TRƯỜNG ĐẠI HỌC SƯ PHẠM TP. HỒ CHÍ MINH

**KHOA CÔNG NGHỆ THÔNG TIN**



**ỨNG DỤNG**

**MÔ PHỎNG THUẬT TOÁN**

**TÌM KIẾM VÀ SẮP XẾP**

**Người thực hiện:**

1/ Nguyễn Thiên Ân 43.01.104.003

2/ Hồ Khả Việt Huấn 43.01.104.058

**Giáo viên hướng dẫn:**

1/ GV. Lương Trần Ngọc Khiết

TP. HỒ CHÍ MINH 2018 – 2019

# MỤC LỤC

Nội dung

[MỤC LỤC 2](#_Toc5951331)

[MỞ ĐẦU 3](#_Toc5951332)

[CHƯƠNG 1: CƠ SỞ LÝ THUYẾT 4](#_Toc5951333)

[1.1. CÂY NHỊ PHÂN 4](#_Toc5951334)

[1.2. MẢNG VÀ HÀNG ĐỢI 5](#_Toc5951335)

[1.3. THUẬT TOÁN TÌM KIẾM 6](#_Toc5951336)

[1.3.1. THUẬT TOÁN TÌM KIẾM TUẦN TỰ (SEQUENTIAL SEARCH) 6](#_Toc5951337)

[1.3.2. THUẬT TOÁN TÌM KIẾM NHỊ PHÂN (BINARY SEARCH) 6](#_Toc5951338)

[1.4. THUẬT TOÁN SẮP XẾP: 7](#_Toc5951339)

[1.4.1. THUẬT TOÁN SẮP XẾP NỔI BỌT (BUBBLE SORT) 7](#_Toc5951340)

[1.4.2. THUẬT TOÁN SẮP XẾP CHỌN (SELECTION SORT) 8](#_Toc5951341)

[1.4.3. THUẬT TOÁN SẮP XẾP CHÈN (INSERTION SORT) 8](#_Toc5951342)

[1.4.4. THUẬT TOÁN SẮP XẾP BẮN TRÁI PHÁ (SHELL SORT) 8](#_Toc5951343)

[1.4.5. THUẬT TOÁN SẮP XẾP NHANH (QUICK SORT) 9](#_Toc5951344)

[1.4.6. THUẬT TOÁN SẮP XẾP TRỘN (MERGE SORT) 10](#_Toc5951345)

[1.4.7. THUẬT TOÁN SẮP XẾP THEO CƠ SỐ (RADIX SORT) 12](#_Toc5951346)

[1.4.8. THUẬT TOÁN SẮP XẾP VUN ĐỐNG (HEAP SORT) 12](#_Toc5951347)

[1.4.9. THUẬT TOÁN SẮP XẾP ĐẾM PHÂN PHỐI (COUNTING SORT) 14](#_Toc5951348)

[CHƯƠNG 2: GIỚI THIỆU ỨNG DỤNG VÀ HƯỚNG DẪN SỬ DỤNG 15](#_Toc5951349)

[2.1. NGÔN NGỮ LẬP TRÌNH VÀ CẤU TRÚC ỨNG DỤNG 15](#_Toc5951350)

[2.1.1. NGÔN NGỮ LẬP TRÌNH VÀ FRAMEWORK 15](#_Toc5951351)

[2.1.2. CẤU TRÚC ỨNG DỤNG 15](#_Toc5951352)

[2.2. GIỚI THIỆU VỀ CHỨC NĂNG VÀ HƯỚNG DẪN SỬ DỤNG 17](#_Toc5951353)

[2.2.1. GIỚI THIỆU CHỨC NĂNG 17](#_Toc5951354)

[2.2.2. HƯỚNG DẪN SỬ DỤNG 17](#_Toc5951355)

[CHƯƠNG 3: PHÂN CÔNG VIỆC 22](#_Toc5951356)

[3.1. NGUYỄN THIÊN ÂN 22](#_Toc5951357)

[3.2. HỒ KHẢ VIỆT HUẤN 22](#_Toc5951358)

[PHỤ LỤC – TÀI LIỆU THAM KHẢO 23](#_Toc5951359)

# MỞ ĐẦU

Cấu trúc dữ liệu và giải thuật được xem như là 2 yếu tố quan trọng nhất trong lập trình, nắm vững các cấu trúc dữ liệu và giải thuật là cơ sở để chúng ta tiếp cận với việc thiết kế và xây dựng phần mềm cũng như việc sử dụng các công cụ lập trình hiện đại sau này.

Cấu trúc dữ liệu là cách thức tổ chức sắp xếp dữ liệu để biểu diễn thông tin trên máy tính. Khi thông tin đã được biểu diễn hay được lưu trữ trên máy, công việc còn lại của người lập trình là phải xử lý các thông tin đó, hay nói một cách khác là thao tác các dữ liệu đó nhằm giải quyết các vấn đề đã đặt ra. Và để xử lý dữ liệu đó một cách hiệu quả thì cần phải có các giải thuật để áp dụng trên các dữ liệu đó. Chính vì vậy, chúng ta đã thấy rằng cấu trúc dữ liệu và giải thuật là hai thuật ngữ luôn đi kèm với nhau. Cấu trúc dữ liệu là đầu vào và cũng là đầu ra của giải thuật. Với mỗi một loại cấu trúc dữ liệu cụ thể phải có những giải thuật phù hợp. Và ngược lại, mỗi giải thuật chỉ thực sự hiệu quả khi được áp dụng trên một số cấu trúc dữ liệu nhất định.

Trong ứng dụng này, chúng tôi áp dụng các thuật toán tìm kiếm và sắp xếp vào các cấu trúc dữ liệu cơ bản theo một trình tự nhất định để người dùng dễ tiếp cận nhất. Nắm rõ các cấu trúc và giải thuật này sẽ giúp cho học viên tìm hiểu dễ dàng hơn các cấu trúc dữ liệu và giải thuật phức tạp, nâng cao hơn. Ứng dụng được xây dựng với giao diện thân thiện người dùng, cách sử dụng phù hợp, mang tính thực tiễn nhằm phục vụ tốt nhất việc học tập của học sinh khi học về thuật toán và bất kì ai muốn tìm hiểu về các thuật toán tìm kiếm và sắp xếp.

Tập thể nhóm thực hiện



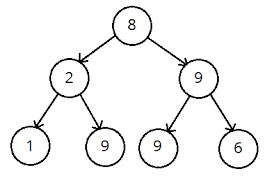
# CHƯƠNG 1: CƠ SỞ LÝ THUYẾT

Trong chương này, nhóm nghiên cứu sẽ giới thiệu các kiến thức nền tảng, cơ sở lý thuyết của ứng dụng. Từ đó, người đọc có thể hiểu rõ lý thuyết và nắm được nguyên lý hoạt động của ứng dụng. Chương này bao gồm 4 phần nhỏ cũng là 4 lý thuyết chính mà nhóm nghiên cứu sử dụng để tạo ra ứng dụng.

* Cây nhị phân.
* Mảng và hàng đợi.
* Thuật toán tìm kiếm.
* Thuật toán sắp xếp.

