duyệt từ đầu mảng tới cuối*

- Trong khi (j < n- i -1) thực hiện: *//Mỗi lần duyệt, bỏ đi i phần tử không xét*
  - Nếu a[j]>a[j+1] thì đổi chỗ a[j], a[j+1]
  - j = j+1;

Bước 3: i = i+1; *// lần duyệt tiếp*

- Nếu i = n-1: Dừng *// Hết dãy*
- Ngược lại: Lặp lại Bước 2

Lưu đồ thuật toán:



2.1.3 Cài đặt thuật toán Bubble sort

Cài đặt thuật toán sắp xếp nổi bọt bằng C/C++:

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

Cách cài đặt cải tiến của Bubble sort giúp tối ưu số lần duyệt mảng:
Trong trường hợp chưa thực hiện n-1 lần duyệt mảng mà dãy đã sắp xếp. Sử dụng thêm biến **`bool swapped`** để kiểm tra xem danh sách còn tiếp tục đổi chỗ hay không. Nếu không đổi chỗ, tức là dãy đã sắp xếp.

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

2.1.4 Đánh giá thuật toán Bubble sort

Bảng đánh giá về độ phức tạp của thuật toán Bubble Sort:

Trường hợp	Số lần so sánh	Số lần đổi chỗ	Độ phức tạp	Bộ nhớ sử dụng
Tốt nhất	$\frac{n(n-1)}{2}$	0	O(n)	1
Xấu nhất	$\frac{n(n-1)}{2}$	$\frac{n(n-1)}{2}$	O(n^2)	O(n)

Trong trường hợp trung bình thuật toán vẫn có độ phức tạp là O(n^2).

Ưu điểm:

- Thuật toán đơn giản dễ hiểu, dễ dàng cài đặt, phù hợp cho người mới nghiên cứu lập trình
- Tốn ít bộ nhớ, là thuật toán có tính ổn định và thích ứng, không phụ thuộc nhiều vào dữ liệu đầu vào.

Nhược điểm:

- Có độ phức tạp lớn O(n^2) nên không thể áp dụng được với các bài toán phải xử lý lượng dữ liệu lớn.
- Không áp dụng nhiều trong thực tế.

2.2 Thuận toán sắp xếp chèn – Insertion sort

2.2.1 Ý tưởng thuật toán sắp xếp chèn

Thuật toán sắp xếp chèn (Insertion sort) được John Mauchly đưa ra rất sớm vào năm 1946 trong cuộc hội thảo đầu tiên về thuật toán sắp xếp trên máy tính. Đây là một thuật toán sắp xếp bắt chước cách sắp xếp quân bài của những người chơi bài. Muốn sắp một bộ bài theo trật tự người chơi bài rút lần lượt từ quân thứ 2, so với các quân đứng trước nó để chèn vào vị trí thích hợp.

Thuật toán sử dụng vòng lặp để duyệt từ đầu danh sách tới cuối danh sách và chèn phần tử vào đúng vị trí để phát triển thành danh sách có thứ tự đúng.

- Tại mỗi vị trí i của phần tử đang duyệt, lấy ra phần tử a[i] đó, tìm vị thích hợp trong đoạn đã xét a[0] đến a[i-1] sau đó chèn để đoạn danh sách a[0] đến a[i] trở thành danh sách có thứ tự đúng.
- Ở lần duyệt thứ i *(i bắt đầu từ i, bỏ qua phần tử đầu tiên),* ta có i+1 phần tử ở đầu danh sách là một danh sách có thứ tự.

Việc sắp xếp thường được thực hiện tại chỗ, bằng cách lặp lại mảng, tăng danh sách được sắp xếp đằng sau nó. Tại mỗi vị trí mảng, nó kiểm tra giá trị ở đó so với giá trị lớn nhất trong danh sách đã sắp xếp (tức là phần tử ở bên trái nó, danh sách trước nó đã được kiểm tra). Nếu lớn hơn, nó đứng tại chỗ và chuyển sang kiểm tra phần tử tiếp theo. Nếu nhỏ hơn, nó sẽ tìm vị trí chính xác trong danh sách đã sắp xếp, dịch chuyển tất cả các giá trị lớn hơn để tạo khoảng trắng và chèn vào vị trí chính xác đó.

Thuật toán sắp xếp chèn còn có một biến thể đó là chèn nhị phân (Binary Insertion sort). Ở biến thể này, thuật toán sẽ sử dụng phương pháp tìm kiếm nhị phân để tìm vị trí chèn phù hợp cho phần tử đang duyệt. Sau đó mới tiến hành chèn.

2.2.2 Giải thuật Insertion sort

Thuật toán Insertion Sort:

Input: Mảng số nguyên a có n phần tử

Output: Mảng a sắp xếp có thứ tự

Bước 1: i = 1; // Bỏ qua phần tử đầu tiên

Bước 2: j = i;

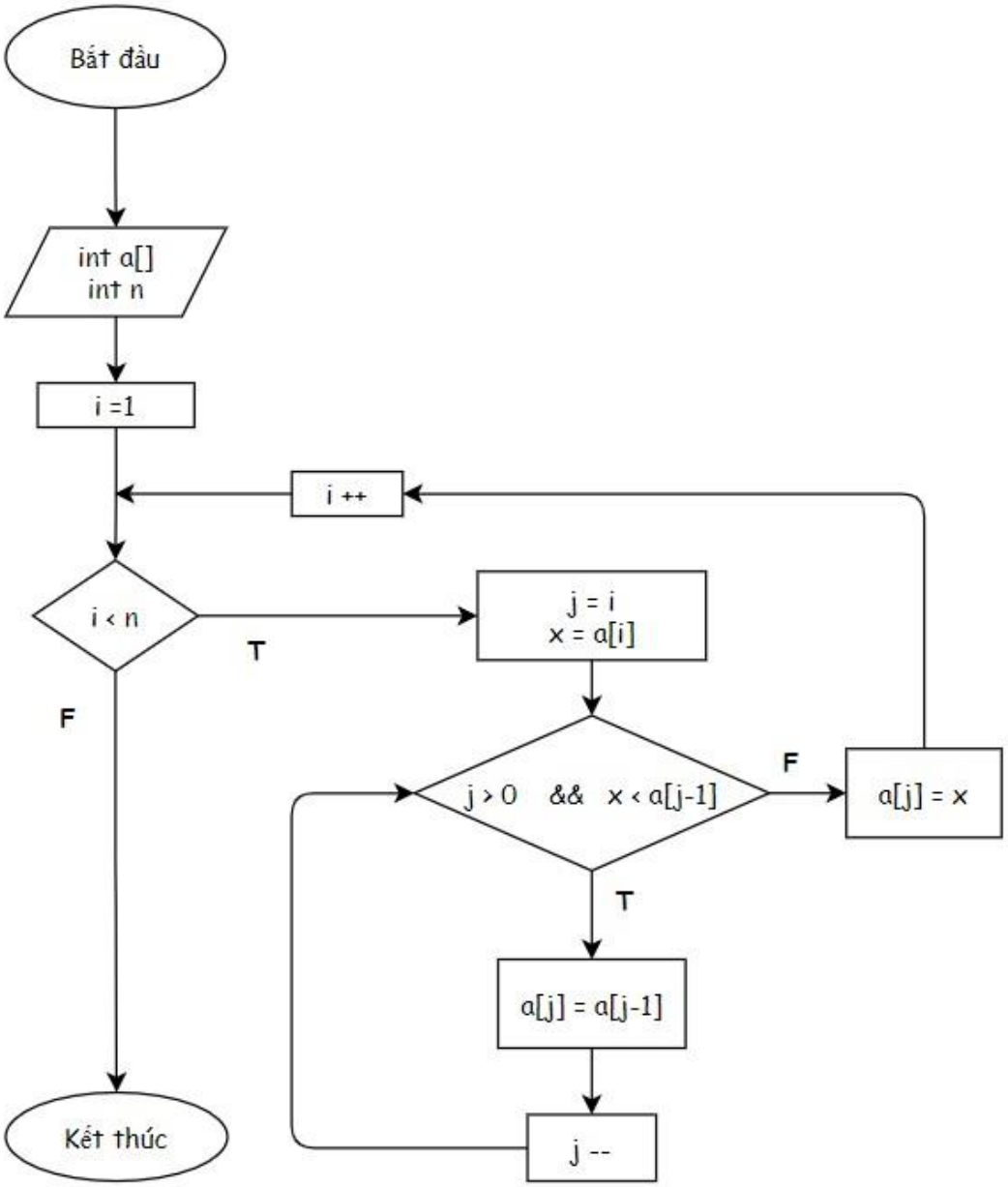
Bước 3: a[j] = x;

- Trong khi (j> 0 và x < a[j-1] // Tìm vị trí thích hợp cho x
  - a[j]=a[j-1];
  - j = j-1;
- a[j] = x;

Bước 4: i = i +1 // Lần duyệt tiếp

- Nếu i > n-1 Dừng vòng lặp, kết thúc thuật toán
- Ngược lại, quay trở về bước 2

Lưu đổ thuật toán:



2.2.3 Cài đặt thuật toán Insertion sort

Thuật toán sắp xếp chèn – Insertion sort cài đặt bằng C/C++

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

Thuật toán sắp xếp chèn còn có hai biến thể đó là xắp xếp chèn nhị phân (Binary Insertion sort). Thuật toán này thay vì duyệt toàn bộ danh sách đang xây dựng phía trước, nó sử dụng phương pháp tìm kiếm nhị phân để tìm vị trí thích hợp cho phần tử trong danh sách, sau đó mới tiến hành chèn.