## CÂY NHỊ PHÂN

* **Nút:** Một nút có thể chứa một giá trị, một điều kiện, một cấu trúc dữ liệu riêng biệt hoặc chính một cây. Mỗi nút trong một cây có thể không có hoặc có một số nút con, các nút con có mức cao hơn nó (theo quy ước khác với cây tự nhiên, cây trong cấu trúc dữ liệu phát triển từ trên xuống). Một nút có con được gọi là nút cha của các nút con. Một nút có nhiều nhất một nút cha.
* **Nút gốc:** Trong mỗi cây có một nút đặc biệt được gọi là nút gốc (hay nói đơn giản là gốc). Nút gốc là nút duy nhất không có nút cha. Nút gốc là nơi khởi đầu của nhiều giải thuật trên cây. Tất cả các nút khác được nối về nút gốc bằng một đường đi qua các cạnh hay các liên kết
* **Nút lá:** Các nút không có nút con được gọi là nút lá hay gọi đơn giản là lá.
* **Nút nhánh:** Nút trong của một cây là nút trên cây có ít nhất một con, nghĩa là các nút không phải là lá. Các khái niệm về mức của mỗi nút, chiều cao của cây được định nghĩa giống như cây trong lý thuyết đồ thị.
* **Cây nhị phân:** Các cây trong đó mỗi nút có không quá hai con được gọi là cây nhị phân.
* **Xây dựng cây nhị phân bằng mảng trong lập trình:** Ta có thể dùng một mảng gồm 2h+1 - 1 {\displaystyle 2^{h+1}-1}phần tử để biểu diễn cây nhị phân, bằng cách lần lượt lưu trữ thông tin của mỗi nút vào mảng theo thứ tự từ trên xuống dưới, từ trái sang phải. Khi đó con trái của nút thứ i là phần tử thứ 2\*i, con phải là phần tử thứ 2\*i+1. Cha của phần tử thứ i là phần tử thứ int(i/2).Ta gán giá trị Null cho các vị trí còn thiếu.



## MẢNG VÀ HÀNG ĐỢI

* **Mảng:** Mảng là một tập hợp các phần tử cố định có cùng một kiểu, được lưu trữ liên tiếp nhau trong các ô nhớ. Kiểu phần tử có thể là có các kiểu bất kỳ: [ký tự](https://vi.wikipedia.org/wiki/K%C3%BD_t%E1%BB%B1), [số](https://vi.wikipedia.org/wiki/S%E1%BB%91), [chuỗi ký tự](https://vi.wikipedia.org/w/index.php?title=Chu%E1%BB%97i_k%C3%BD_t%E1%BB%B1&action=edit&redlink=1)…; cũng có khi ta sử dụng kiểu mảng để làm kiểu phần tử cho một mảng (trong trường hợp này ta gọi là mảng của mảng hay mảng nhiều chiều).
* **Hàng đợi:** Hàng đợi (queue) là một [cấu trúc dữ liệu](https://vi.wikipedia.org/wiki/C%E1%BA%A5u_tr%C3%BAc_d%E1%BB%AF_li%E1%BB%87u) dùng để chứa các đối tượng làm việc theo cơ chế FIFO (First In First Out), nghĩa là "vào trước ra trước"

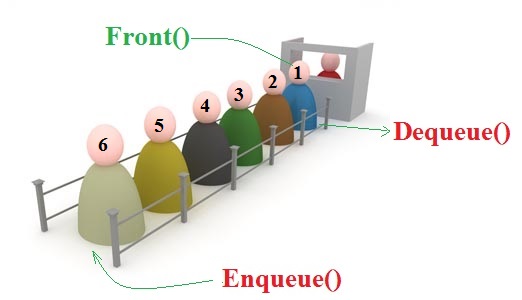
Trong hàng đợi, các đối tượng có thể được thêm vào hàng đợi bất kỳ lúc nào, nhưng chỉ có đối tượng thêm vào đầu tiên mới được phép lấy ra khỏi hàng đợi. Thao tác thêm vào và lấy một đối tượng ra khỏi hàng đợi được gọi lần lượt là "enqueue" và "dequeue". Việc thêm một đối tượng luôn diễn ra ở cuối hàng đợi và một phần tử luôn được lấy ra từ đầu hàng đợi.

Trong tin học, cấu trúc dữ liệu hàng đợi có nhiều ứng dụng: khử [đệ quy](https://vi.wikipedia.org/wiki/%C4%90%E1%BB%87_quy), tổ chức lưu vết các quá trình [tìm kiếm theo chiều rộng](https://vi.wikipedia.org/wiki/T%C3%ACm_ki%E1%BA%BFm_theo_chi%E1%BB%81u_r%E1%BB%99ng) và [quay lui](https://vi.wikipedia.org/wiki/Quay_lui_(khoa_h%E1%BB%8Dc_m%C3%A1y_t%C3%ADnh)), [vét cạn](https://vi.wikipedia.org/w/index.php?title=V%C3%A9t_c%E1%BA%A1n&action=edit&redlink=1), tổ chức quản lý và phân phối [tiến trình](https://vi.wikipedia.org/wiki/Ti%E1%BA%BFn_tr%C3%ACnh) trong các [hệ điều hành](https://vi.wikipedia.org/wiki/H%E1%BB%87_%C4%91i%E1%BB%81u_h%C3%A0nh), tổ chức [bộ đệm bàn phím](https://vi.wikipedia.org/w/index.php?title=B%E1%BB%99_%C4%91%E1%BB%87m_b%C3%A0n_ph%C3%ADm&action=edit&redlink=1).

Cấu trúc dữ liệu hàng đợi có thể định nghĩa như sau: Hàng đợi là một [cấu trúc dữ liệu trừu tượng](https://vi.wikipedia.org/w/index.php?title=C%E1%BA%A5u_tr%C3%BAc_d%E1%BB%AF_li%E1%BB%87u_tr%E1%BB%ABu_t%C6%B0%E1%BB%A3ng&action=edit&redlink=1) (ADT) tuyến tính. Tương tự như [ngăn xếp](https://vi.wikipedia.org/wiki/Ng%C4%83n_x%E1%BA%BFp), hàng đợi hỗ trợ các thao tác:

* EnQueue(x): thêm đối tượng x vào cuối hàng đợi.
* DeQueue(): lấy đối tượng ở đầu queue ra khỏi hàng đợi và trả về giá trị của nó. Nếu hàng đợi rỗng thì lỗi sẽ xảy ra.
* Queue.Count != 0: kiểm tra xem hàng đợi có rỗng không.
* Peek(): trả về giá trị của phần tử nằm ở đầu hàng đợi mà không hủy nó. Nếu hàng đợi rỗng thì lỗi sẽ xảy ra.

Các thao tác thêm, trích và huỷ một phần tử phải được thực hiện ở hai phía khác nhau của hàng đợi, do đó hoạt động của hàng đợi được thực hiện theo nguyên tắc FIFO. Cũng như ngăn xếp, cấu trúc mảng một chiều hoặc cấu trúc [danh sách liên kết](https://vi.wikipedia.org/wiki/Danh_s%C3%A1ch_li%C3%AAn_k%E1%BA%BFt) có thể dùng để biểu diễn cấu trúc hàng đợi.



## THUẬT TOÁN TÌM KIẾM

### THUẬT TOÁN TÌM KIẾM TUẦN TỰ (SEQUENTIAL SEARCH)

* Trong [Khoa học máy tính](https://vi.wikipedia.org/wiki/Khoa_h%E1%BB%8Dc_m%C3%A1y_t%C3%ADnh) tìm kiếm tuần tự (Sequential search) hay tìm kiếm tuyến tính (linear search) là một phương pháp tìm kiếm một phần tử cho trước trong một [danh sách](https://vi.wikipedia.org/wiki/Danh_s%C3%A1ch) bằng cách duyệt lần lượt từng phần tử của danh sách đó cho đến lúc tìm thấy giá trị mong muốn hay đã duyệt qua toàn bộ danh sách.
* Code:

for (i=1; i<=size; i++)

{

if (arr[i] == Value)

{

kt = true;

break;

}

}

if (i > size) kt = false;

### THUẬT TOÁN TÌM KIẾM NHỊ PHÂN (BINARY SEARCH)

* Trong [khoa học máy tính](https://vi.wikipedia.org/wiki/Khoa_h%E1%BB%8Dc_m%C3%A1y_t%C3%ADnh), thuật toán tìm kiếm nhị phân (Binary search) là một thuật toán dùng để tìm kiếm phần tử trong một danh sách **đã được sắp xếp**. Thuật toán hoạt động như sau:
  + Trong mỗi bước, so sánh phần tử cần tìm với phần tử nằm ở chính giữa danh sách.
  + Nếu hai phần tử bằng nhau thì phép tìm kiếm thành công và thuật toán kết thúc.
  + Nếu phần tử cần tìm lớn hơn phần tử nằm ở chính giữa danh sách thì thuật toán lặp lại bước so sánh trên với nửa sau của danh sách.
  + Nếu phần tử cần tìm bé hơn phần tử nằm ở chính giữa danh sách thì thuật toán lặp lại bước so sánh trên với nửa đầu của danh sách.
  + Vì số lượng phần tử trong danh sách cần xem xét giảm đi một nửa sau mỗi bước, nên thời gian thực thi của thuật toán là hàm [lôgarit](https://vi.wikipedia.org/wiki/L%C3%B4garit).