2.2.4 Đánh giá thuật toán Insertion sort

Bảng đánh giá thuật toán:

Trường hợp	Số lần so sánh	Số lần gán	Độ phức tạp
Tốt nhất	n-1	2(n-1)	O(n)
Xấu nhất	$\frac{n(n-1)}{2}$	$\frac{n(n-1)}{2}$	O(n^2)

Trung bình, thuật toán sắp xếp chèn – Insertion sort có độ phức tạp là O(n^2). Trường hợp tốt nhất là với đầu vào đã được sắp xếp đúng thứ tự. Trường hợp xấu là dãy bị đảo ngược thứ tự hoàn toàn.

Ưu điểm:

- Dễ dàng cài đặt, giải thuật đơn giản dễ hiểu, phù hợp cho việc học tập và nghiên cứu
- Thuật toán có tính ổn định
- Hoạt động tốt nếu mảng cần sắp xếp nhỏ và đã được sắp xếp một phần
- Tốn ít bộ nhớ

Nhược điểm:

- Là thuật toán có độ phức tạp lớn O(n^2) chính vì thế không thể áp dụng với dữ liệu lớn
- Hiệu suất của thuật toán thấp

Trên thực tế, thuật toán ít được sử dụng trong các dự án lớn, tuy nhiên trong một số trường hợp, thuật toán này vẫn rất tối ưu và được sử dụng rộng rãi.

2.3 Thuật toán sắp xếp chọn – Selection sort

2.3.1 Ý tưởng thuật toán Selection sort

Sắp xếp chọn là một thuật toán sắp xếp đơn giản, dựa trên việc so sánh tại chỗ giống với việc sắp xếp tự nhiên.

Giả sử xét với danh sách a có n phần tử. Chọn phần tử nhỏ nhất trong n phần tử ban đầu, đưa phần tử này về vị trí đúng là đầu tiên của dãy hiện hành. Sau đó không quan tâm đến nó nữa, xem dãy hiện hành chỉ còn n-1 phần tử của dãy ban đầu, bắt đầu từ vị trí thứ 2. Giải thuật kết thúc khi phần tử cần tìm vị trí là phần tử cuối cùng.

2.3.2 Giải thuật Selection sort

Thuật toán:

Input: Mảng số nguyên a có n phần tử

Output: mảng a đã được sắp xếp

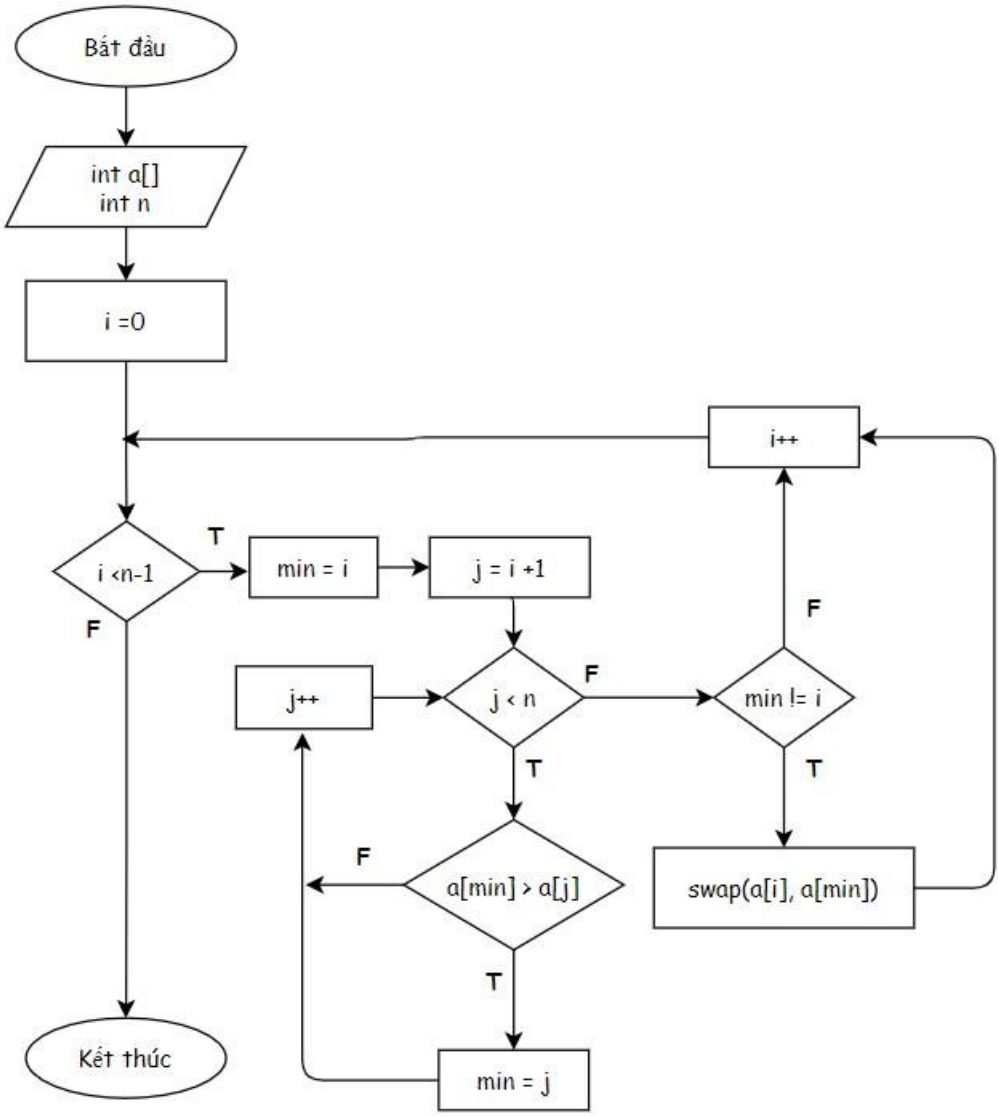
Bước 1 : i = 0

Bước 2 : Tìm phần tử a[min] nhỏ nhất trong dãy hiện hành từ a[i] đến a[n-1]

- Nếu min ≠ i: Đổi chỗ a[min] và a[i]

- Nếu i < n:
  - i =i+1
  - Lặp lại Bước 2
- Ngược lại: Dừng. *//n phần tử đã nằm đúng vị trí*

Lưu đồ thuật toán Sắp xếp chọn:



2.3.3 Cài đặt thuật toán Selection sort

Với tư tưởng và thuật toán nêu ở phần trên, ta có thể cài đặt thuật toán bằng ngôn ngữ C++ như sau:

```
// Thuat toan sap xep chon
```

```
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[i], a[min]);
     }
}
```

2.3.4 Đánh giá thuật toán Selection sort

Thuật toán sắp xếp chọn là thuật toán có độ phức tạp lớn. Thuật toán không phải thực hiện việc đổi chỗ nhiều như Insertion sort hay Bubble sort, tuy nhiên lại phải thực hiện rất nhiều phép so sánh dẫn đến độ phức tạp lớn.

Số phép so sánh: $\sum_{i=1}^{n-1}(n-i) = (n^2-n)/2$

Độ phức tạp của thuật toán: O(n^2) trong mọi trường hợp

Bộ nhớ sử dụng: O(1)

Ưu điểm:

- Dễ dàng cài đặt, dễ hiểu, dễ sử dụng, tốn ít bộ nhớ máy tính
- Phù hợp với danh sách có ít dữ liệu
- Có tính ổn định và tính thích ứng với dữ liệu đầu vào

Nhược điểm:

- Thuật toán tốn thời gian gần như bằng nhau đối với mảng đã được sắp xếp cũng như mảng chưa được sắp xếp.
- Độ phức tạp của thuật toán lớn, tốn nhiều tài nguyên

2.4 Thuật toán sắp xếp nhanh – Quicksort

2.4.1 Nguồn gốc thuật toán

Thuật toán sắp xếp nhanh được phát triển vào năm 1959 bởi Tony Hoare khi ông đang là sinh viên thỉnh giảng tại Đại học Tổng hợp Moscow. Khi đó, Hoare đang thực hiện một dự án về máy dịch cho Phòng thí nghiệm Vật lý Quốc gia. Là một phần của quá trình dịch thuật, ông phải sắp xếp các từ của các câu tiếng Nga trước khi tra cứu chúng trong từ điển Nga-Anh đã được sắp xếp theo thứ tự bảng chữ cái và được lưu trữ trong băng từ. Để hoàn thành nhiệm vụ này, ông đã phát hiện ra Quick Sort và sau đó đã xuất bản mã năm 1961.