Code:

int left = 1, right = size, mid = left; kt = false;

while (left <= right && mid <= size && mid >= 1)

{

mid = (left + right) / 2;

if (arr[mid] == Value)

{

kt = true;

break;

}

else if (arr[mid] > Value)

{

right = mid - 1;

}

else

{

left = mid + 1;

}

}

## THUẬT TOÁN SẮP XẾP:

### THUẬT TOÁN SẮP XẾP NỔI BỌT (BUBBLE SORT)

* Sắp xếp nổi bọt (Bubble sort) là một [thuật toán sắp xếp](https://vi.wikipedia.org/wiki/Thu%E1%BA%ADt_to%C3%A1n_s%E1%BA%AFp_x%E1%BA%BFp) đơn giản, với thao tác cơ bản là so sánh hai phần tử kề nhau, nếu chúng chưa đứng đúng thứ tự thì đổi chỗ (swap). Có thể tiến hành từ trên xuống (bên trái sang) hoặc từ dưới lên (bên phải sang). Sắp xếp nổi bọt còn có tên là sắp xếp bằng so sánh trực tiếp. Nó sử dụng phép so sánh các phần tử nên là một giải thuật [sắp xếp kiểu so sánh](https://vi.wikipedia.org/w/index.php?title=S%E1%BA%AFp_x%E1%BA%BFp_ki%E1%BB%83u_so_s%C3%A1nh&action=edit&redlink=1):
  + Giả sử dãy cần sắp xếp có N phần tử. Khi tiến hành từ trên xuống, ta so sánh hai phần tử đầu, nếu phần tử đứng trước lớn hơn phần tử đứng sau thì đổi chỗ chúng cho nhau. Tiếp tục làm như vậy với cặp phần tử thứ nhất và thứ ba và tiếp tục cho đến cuối tập hợp dữ liệu. Sau bước này phần tử đầu tiên chính là phần tử nhỏ nhất của dãy.
  + Sau đó, quay lại so sánh (và đổi chố nếu cần) phần tử thứ hai cho đến khi gặp phần tử thứ N....
  + Ghi chú: Nếu trong một lần duyệt, không phải đổi chỗ bất cứ cặp phần tử nào thì danh sách đã được sắp xếp xong.
  + Nếu muốn sắp xếp từ lớn đến nhỏ thì ngược dấu so sánh lại.



* Code:

for (int i = 1; i < size; i++)

{

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

{

if (arr[i] > arr[j]) **Swap**(ref arr[i], ref arr[j]);

}

}

### THUẬT TOÁN SẮP XẾP CHỌN (SELECTION SORT)

* Sắp xếp chọn là một [thuật toán sắp xếp](https://vi.wikipedia.org/wiki/Thu%E1%BA%ADt_to%C3%A1n_s%E1%BA%AFp_x%E1%BA%BFp) đơn giản, dựa trên việc so sánh tại chỗ:
  + 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.
  + Lặp lại quá trình trên cho dãy hiện hành đến khi dãy hiện hành chỉ còn một phần tử.
  + Dãy ban đầu có N phần tử, vậy tóm tắt ý tưởng thuật toán là thực hiện N-1 lượt việc đưa phần tử nhỏ nhất trong dãy hiện hành về vị trí đúng ở đầu dãy.
* Code:

for (int i = 1; i < size; i++)

{

min = i;

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

{

if (arr[j] < arr[min]) min = j;

}

if (min != i) **Swap**(ref arr[min], ref arr[i]);

}

### THUẬT TOÁN SẮP XẾP CHÈN (INSERTION SORT)

* Sắp xếp chèn (Insertion sort) là một [thuật toán sắp xếp](https://vi.wikipedia.org/wiki/Thu%E1%BA%ADt_to%C3%A1n_s%E1%BA%AFp_x%E1%BA%BFp) 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.
* Code:

for (int i = 2; i <= size; i++)

{

int h = arr[i];

int j = i;

while (j - 1 > 0 && h < arr[j-1])

{

arr[j] = arr[j-1];

j--;

}

arr[j] = h;

}

### THUẬT TOÁN SẮP XẾP BẮN TRÁI PHÁ (SHELL SORT)

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

inc = 4;

while (inc > 0)

{

for (i = 1; i <= size; i++)

{

j = i;

temp = arr[i];

while ((j > inc) && (arr[j - inc] > temp))

{

arr[j] = arr[j - inc];

j = j - inc;

}

arr[j] = temp;

}

if (inc / 2 != 0)

inc = inc / 2;

else if (inc == 1)

inc = 0;

else

inc = 1;

}

### THUẬT TOÁN SẮP XẾP NHANH (QUICK SORT)

* Sắp xếp nhanh (Quick sort), còn được gọi là sắp xếp kiểu phân chia (part sort) là một [thuật toán sắp xếp](https://vi.wikipedia.org/wiki/Thu%E1%BA%ADt_to%C3%A1n_s%E1%BA%AFp_x%E1%BA%BFp) phát triển bởi [Charles Antony Richard Hoare](https://vi.wikipedia.org/wiki/Charles_Antony_Richard_Hoare)c sắp thành hai danh sách con:
  + Khác với [sắp xếp trộn](https://vi.wikipedia.org/wiki/S%E1%BA%AFp_x%E1%BA%BFp_tr%E1%BB%99n), chia danh sách cần sắp xếp {\displaystyle a[1..n]}A[1..N] thành hai danh sách con có kích thước tương đối bằng nhau nhờ chỉ số đứng giữa danh sách, sắp xếp nhanh chia nó thành hai danh sách bằng cách so sánh từng phần tử của danh sách với một phần tử được chọn được gọi là phần tử chốt.
  + Những phần tử nhỏ hơn hoặc bằng phần tử chốt được đưa về phía trước và nằm trong danh sách con thứ nhất, các phần tử lớn hơn chốt được đưa về phía sau và thuộc danh sách đứng sau.
  + Cứ tiếp tục chia như vậy tới khi các danh sách con đều có độ dài bằng 1.

Code:

Queue left = new Queue(), right = new Queue();

left.**Enqueue**(1);

right.**Enqueue**(size);

int l, r, m, i, j;

while (left.Count != 0)

{

l = (int)left.**Dequeue**();

r = (int)right.**Dequeue**();

m = arr[(l + r) / 2];

i = l;

j = r;

while (i <= j)

{

while (arr[i] < m) i++;

while (arr[j] > m) j--;

if (i <= j)

{

if (i < j) **Swap**(ref arr[i], ref arr[j]);

i++;

j--;

}

}

if (l < j)

{

left.**Enqueue**(l);

right.**Enqueue**(j);

}

if (i < r)

{

left.**Enqueue**(i);

right.**Enqueue**(r);

}

}

### THUẬT TOÁN SẮP XẾP TRỘN (MERGE SORT)

* Sắp xếp trộn (Merge Sort) là một giải thuật sắp xếp dựa trên giải thuật chia để trị (Divide and Conquer). Với độ phức tạp thời gian trường hợp xấu nhất là Ο(n log n) thì đây là một trong các giải thuật đáng được quan tâm nhất.

Code:

public void **MergeSort\_Recursive**(int left, int right)

{

int mid;

if (right > left)

{

mid = (right + left) / 2; *//Divide step*

**MergeSort\_Recursive**(left, mid);*//Conquer step*

**MergeSort\_Recursive**((mid + 1), right);*//Conquer step*

**DoMerge**(left, (mid + 1), right);*//Conquer step*

}

}

public void **DoMerge**(int left, int mid, int right)

{

int[] temp = new int[25];

int i, left\_end, num\_elements, tmp\_pos;

left\_end = (mid - 1);

tmp\_pos = left;

num\_elements = (right - left + 1);

while ((left <= left\_end) && (mid <= right))

{

if (numbers[left] <= numbers[mid])

{

temp[tmp\_pos++] = numbers[left++];

}

else

{

temp[tmp\_pos++] = numbers[mid++];

}

}

while (left <= left\_end)

{

temp[tmp\_pos++] = numbers[left++];

}

while (mid <= right)

{

temp[tmp\_pos++] = numbers[mid++];

}

for (i = 0; i < num\_elements; i++)

{

numbers[right] = temp[right];

right--;

}

}

### THUẬT TOÁN SẮP XẾP THEO CƠ SỐ (RADIX SORT)

* Trong [khoa học máy tính](https://vi.wikipedia.org/wiki/Khoa_h%E1%BB%8Dc_m%C3%A1y_t%C3%ADnh), thuật toán sắp xếp theo cơ số (Radix sort) là một thuật toán [sắp xếp không so sánh](https://vi.wikipedia.org/w/index.php?title=S%E1%BA%AFp_x%E1%BA%BFp_kh%C3%B4ng_so_s%C3%A1nh&action=edit&redlink=1):
  + Thuật toán này được thực hiện dựa trên ý tưởng nếu một dãy số đã được sắp xếp hoàn chỉnh thì từng chữ số cũng sẽ được sắp xếp hoàn chỉnh dựa trên giá trị của các chữ số đó.
  + Thuật toán này yêu cầu dãy cần được sắp xếp có thể so sánh thứ tự các vị trí vì thế sắp xếp theo cơ số không giới hạn ở tập số nguyên (ta có thể dễ dàng đưa dạng xâu về cơ số nhị phân).

Code:

int[] tmp = new int[arr.Length];

for (int shift = 31; shift > -1; --shift)

{

j = 0;

for (i = 0; i < arr.Length; ++i)

{

bool move = (arr[i] << shift) >= 0;

if (shift == 0 ? !move : move)

arr[i - j] = arr[i];

else

tmp[j++] = arr[i];

}

Array.**Copy**(tmp, 0, arr, arr.Length - j, j);

}

### THUẬT TOÁN SẮP XẾP VUN ĐỐNG (HEAP SORT)

* Sắp xếp vun đống (Heap sort) dựa trên một [cấu trúc dữ liệu](https://vi.wikipedia.org/wiki/C%E1%BA%A5u_tr%C3%BAc_d%E1%BB%AF_li%E1%BB%87u) được gọi là [đống nhị phân](https://vi.wikipedia.org/wiki/%C4%90%E1%BB%91ng_nh%E1%BB%8B_ph%C3%A2n) (binary heap), gọi đơn giản là đống, nó có thể được xem như bản cải tiến của Selection sort khi chia các phần tử thành 2 mảng con, 1 mảng các phần tử đã được sắp xếp và mảng còn lại các phần tử chưa được sắp xếp. Trong mảng chưa được sắp xếp, các phần tử lớn nhất sẽ được tách ra và đưa vào mảng đã được sắp xếp. Điều cải tiến ở Heapsort so với Selection sort ở việc sử dụng cấu trúc dữ liệu heap thay vì tìm kiếm tuyến tính (linear-time search) như Selection sort để tìm ra phần tử lớn nhất:
  + Ta sẽ sắp xếp chúng thành một heap (dạng cấu trúc cây nhị phân). Heap này có thể là Min-heap (nút gốc có giá trị bé nhất) hoặc Max-heap (nút gốc có giá trị lớn nhất), trong bài viết này, ta sẽ sử dụng Max-heap với một số yêu cầu thỏa mãn sau:
    - Nút cha sẽ luôn lớn hơn tất cả các nút con, nút gốc của heap sẽ là phần tử lớn nhất.
    - Heap được tạo thành phải là một cây nhị phân đầy đủ, tức ngoại trừ các nút lá, ở cùng một cấp độ các nút nhánh không được thiếu.
  + Code:

private void **BuildHeap**(int size)

{

heapSize = size;

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

{

**Heapify**(i);

}

}

private void **Heapify**(int index)

{

int left = 2 \* index;

int right = 2 \* index + 1;

int largest = index;

if (left <= heapSize && arr[left] > arr[index])

{

largest = left;

}

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

{

largest = right;

}

if (largest != index)

{

**Swap**(index, largest);

**Heapify**(largest);

}

}

* Giai đoạn này gồm các thao tác được lặp đi lặp lại cho đến khi mảng dữ liệu được toàn tất sắp xếp:
* Đưa phần tử lớn nhất của heap được tạo vào mảng kết quả, mảng này sẽ chứa các phần tử đã được sắp xếp.
* Sắp xếp lại heap sau khi loại bỏ nút gốc (có giá trị lớn nhất) để tìm phần tử có giá trị lớn nhất tiếp theo.
* Thực hiện lại thao tác 1 cho đến khi các phần tử của heap đều được đưa vào mảng kết quả.
  + - Như thế, mảng kết quả sẽ chứa các phần tử được sắp xếp giảm dần.
  + Code:

public void **PerformHeapSort**(int size)

{

**BuildHeap**(size);

for (int i = arr.Length - 1; i >= 0; i--)

{

**Swap**(0, i);

heapSize--;

**Heapify**(0);

}

}

### THUẬT TOÁN SẮP XẾP ĐẾM PHÂN PHỐI (COUNTING SORT)

* Sắp xếp đếm phân phối (Counting Sort) là một thuật toán sắp xếp đơn giản cho trường hợp đặc biệt. Trường hợp đặc biệt ở đây là các giá trị trong mảng cần sắp xếp đều là số nguyên và biết được giá trị của dãy nằm trong khoảng nào đó. Ví dụ như [0..M], [3..N],…

Code:

for (int i=1; i<=size; i++)

{

count[arr[i]]++;

if (copy[i] > maxx) maxx = copy[i];

}

for (int i = 1; i <= maxx; i++)

{

count[i] += count[i - 1];

}

arr = new int[101];

for (int i = 1; i <= size; i++)

{

arr[count[copy[i]]] = copy[i];

count[copy[i]] -= 1;

}

# CHƯƠNG 2: GIỚI THIỆU ỨNG DỤNG VÀ HƯỚNG DẪN SỬ DỤNG

Trong chương này chúng tôi sẽ giới thiệu các bước xây dựng và hoàn thiện ứng dụng của nhóm. Qua chương này, người đọc sẽ hiểu các bước thực hiện ứng dụng của chúng tôi.