2.4.2 Ý tưởng thuật toán

Quicksort là thuật toán ứng dụng ý tưởng chia nhỏ để trị. Đầu tiên nó chia mảng đầu vào thành hai mảng con một nửa nhỏ hơn, một nửa lớn hơn dựa vào một phần tử trung gian. Sau đó, nó sắp xếp đệ quy các mảng con để giải quyết bài toán.

Mấu chốt của thuật toán ở việc chia mảng con hay còn gọi là thuật toán phân đoạn. Cách giải quyết có thể tóm tắt như sau:

- Chọn một phần tử trong mảng làm phần tử trung gian để chia đôi mảng gọi là **pivot**. Thông thường ta thường chọn phần tử đầu tiên, phần tử ở giữa mảng hoặc phần tử cuối cùng của mảng để làm pivot.
- Phân đoạn: di chuyển phần tử có giá trị nhỏ hơn pivot về một bên, tất cả các phần tử có giá trị lớn hơn pivot xếp về một bên (các giá trị bằng pivot có thể đi theo một trong hai cách).
- Sau bước phân đoạn di chuyển pivot về đúng vị trí giữa của 2 mảng con
- Áp dụng đệ quy các bước phân đoạn trên cho hai mảng con

Trường hợp dừng của đệ quy là các mảng có kích thước bằng không hoặc một, khi đó nó đã được sắp xếp theo định nghĩa, vì vậy chúng không bao giờ cần phải sắp xếp.

2.4.3 Giải thuật Quicksort

Có ba phương án chọn phần tử làm chốt cho thuật toán phân đoạn.

- Chọn phần tử đầu
- Chọn phần tử giữa
- Chọn phần tử cuối

Giải thuật dưới đây em viết cho trường hợp chọn phần tử cuối cùng làm pivot.

Giải thuật phân đoạn:

Input: dãy a(low, high)

Output: dãy con a(low, high) được chia làm 3 đoạn

Bước 1: pivot = a[high]; // chọn pivot là phần tử cuối cùng

left = low; right = high-1 /// hai biến duyệt từ đầu và từ cưới

Bước 2: Trong khi left< right:

- Trong khi a[left] < pivot; left++
- Trong khi a[right] > pivot; righ - -
- swap(a[left], a[right];
- left ++ ; right ++

Bước 3:

- swap(a[left], a[high]
- return left;

Giải thuật quick sort:

Input: mảng a[], chỉ số đầu low, chỉ số cuối high.

Output: mảng a[] đã được sắp xếp.

Bước 1: Nếu low = high *//dãy có ít hơn 2 phần tử*

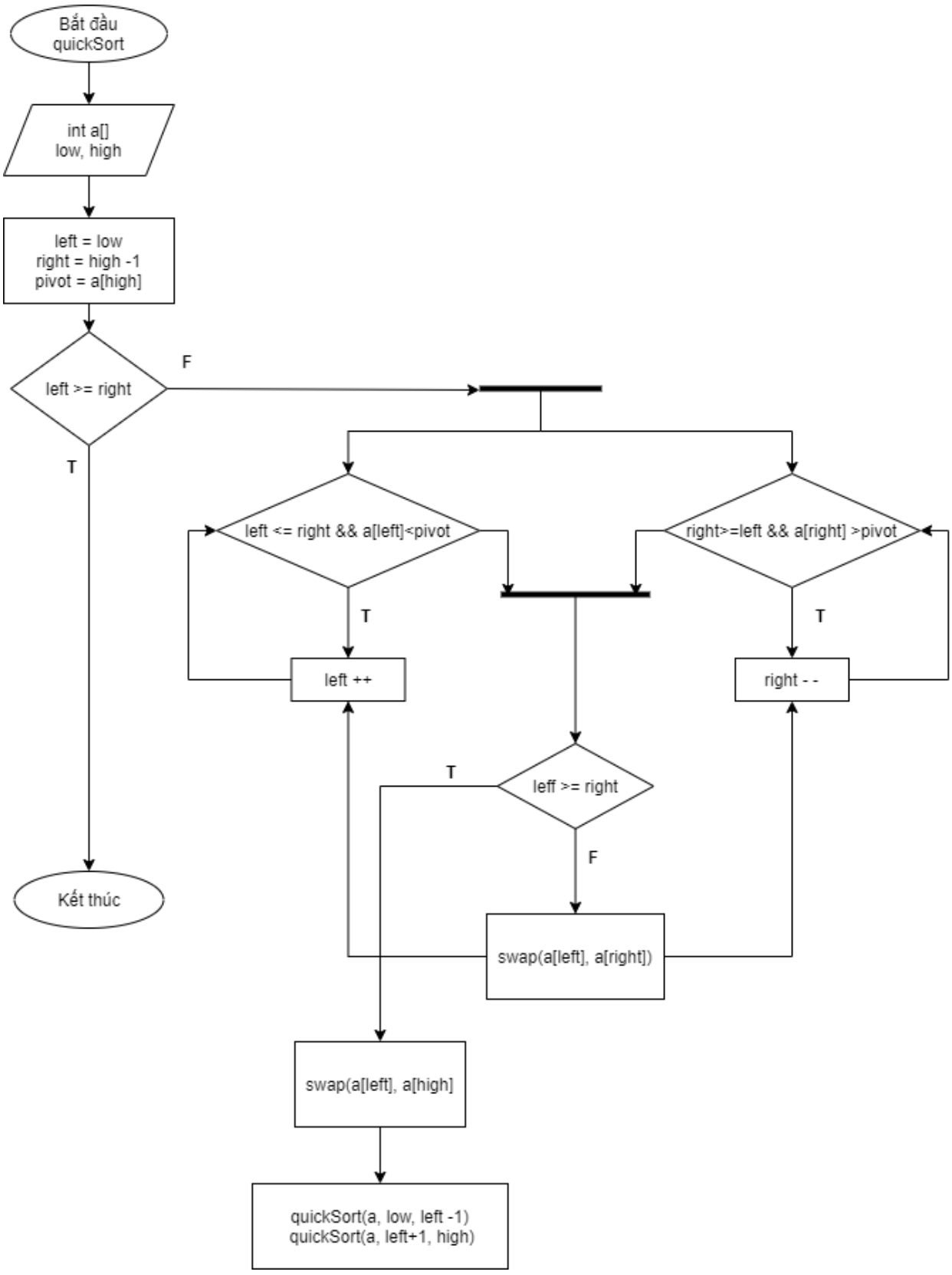
Kết thúc; *//dãy đã được sắp xếp*

Bước 2: Phân hoạch dãy thành 3 đoạn, lấy pivot làm điểm chốt sau khi phân hoạch

Bước 3: Sắp xếp nhanh : a[left].. a[pivot]

Bước 4: Sắp xếp đoạn 3: a[pivot].. a[hight]

Lưu đồ thuật toán sắp xếp nhanh Quicksort:



2.4.4 Cài đặt thuật toán Quicksort

Cách 1: Sử dụng phần tử giữa làm chốt

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

```
     int  i, j, x;
     if (left >= right)  return;
     x = a[(left+right)/2]; // chọn phần tử ở giữa làm chốt
     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);
}
```

Cách 2: Sử dụng phần tử cuối cùng làm chốt:

(vì cách sử dụng phần tử đầu làm chốt tương tự như cách chọn phần tử cuối, nên em bỏ qua trường hợp này)

```
//  Thuật toán sắp xếp nhanh
void Quicksort(int a[], int low, int high){
     if(low >= high)              // Nếu danh sách có hai phần tử
               return;           // Kết thúc
     else{
          int pivot = a[high];
          int right = high-1, left = low;
          while(left<right){
               while(a[left]<pivot) left++;
               while(a[right]>pivot) right--;
               if(left<=right){
                    swap(a[left], a[right]);
```

```
                left++;
                right--;
            }
        }
        swap(a[left], a[high]);
        quickSort(a,low, right); // Quick sort hai nửa
        quickSort(a,left, high);
    }
}
```

2.4.5 Đánh giá thuật toán Quicksort

Bảng đánh giá độ phức tạp của thuật toán:

Trường hợp	Độ phức tạp	Bộ nhớ sử dụng
Tốt nhất	O (n log n)	log n
Trung bình	O (n log n)	log n
Xấu nhất	O(n^2)	log n

Trường hợp xấu nhất xảy ra khi mỗi lần phân đoạn, ta chọn phải phần tử lớn nhất hoặc nhỏ nhất làm chốt.