## NGÔN NGỮ LẬP TRÌNH VÀ CẤU TRÚC ỨNG DỤNG

### NGÔN NGỮ LẬP TRÌNH VÀ FRAMEWORK

* Chúng tôi đã sử dụng IDE Visual Studio xây dựng 1 WPF (Windows Presentation Foundation) do [Microsoft](https://vi.wikipedia.org/wiki/Microsoft) phát triển, là công nghệ kế tiếp [Windows Form](https://vi.wikipedia.org/w/index.php?title=Windows_Form&action=edit&redlink=1) dùng để xây dựng các ứng dụng dành cho máy trạm chạy hệ điều hành Windows. WPF được giới thiệu từ năm 2006 trong .NET Framework 3.0 (dưới tên gọi Avalon), công nghệ này nhận được sự quan tâm của cộng đồng lập trình viên bởi nhiều điểm đổi mới trong lập trình ứng dụng và khả năng xây dựng giao diện thân thiện, sinh động. Tại Việt Nam, WPF thực sự chưa phát triển so với nhánh khác là Silverlight (WPF/E).
* WPF hỗ trợ tốt hơn Winform trong việc xử lý hình ảnh, âm thanh, video, quản lý phông chữ, quản lý hiển thị và chỉnh sửa văn bản. Các control trong WPF có thể được lồng ghép linh động để tạo ra giao diện do được viết bằng [XAML](https://vi.wikipedia.org/w/index.php?title=XAML&action=edit&redlink=1). Một ứng dụng WPF có thể được xây dựng để chạy độc lập dưới dạng mở rộng EXE hoặc đóng gói với phần mở rộng là XBAP để có thể tích hợp lên website.
* Và ngôn ngữ lập trình chính được chúng tôi sử dụng là C#, một [ngôn ngữ lập trình](https://vi.wikipedia.org/wiki/Ng%C3%B4n_ng%E1%BB%AF_l%E1%BA%ADp_tr%C3%ACnh) [hướng đối tượng](https://vi.wikipedia.org/wiki/L%E1%BA%ADp_tr%C3%ACnh_h%C6%B0%E1%BB%9Bng_%C4%91%E1%BB%91i_t%C6%B0%E1%BB%A3ng) được phát triển bởi [Microsoft](https://vi.wikipedia.org/wiki/Microsoft). Theo một hướng nào đó, là ngôn ngữ lập trình phản ánh trực tiếp nhất đến [.NET Framework](https://vi.wikipedia.org/wiki/.NET_Framework) mà tất cả các chương trình.NET chạy, và nó phụ thuộc mạnh mẽ vào framework này. Mọi dữ liệu cơ sở đều là đối tượng, được cấp phát và hủy bỏ bởi trình dọn rác Garbage-Collector (GC), và nhiều kiểu trừu tượng khác chẳng hạn như class, delegate, interface, exception... phản ánh rõ ràng những đặc trưng của.NET runtime.

### CẤU TRÚC ỨNG DỤNG

* + Vì có nhiều thuật toán và mỗi thuật toán có các cách áp dụng cấu trúc dữ liệu khác nhau, nên cách biểu diễn cấu trúc đó cũng khác nhau, ở đây, nhóm chúng tôi chia làm 3 bộ cấu trúc là Columm, Radix và Tree:
    - Columm là cách biểu diễn một mảng số thành các cột số tương ứng với các phần tử mảng như biểu đồ thống kê.
    - Radix là biểu diễn 2 mảng số, 1 mảng thập phân và mảng nhị phân tương ứng, sau đó có 2 biểu mẫu 0 và 1 tương ứng là 2 cơ số ở hệ nhị phân để thực hiện phép sắp xếp theo hệ cơ số.
    - Tree là cách biểu diễn mảng thành cây nhị phân với mỗi giá trị phần tử là một nút.

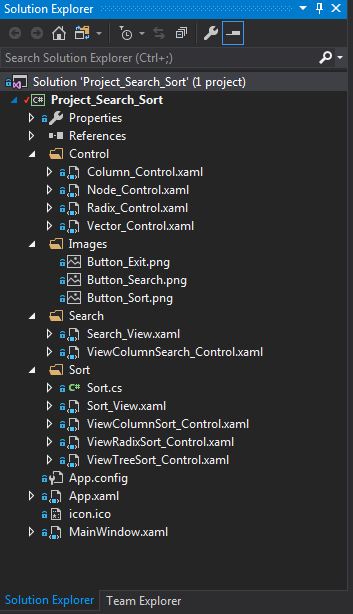


Figure 1: Project's structure

* Folder Control: là những class của các bộ cấu trúc:
  + Columm\_Control: layout của 1 cột và hiệu ứng trên cột.
  + Radix\_Control: layout của 1 ô nhị phân và hiệu ứng tô màu cơ số.
  + Node\_Control: layout của 1 nút trên cây trúc cây và hiệu ứng trên nút.
  + Vector\_Control: layout của 1 đường nối giữa các nút.
* Folder Images: hình backrough.
* Folder Search: là giao diện của các cửa sổ và layout liên quan đến Search:
  + Search\_View: Cửa sổ Search.
  + ViewColummSearch\_Control: layout 1 group mô hình cột.
* Folder Sort: là giao diện của các cửa sổ và layout liên quan đến Sort:
  + Sort.cs: code mẫu các thuật toán sắp xếp.
  + Sort\_View: Cửa sổ Sort.
  + ViewColummSort\_Control: layout 1 group mô hình cột.
  + ViewRadixSort\_Control: layout 1 group mô hình biểu diễn mảng cơ số.
  + ViewTreeSort\_Control: layout 1 group mô hình cây nhị phân.
* MainWindow: Cửa sổ khởi đầu.

## GIỚI THIỆU VỀ CHỨC NĂNG VÀ HƯỚNG DẪN SỬ DỤNG

### GIỚI THIỆU CHỨC NĂNG

Chức năng chính ứng dụng của chúng tôi là mô phỏng lại các thuật toán tìm kiếm và sắp xếp một cách trực quan, để những ai tìm hiểu về chúng sẽ dễ dàng hình dung cách thức làm việc của mỗi thuật toán. Việc này giúp ích rất nhiều cho những giáo viên tin học ở trung học phổ thông khi dạy về lập trình. Và ứng dụng của chúng tôi thiết kế đơn giản nhất chỉ tập trung vào 2 chức năng tìm kiếm và sắp xếp:

* Chức năng tìm kiếm gồm có 3 lựa chọn là:
  + Tìm kiếm tuần tự (Linear Search).
  + Tìm kiếm nhị phân (Binary Search).
* Chức năng sắp xếp gồm có n lựa chọn là:
  + Sắp xếp nổi bọt (Bubble Sort).
  + Sắp xếp chọn (Selection Sort).
  + Sắp xếp chèn (Insertion Sort).
  + Sắp xếp bắn trái phá (Shell Sort).
  + Sắp xếp nhanh (Quick Sort).
  + Sắp xếp trộn (Merge Sort).
  + Sắp xếp theo cơ số (Radix Sort).
  + Sắp xếp vun đống (Heap Sort).
  + Sắp xếp đếm phân phối (Counting Sort).