Trường hợp tốt nhất khi mỗi lần phân đoạn, hai mảng con có độ dài bằng nhau.

Ưu điểm:

- Thuật toán có độ phức tạp nhỏ hơn các thuật toán sắp xếp đơn giản, tốc độ xử lý tương đối nhanh. Trong một số trường hợp, quicksort là thuật toán có tốc độ tốt nhất
- Thông dụng, được áp dụng nhiều trong lập trình, trong thư viện của các ngôn ngữ lập trình như C++, Java, C# . . .
- Có thể ứng dụng vào xử lý dữ liệu lớn

Nhược điểm:

- Thuật toán không có tính ổn định, không có tính thích ứng, dễ ảnh hưởng bởi dữ liệu đầu vào
- Tốn không gian bộ nhớ hơn so với các thuật toán sắp xếp đơn giản
- Tư tưởng và giải thuật khá phức tạp
- Khó khăn trong việc lựa chọn phần tử làm chốt trong phân hoạch. Việc lựa chọn có thể ảnh hưởng rất nhiều đến hiệu suất của thuật toán tuy nhiên ta không thế đưa ra lựa chọn tối ưu nhất.

2.5 Thuật toán sắp xếp trộn – Merge sort

Thuật toán sắp xếp trộn – Merge sort là một trong những phương pháp đầu tiên sử dụng trong bài toán sắp xếp. Thuật toán này được đề xuất bởi John Von Neumann vào đầu năm 1945. Một cuộc thảo luận và phân tích chi tiết về sắp xếp hợp nhất đã được xuất hiện trong một báo cáo của Goldstine và Neumann như đầu năm 1948.

2.5.1 Ý tưởng thuật toán

Các thuật toán sắp xếp đơn giản đều không thể xử lý được dữ liệu lớn. Thuật toán sắp xếp trộn lấy ý tưởng từ việc chia để trị để chia nhỏ bài toán thành các bài toán nhỏ hơn, sau đó giải quyết chúng.

Ý tưởng cụ thể như sau:

- Chia danh sách chưa được sắp xếp thành n danh sách con, mỗi danh sách chứa một phần tử (danh sách một phần tử được coi là đã sắp xếp).
- Liên tục hợp nhất các danh sách con để tạo ra các danh sách con được sắp xếp mớ cho đến khi chỉ còn lại một danh sách. Đây sẽ là danh sách được sắp xếp.

2.5.2 Giải thuật Merge sort

Từ tư tưởng đã nêu ở phần trên, ta cần thiết kế giải thuật như sau:

Bước 1:

- Chia dãy cần sắp xếp thành 2 dãy con
- Từ dãy con thu được lại tiếp tục chia thành 2 dãy con nhỏ hơn nữa
- Quá trình phân chia tiếp tục cho đến khi thu được dãy con chỉ còn duy nhất 1 phần tử.

Bước 2:

- Hòa nhập 2 dãy con nhỏ nhất thành dãy con lớn hơn sao cho đúng thứ tự
- Từ hai dãy con lớn hơn lại hòa nhập thành 2 dãy con lớn hơn nữa….

- Quá trình hòa nhập cứ tiếp tục như vậy cho đến khi thu được dãy số ban đầu đã được sắp xếp.

Với giải thuật nêu trên, ta cần thiết kế hai hàm, một hàm dùng để chia nhỏ danh sách, một hàm dùng để trộn hai danh sách con lại với nhau,

a, Giải thuật hàm trộn (merge):

Input: Danh sách a, điểm bắt đầu *start*, điểm giữa *mid*, điểm kết thúc *n*

Output: Danh sách a được trộn đúng thứ tự

Bước 1: Tạo hai mảng phụ để chia mảng a ban đầu thành hai mảng con left[n1], right[n2] dựa vào phần tử ở vị trí giữa.

Bước 2: i = 0, j =0, k = start

Trong khi (i<n1 && j <n2)

- Nếu a[i] > a[j] thì:
    - a[k] = a[j]; j++;
- Ngược lại
    - a[k] = a[i]; i++;
- k++

Bước 3: Nếu i<n1

- a[k] = a[i]; i++; k++;

Bước 4: Nếu j < n1

- a[k] = a[j]; j++; k++;

b, Giải thuật hàm Mergesort

Input: Dãy a, vị trí đầu *first*, vị trí cuối *end*

Output: Dãy a đã được sắp xếp

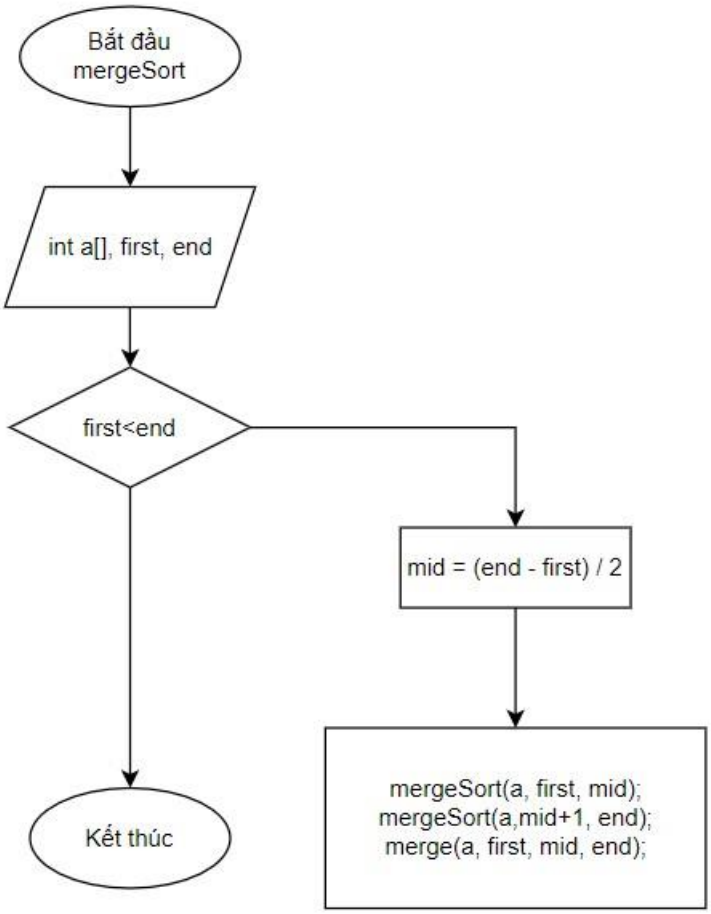
mergeSort(a, first, end):

Nếu first < end:

```
mid = (end – first )/2 ;
mergeSort(a, first, mid);
mergeSort(a, mid+1, end);
merge(a, first, mid, end);
```

Ngược lại, dừng thuật toán

Lưu đồ thuật toán:



2.5.3 Cài đặt thuật toán sắp xếp trộn

Hàm trộn:

```
// Hàm merge
// Mảng a, start, mid, n
void merge(int a[], int p, int t, int n){
     int n1 = mid-start +1;
     int n2= n-mid;
     int left[n1]; int right[n2]; // Tạo hai mảng con

     for(int x=0;x<n1; x++) left[x] = a[p+x];  // Truyền giá trị
     for(int y=0; y<n2; y++) right[y] = a[t+1+y];
     int i=0, j=0;
```

```
      int k=start;
      while(i<n1 && j<n2){   // Trộn hai mảng con
            if( left[i] >= right[j] ){
                  a[k]=right[j];
                  j++;
            }
            else{
                  a[k]=left[i];
                  i++;
            }
            k++;
      }

      while(j<n2){     // Sau khi trộn, mảng phải chưa hết
            a[k]=right[j];
            k++;
            j++;
      }
      while(i<n1){    // Sau khi trộn, mảng trái chưa hết
            a[k]= left[i];
            k++;
            i++;
      }
}
```

Thuật toán sắp xếp trộn:

```
// MergeSort
// Mảng a, vị trí đầu mảng, vị trí cuối mảng
void mergeSort(int a[], int first, int end){
      int mid;
      if(first<end){
```

```
            mid=(first+end)/2;
            mergeSort(a,first,mid);
            mergeSort(a,mid+1,end);
            merge(a,first,mid,end);
        }
}
```

2.5.4 Đánh giá thuật toán Merge sort

Bảng đánh giá thuật toán:

Trường hợp	Độ phức tạp	Bộ nhớ sử dụng
Tốt nhất	O (n log n)	n
Trung bình	O (n log n)	n
Xấu nhất	O(n log n)	n

Ưu điểm:

- Độ phức tạp trung bình O(n log n), tốc độ giải quyết khá nhanh
- Có tính ổn định và thích ứng, tốc độ không bị ảnh hưởng nhiều bởi dữ liệu đầu vào
- Xử lý khá tốt với dữ liệu lớn đặc biệt là dạng list, file

Nhược điểm:

- Tốn nhiều bộ nhớ nếu sử dụng đệ quy
- Code khó cài đặt, tương đối phức tạp
- Trong hầu hết các trường hợp, thuật toán này không được đánh giá cao hơn Quick sort

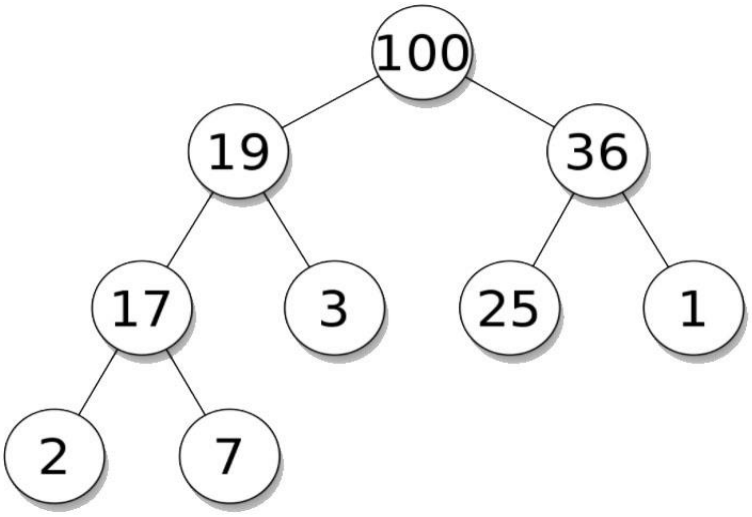
2.6 Thuật toáp sắp xếp vun đống – Heapsort

2.6.1 Cấu trúc heap là gì?

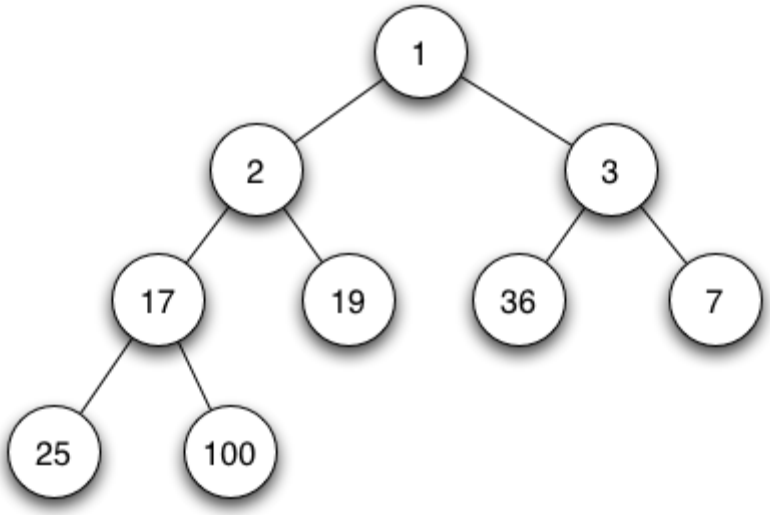
Cấu trúc dữ liệu *heap* hay còn gọi là đống là một trường hợp đặc biệt của cấu trúc dữ liệu cây nhị phân cân bằng. Trong đó khóa của nút gốc được so sánh với các con của nó và được sắp xếp một cách phù hợp.

Dựa vào tính chất so sánh, người ta chia cấu trúc heap thành hai loại:

Max heap: Nút nút cha lớn hơn hoặc bằng nút con



Min heap: Nút cha nhỏ hơn hoặc bằng nút con



Thuật toán sắp xếp vun đống – heapsort thường

2.6.2 Ý tưởng thuật toán Heap sort

Thuật toán Heap sort lấy ý tưởng giải quyết từ cấu trúc heap.

- Ta coi dãy cần sắp xếp là một cây nhị phân hoàn chỉnh, sau đó hiệu chỉnh cây thành dạng cấu trúc heap.
- Dựa vào tính chất của cấu trúc heap, ta có thể lấy được ra phần từ lớn nhất hoặc nhỏ nhất của dãy, phần tử này chính là gốc của heap. Giảm số lượng phần tử của cây nhị phân và tái cấu trúc heap.

- Đưa phần tử đã lấy về đúng vị trí của dãy bằng cách sử dụng một danh sách trung gian để lưu dữ liệu.
- Lặp lại việc lấy phần tử gốc của cấu trúc heap và tái cấu trúc heap cho tới khi danh sách ban đầu chỉ còn 1 phần tử. Đưa phần tử này về đúng vị trí và kết thúc thuật toán.

2.6.3 Giải thuật Heap sort

Thuật toán heapsort có thể áp dụng max heap hoặc min heap để giải quyết. Thông thường, nếu sắp xếp dãy số tăng dần người ta sử dụng max heap, sắp xếp dãy số giảm dần sử dụng min heap. Trong nghiên cứu, ta thường sắp xếp dãy theo chiều tăng dần, vì thế phần này em sử dụng max heap để triển khai thuật toán.

Giải thuật heap sort trải qua hai giai đoạn:

Giai đoạn 1: Hiệu chỉnh heap

Xây dựng hàm heapify

Input: mảng số nguyên a có n phần tử, gốc i cần vun: heapify(a,n,i)

Output: Gốc i đẩy xuống đúng vị trí

- Bước 1: Coi gốc là phần tử lớn nhất, vị trí max = i
- Bước 2: Con trái của i có vị trí: left = i*2 + 1;

  Con phải của i có vị trí: right = i*2 +2;
- Bước 3: Trong khi tồn tại con trái và con phải của i (left<n ; righ<n);
  - Nếu a[left] > a[max] thì max = left;
  - Nếu a[right] >a[max] thì max = right;
- Bước 4: Nếu max != i:
  - Đổi chỗ: swap(a[i], a[max]);
  - heapify(a, n, max);

Hiệu chỉnh thành heap:

Input: mảng số nguyên a, n

Output: Heap

- Bước 1: Xác định vị trí nút gốc cao nhất : i = n/2 -1
- Bước 2: Trong khi chưa hiệu chỉnh hết các nút gốc: i >= 0:
  - Heapify(a,n,i)

- i = i -1;

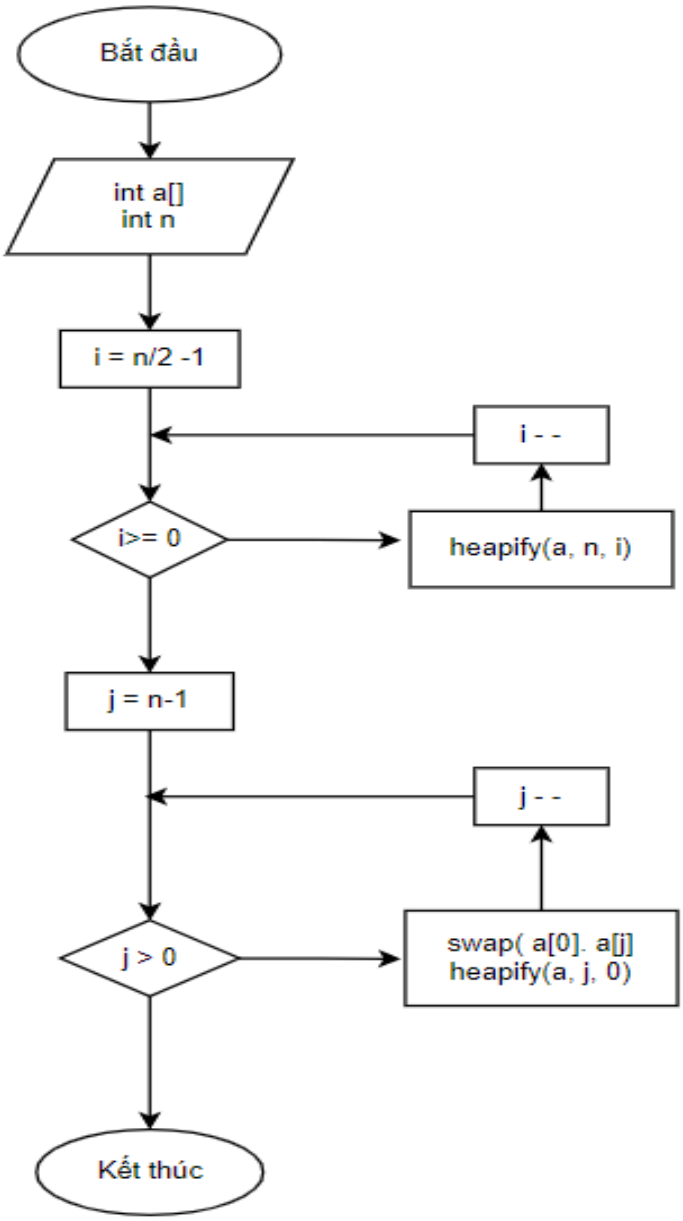
Giai đoạn 2: Sắp xếp danh sách dựa vào heap

Input: Mảng a, có n phần tử đã được hiệu chỉnh thành heap

Output: a được sắp xếp theo chiều tăng dần

- Bước 1: Duyệt từ vị trí phần tử cuối danh sách j = n -1
- Bước 2: Trong khi j > 0:
  - Đổi chỗ, đẩy phần tử lớn nhất về cuối mảng swap(a[0], a[j]
  - Đẩy nút 0 xuống đúng vị trí trong mảng có j phần tử: heapify(a, j, 0)
  - Giảm vị trí phần tử cuối cùng của mảng: j - -;

Lưu đồ thuật toán Heap sort:



2.6.4 Cài đặt thuật toán sắp xếp vun đống – Heap sort

Cài đặt thuật toán sắp xếp vun đống bằng C++:

```
// Thuat toan sap xep vun dong
// Ham vun dong cho mot dinh
void heapify(int arr[], int n, int i){
     int max =i;
     int l = i*2 +1;
     int r = l+1;
     if(l<n && arr[l] > arr[max]){  // Nếu con trái lớn hơn gốc
          max = l;
     }
     if(r<n && arr[r] > arr[max]){// Nếu con phải nhỏ hơn gốc
          max = r;
     }
     if(max != i){
          swap(arr[i], arr[max]);   // Đổi chỗ gốc
          heapify(arr ,n , max);
     }

}

// Hàm sắp xếp vun đống
void heapSort(int arr[], int n){
     // Vun đống từ dưới lên
     for(int i = n/2 - 1; i>=0; i--)
          heapify(arr,n, i);
     for(int j = n-1 ; j>0; j--){
          swap(arr[0], arr[j] );  // Đối chỗ phần tử lớn nhất
          heapify(arr, j, 0);     // Tái tạo heap
     }
}
```

2.6.5 Đánh giá thuật toán sắp xếp vun đống

Sắp xếp vun đống là thuật toán sắp xếp có ứng dụng đến cấu trúc dữ liệu heap, cây nhị phân. Chính vì thế đây là một thuật toán có độ phức tạp không quá cao nhưng lại khá phức tạp.

Bảng đánh giá thuật toán:

Trường hợp	Độ phức tạp	Bộ nhớ sử dụng
Tốt nhất	O (n log n)	1
Trung bình	O (n log n)	1
Xấu nhất	O(n log n)	1

Ưu điểm:

- Có độ phức tạp trung bình O (n log n) trong mọi trường hợp. Là một trong các thuật toán sắp xếp nhanh nhất
- Ít bị ảnh hưởng bởi dữ liệu đầu vào, có thể ứng dụng nhiều trong thực tế

Nhược điểm:

- Thuật toán cài đặt phức tạp, khó khăn trong việc hiểu thuật toán
- Là thuật toán sắp xếp không có tính ổn định
- Không tối ưu trong mọi trường hợp

CHƯƠNG III: CHƯƠNG TRÌNH MINH HỌA THUẬT TOÁN SẮP XẾP

3.1 Giới thiệu công cụ và ngôn ngữ sử dụng

3.1.1 Tổng quan về ngôn ngữ lập trình Csharp

C# hay "C-sharp" là một ngôn ngữ lập trình hướng đối tượng được phát triển bởi Tập đoàn Microsoft. Ngôn ngữ này được xem là ngôn ngữ lập trình hướng đối tượng trong sáng và thuần nhất. Nó hiện thực hầu hết các tính chất tốt của mô hình hướng đối tượng giống như ngôn ngữ lập trình Java. C# là ngôn ngữ được Microsoft phát triển dựa trên 2 ngôn ngữ huyền thoại đó là C++ và Java. Và nó cũng được miêu tả là loại ngôn ngữ có được sự cân bằng giữa C++, Visual Basic, Delphi và Java.

C# là ngôn ngữ lập trình: đơn giản, hiện đại, hướng đối tượng. C# làm việc chủ yếu trên bộ khung .NET (.NET framework). Ngôn ngữ lập trình này có khả năng tạo ra nhiều ứng dụng mạnh mẽ và an toàn cho nền tảng Windows. Các thành phần máy chủ, dịch vụ web, ứng dụng di động và nhiều khả năng khác nữa.

Một số loại ứng dụng được xây dựng bằng CSharp:

- Ứng dụng console
- Ứng dụng windows form
- Ứng dụng web sử dụng công nghệ ASP.net
- Lập trình game với Unity

3.1.2 Ứng dụng windows form

Windows Forms (WinForms) là thư viện lớp đồ họa (GUI) mã nguồn mở và miễn phí được bao gồm như một phần của Microsoft.NET Framework hoặc Mono Framework cung cấp nền tảng để viết các ứng dụng khách phong phú cho máy tính windows.

Ứng dụng windows form là ứng dụng được xây dựng từ thư viện nêu trên. Đây là loại ứng dụng có khả năng:

- Giao tiếp với người dùng bằng bàn phím và mouse.
- Có giao diện đồ họa
- Thực hiện các xử lý do người dùng xây dựng

Windows form có khả năng xử lý đồ họa rất tốt, cùng với kiến thức đã học em đã chọn nó để xây dựng ứng dụng minh họa thuật toán sắp xếp quick sort của mình.

3.1.3 Microsoft Visual studio

a, Tổng quan Visual Studio

Microsoft Visual Studio là một môi trường phát triển tích hợp (IDE) từ Microsoft. Nó được sử dụng để phát triển chương trình máy tính cho Microsoft Windows, cũng như các trang web, các ứng dụng web và các dịch vụ web. Visual Studio sử dụng nền tảng phát triển phần mềm của Microsoft như Windows API, Windows Forms, Windows Presentation Foundation, Windows Store và Microsoft Silverlight.

Visual Studio được viết bằng 2 ngôn ngữ đó chính là C++ và C# chính vì thế khả năng biên tập, xây dựng chương trình ứng dụng rất mạnh mẽ.

b, Một số tính năng của phần mềm Visual Studio

- Biên tập mã: Visual Studio là một trình soạn thảo mã hỗ trợ tô sáng cú pháp và hoàn thiện tự động mã nguồn. Bên cạnh đó, IDE còn có khả năng biên dịch ngầm nhằm kiểm tra và phát hiện mã lỗi ngay khi người dùng sử dụng.
- Trình gỡ lỗi: Visual Studio có một trình gỡ lỗi có tính năng vừa lập trình gỡ lỗi cấp máy và gỡ lỗi cấp mã nguồn. Nó có thể gỡ lỗi các ứng dụng được viết bằng các ngôn ngữ được hỗ trợ bởi Visual Studio.
- Thiết kế: Visual studio cung cấp khả năng thiết kế giao diện cho các ứng dụng windows forms, WPF, Web

Đặc biệt, đây là ứng dụng quan trọng giúp xây dựng thiết kế các ứng dụng windows form.

3.2 Chương trình minh họa thuật toán

3.2.1 Mô tả chung về chương trình

Chương trình cài đặt, xây dựng bằng windows form C# nhằm mục đích cung cấp giao diện người dùng và mô phỏng cách thức hoạt động cho thuật toán sắp xếp Quicksort.

Các phần tử trong mảng được minh họa bằng các button hình tròn. Mảng dữ liệu do người dùng thiết lập được mô phỏng bằng mảng các button. Các chuyển động trên chương trình được xây dựng nhờ vào việc thay đổi vị trí của các button.