### HƯỚNG DẪN SỬ DỤNG

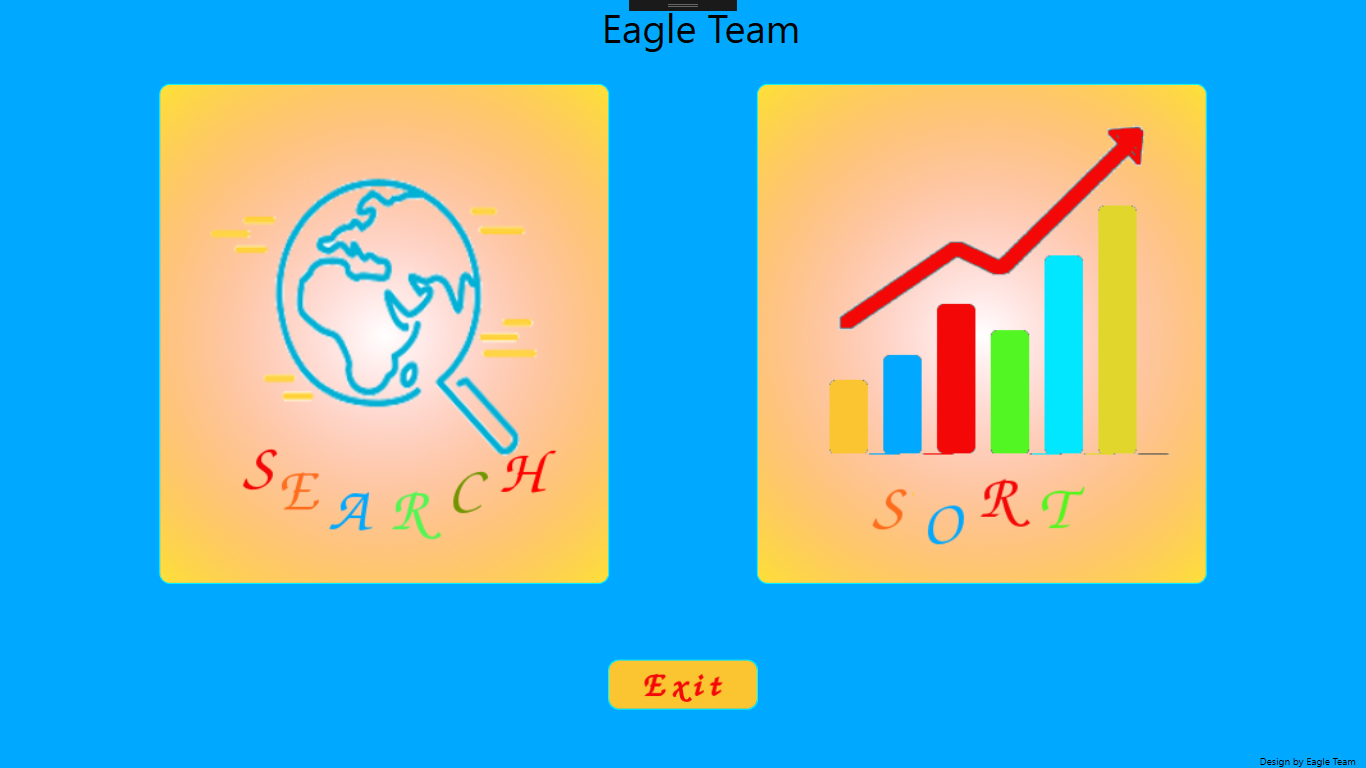


Figure 2: Main Window

* Màn hình chính gồm 3 button là Search, Sort và Exit:
  + Khi chọn button Search sẽ vào cửa sổ Tìm Kiếm

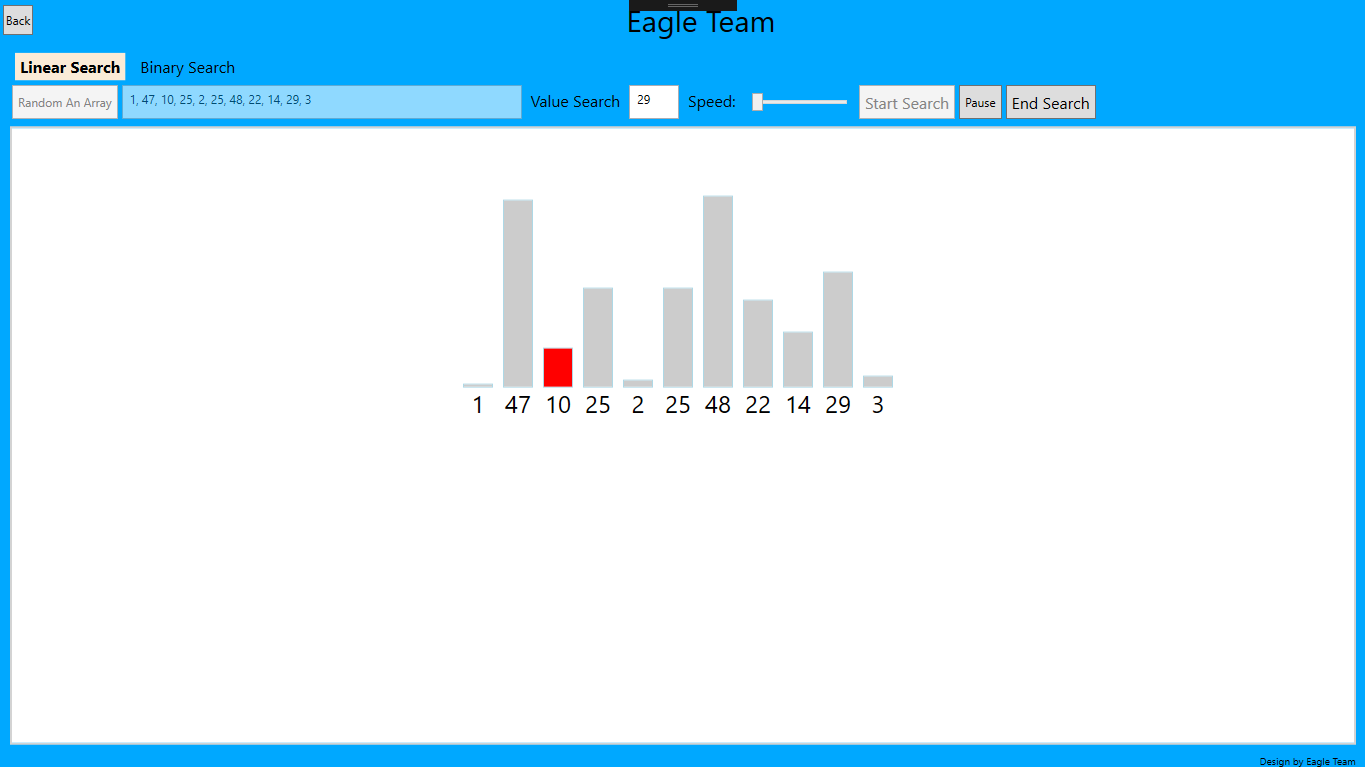


Figure 3: Search Window

* + - * Góc trên màn hình là nút “Back” dùng để quay lại màn hình chính ban đầu.
      * Hai button dùng để chọn thuật toán tìm kiếm mà người dùng muốn thực hiện: “Linear Search” và “Binary Search”.
      * Button “Random An Array” dùng để tạo ngẫu nhiên 1 mảng số không quá 20 phần tử và giá trị mỗi phần tử nằm trong khoảng 1..50. Cùng với đó là khay mảng số.
      * Ô “Value Search” để nhập giá trị cần tìm kiếm.
      * Thanh “Speed” dùng để chỉnh tốc độ mô phỏng, kéo sang bên trái là chậm và kéo sang bên phải là nhanh. Lưu ý: chỉ có thể chỉnh tốc độ khi thuật toán và mô phỏng chưa chạy, không thể chỉnh khi đang chạy mô phỏng.
      * Ba button cuối cùng là trạng thái mô phỏng: “Start Search”, “Pause”, “End Search”.
      * Khi tìm kiếm thành công sẽ cho ra màn hình như thế này:

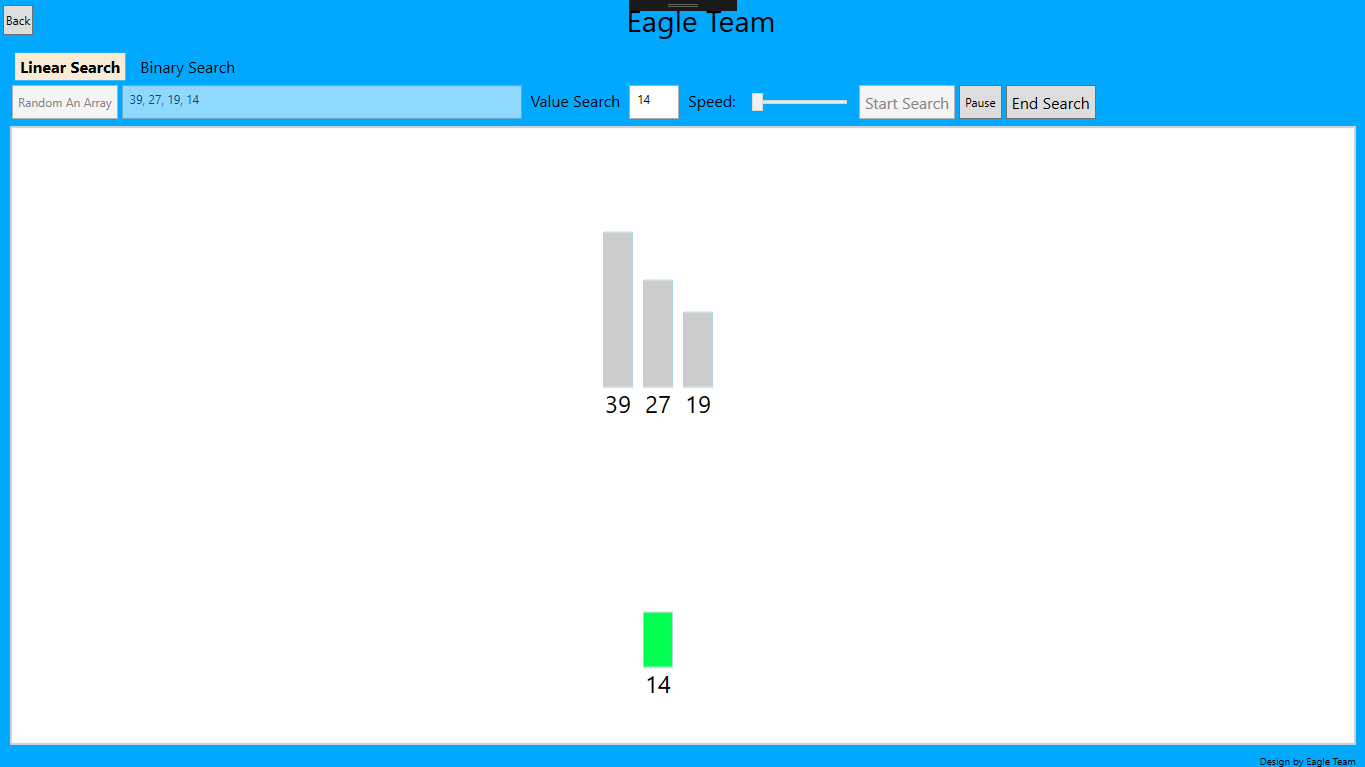


Figure 4: Search Success

* + - * Khi tìm kiếm không thành công sẽ cho ra màn hình như thế này:

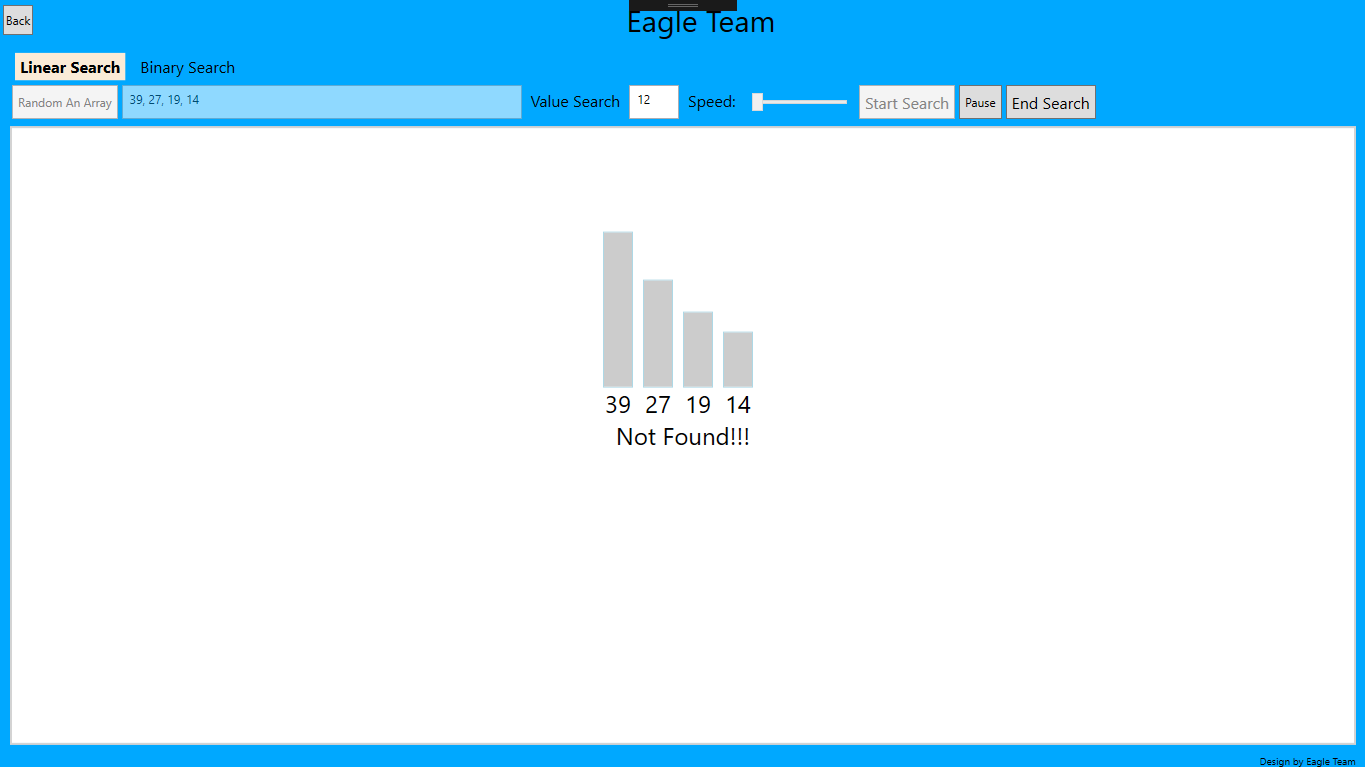


Figure 5: Search Failue

* + Khi chọn button Sort sẽ vào cửa sổ sắp xếp

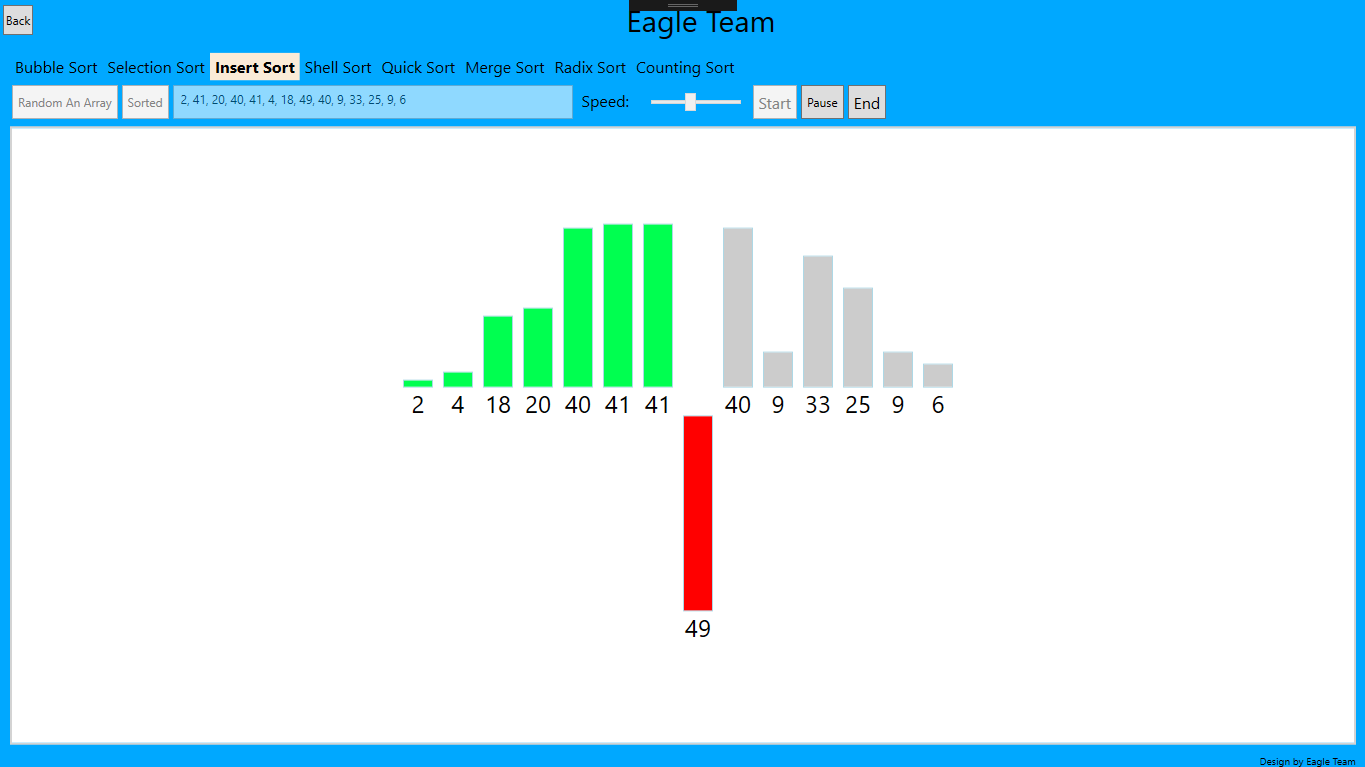


Figure 6: Sort Window

* + - * Các button cũng tương tự như cửa sổ tìm kiếm, một vài button khác như “Sorted”: dùng để sắp xếp mảng số ngay, không qua mô phỏng.
      * Khi sắp xếp xong sẽ cho ra màn hình thế này:

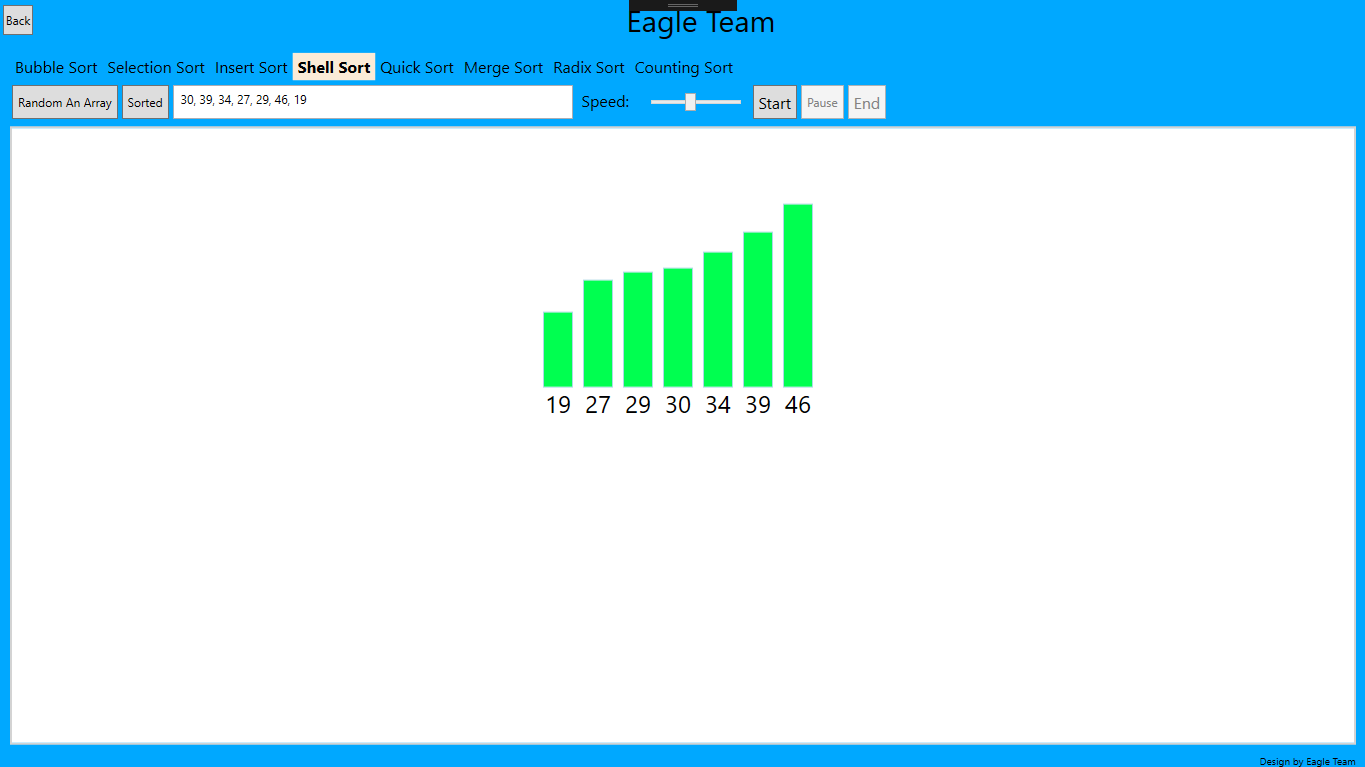


Figure 5: Sort Success

* + - * Tất cả các cột được tô xanh là cột đã đúng theo thứ tự sắp xếp (cột đã khóa).

# CHƯƠNG 3: PHÂN CÔNG VIỆC

## NGUYỄN THIÊN ÂN

* Tạo git đồ án, xây dựng khung đồ án.
* Tìm hiểu thuật toán tìm kiếm và sắp xếp, code và tổ chức dữ liệu lại mỗi thuật toán theo tư duy ngắn gọn bằng ngôn ngữ C#.
  + Code tính năng
* Hỗ trợ việc thêm hiệu ứng vào các thuật toán tìm kiếm và sắp xếp như: tô màu cột đang được chọn, đánh dấu cột, đổi chỗ cột, …
* Kiểm thử ứng dụng.
* Viết báo cáo ứng dụng, hướng dẫn sử dụng, phân công việc.

## HỒ KHẢ VIỆT HUẤN

* Tìm hiểu và dùng thử WPF.
* Thiết kế backrough button SEARCH, SORT, EXIT.
* Viết layout cho đồ án bằng WPF:
  + Code hệ thống WPF, thiết kế các cửa sổ và thứ tự chuyển.
  + Code giao diện, bố cục mỗi cửa sổ, hiệu ứng di chuyển, ẩn hiện, tô màu, …
* Chỉnh sửa ứng dụng sau kiểm thử.

# PHỤ LỤC – TÀI LIỆU THAM KHẢO

[1]: Lê Minh Hoàng – [Giải Thuật Và Lập Trình] Phần 2: Cấu trúc dữ liệu và giải thuật - <https://sachvui.com/ebook/giai-thuat-va-lap-trinh-le-minh-hoang.2238.html>

[2]: Lương Trần Ngọc Khiết – [Cấu Trúc Dữ Liệu]: Slide bài giảng - <https://drive.google.com/drive/folders/1mqxEAXh5KvEYgG6z67z8VIpsoCe3T-xk>