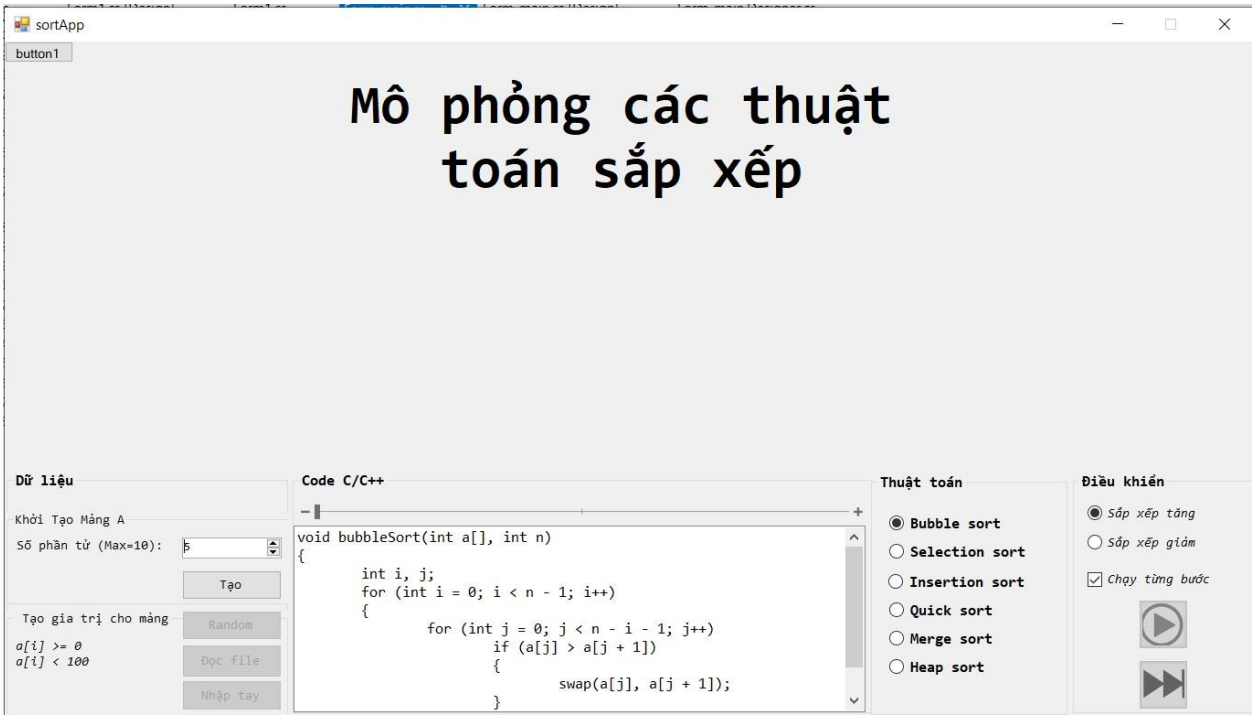
3.2.2 Các tính năng của chương trình

Các tính năng của chương trình mô phỏng thuật toán Quicksort như sau:

- Tính năng tạo mảng, khởi tạo mảng ban đầu với tùy chọn
- Tinh năng nhập giá trị cho mảng: Có thể chọn ngẫu nhiên, đọc từ file hoặc nhập tay
- Tính năng sắp xếp: có thể chọn sắp xếp tăng, giảm, từng bước hoặc nhanh

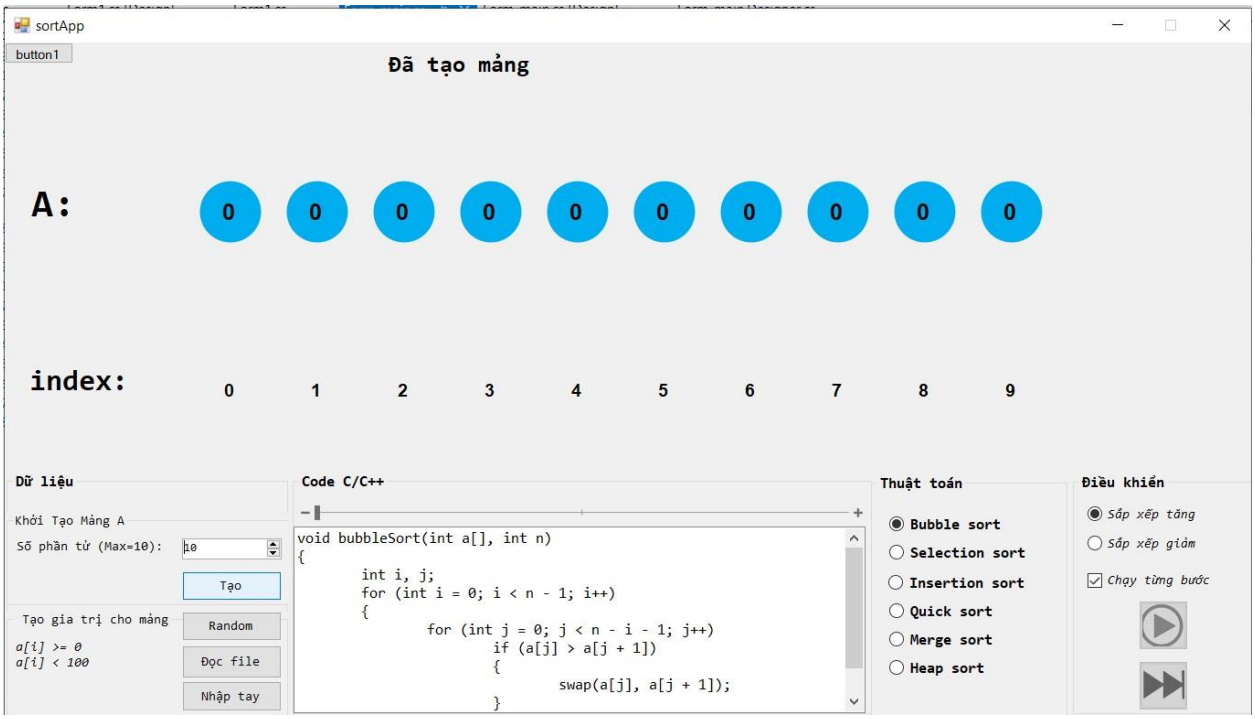
3.2 Một số hình ảnh hoạt động của chương trình

Hình 1: Giao diện khi khởi chạy chương trình



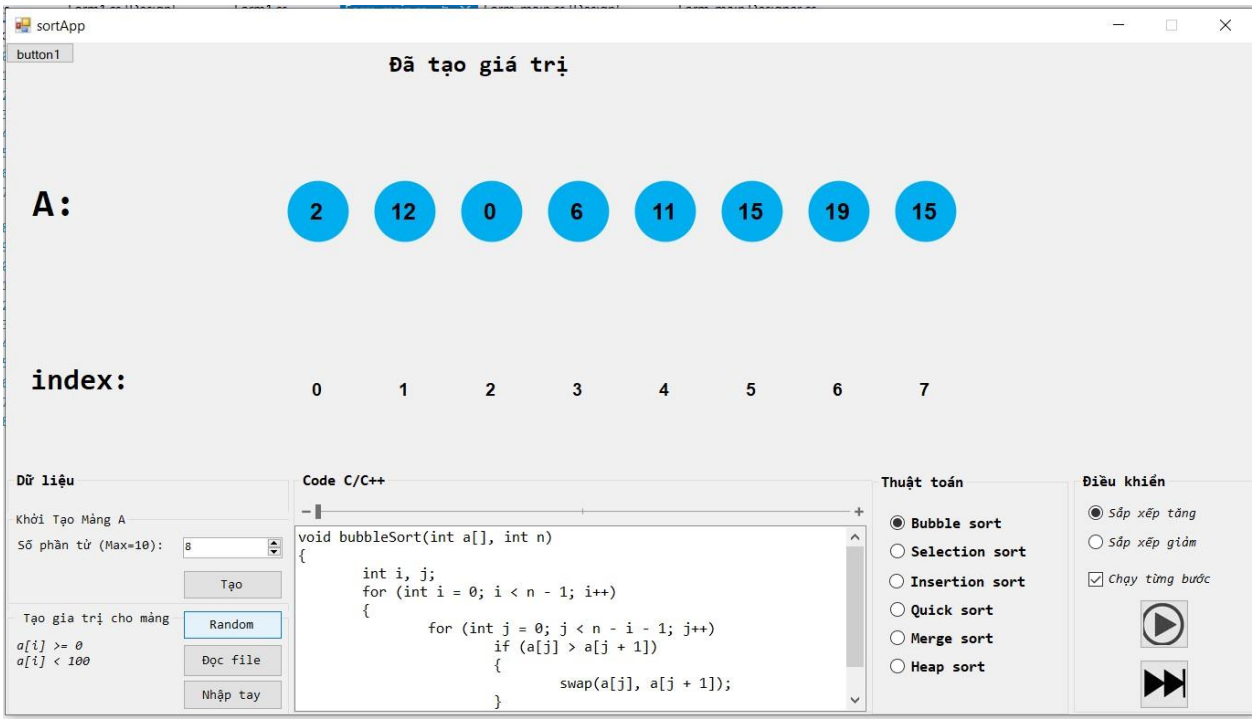
Khi khởi chạy chương trình, con trỏ chuột sẽ tập trung vào phần nhập số phần từ mảng, người dùng chọn số lượng phần tử trong mảng cần tạo. Ở đây để có thể thực hiện minh họa, em thiết kế cho chỉ có thể tạo số lượng phần từ lớn hơn 1 và nhỏ hơn 10, nếu sai sẽ báo nhập lại.

Hình 2: Giao diện sau chi chọn tạo mảng thành công



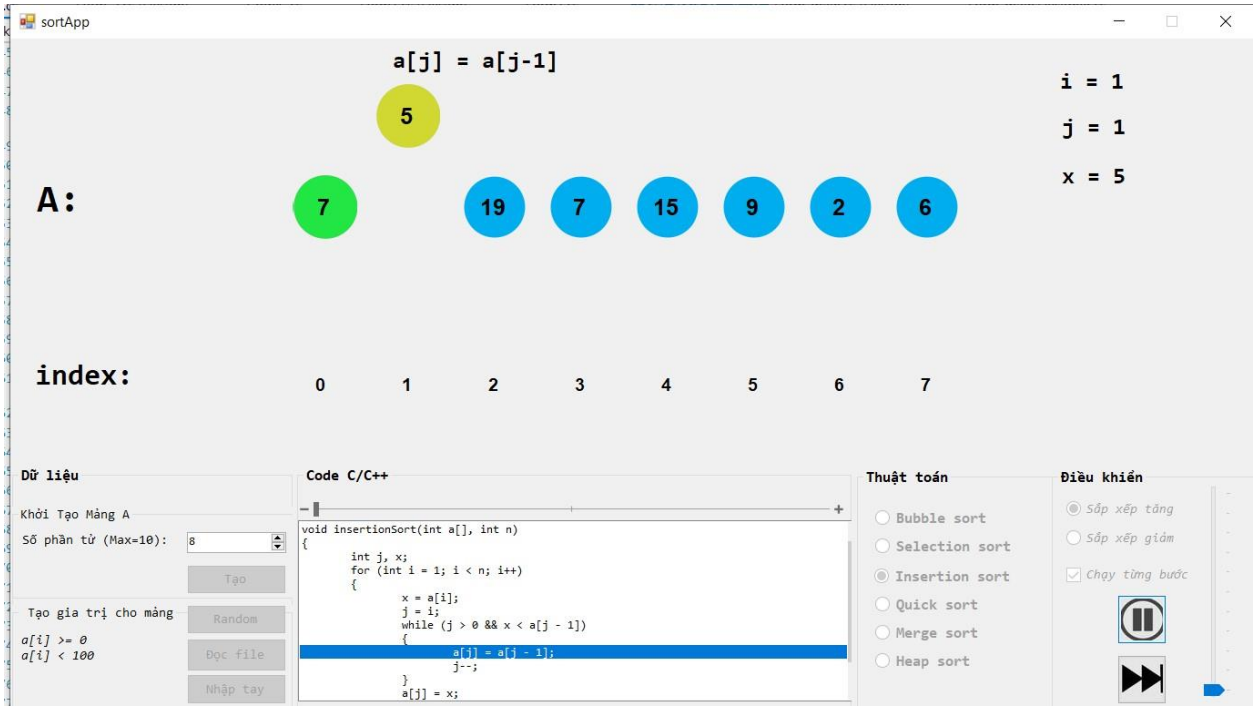
Sau khi tạo mảng thành công, các button tạo giá trị cho mảng có thể tương tác. Người dùng có thể tạo giá trị bằng cách random, đọc từ file và nhập tay từ bàn phím.

Hình 3: Giao diện sau khi đã tạo giá trị

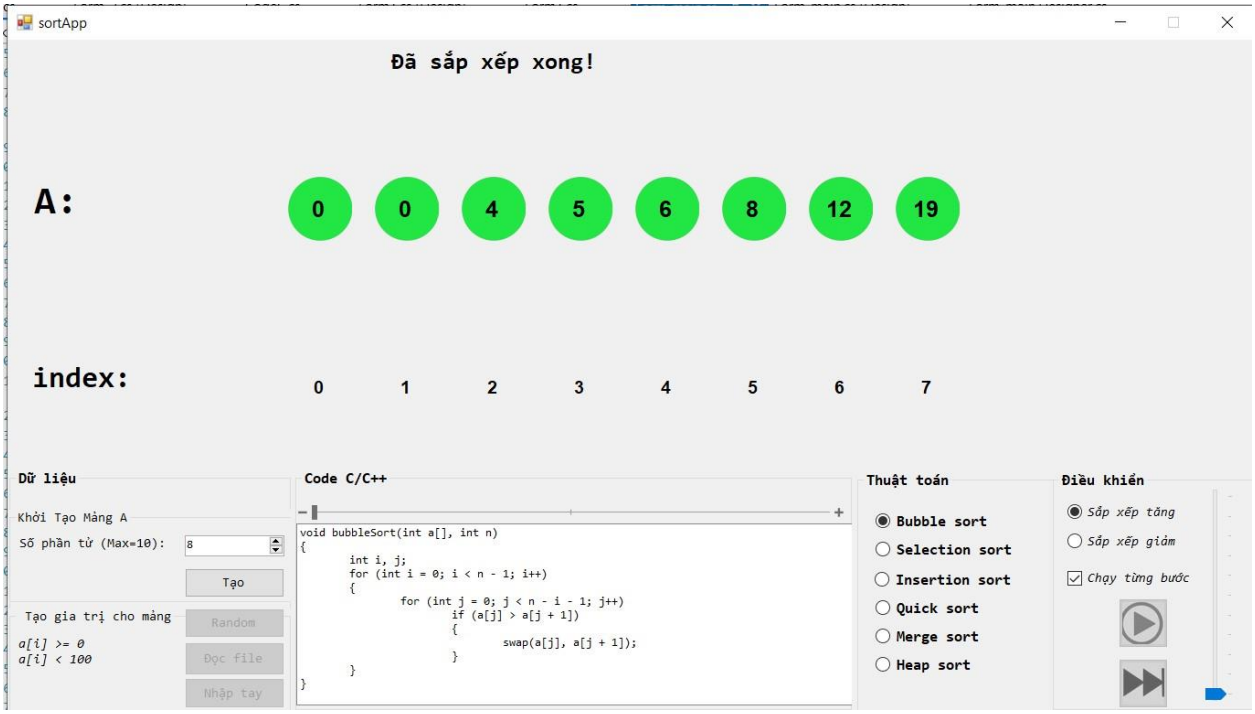


- Trong khu vực thuật toán, người dùng có thể chọn một trong các thuật toán sắp xếp tương ứng.
- Trong khu vực điều khiển, người dùng có thể chọn kiểu sắp xếp mong muốn như tăng dần, giảm dần.

Hình 4: Giao diện chương trình khi đang tiến hành sắp xếp



Hình 5: Giao diện khi thực hiện sắp xếp xong



KẾT LUẬN

Trong thời gian 8 tuần thực hiện đề tài thực tập cơ sở nghiên cứu về các thuật toán sắp xếp thông dụng này, em đã phần nào nắm vững hơn về các kiến thức lập trình cơ sở, tư duy thuật toán và làm chủ được một số thuật toán sắp xếp thông dụng. Ngoài ra còn nắm vững hơn kiến thức về lập trình hướng đối tượng với Csharp, window form.

Xây dựng được một chương trình mô phòng thuật toán sắp xếp đơn giản. Chương trình có thể giúp người xem hiểu về cách thực hoạt động của thuật toán.

Em biết rằng, kiến thức của mình vẫn còn nhiều thiếu sót, những kiến thức nghiên cứu chưa thực sự nhiều và chính xác nhưng cũng có phần nào hoàn thiện thêm vốn kiến thức của mình.

Rất mong nhận được những lời góp ý của thầy cô và các bạn giúp em hoàn thiện hơn kiến thức của mình.

TÀI LIỆU THAM KHẢO

Bài báo cáo thực tập cơ sở được em tham khảo và nghiên cứu nội dung kiến thức từ

Sách và giáo trình:

[1]. Lê Minh Hoàng (1999-2006). Giải thuật và lập trình : Đại học sư phạm Hà Nội

[2].Giáo trình Ứng dụng thuật toán: Trường ĐH CNTT & TT Thái Nguyên

Trang web về các thuật toán sắp xếp:

[3].https://vi.wikipedia.org/wiki/Thuật_toán_sắp_xếp/

[4].https://en.wikipedia.org/wiki/Sorting_algorithm/

[5].https://en.wikipedia.org/wiki/Quicksort/

[6].https://en.wikipedia.org/wiki/Merge_sort/

[7].https://en.wikipedia.org/wiki/Heapsort/

[8].https://en.wikipedia.org/wiki/Selection_sort/

[9].https://en.wikipedia.org/wiki/Insertion_sort/

[10].https://www.geeksforgeeks.org/heap-sort/

[11].https://www.geeksforgeeks.org/merge-sort/

Trang web về ngôn ngữ lập trình C#, Windows Form và Visual Studio:

[12].https://visualstudio.microsoft.com/

[13].https://docs.microsoft.com/en-us/dotnet/csharp/tutorials/

[14].https://www.howkteam.vn/course/lap-trinh-winform-co-ban/

NHẬN XÉT CỦA GIẢNG VIÊN HƯỚNG DẪN