BỘ GIÁO DỤC VÀ ĐÀO TẠO
TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI
PHÂN HIỆU TẠI THÀNH PHỐ HỒ CHÍ MINH

BÁO CÁO ĐỒ ÁN
ĐỀ TÀI:
CHƯƠNG TRÌNH MINH HỌA TỪNG BƯỚC CÁC THUẬT TOÁN TRONG MÔN CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

Giảng viên hướng dẫn: Nguyễn Đình Hiển

Sinh viên thực hiện:

Hoàng Minh Tài	MSSV: 6051071102
Nguyễn Hoàng Hiệp	MSSV: 6051071147
Cao Lâm Bảo Khanh	MSSV: 6051071056
Nguyễn Hữu Đại	MSSV: 6051071023
Trương Được	MSSV: 6051071033

Lớp: CQ.60.CNTT

TP Hồ Chí Minh, năm 2022

BỘ GIÁO DỤC VÀ ĐÀO TẠO
TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI
PHÂN HIỆU TẠI THÀNH PHỐ HỒ CHÍ MINH

BÁO CÁO ĐỒ ÁN

ĐỀ TÀI:

CHƯƠNG TRÌNH MINH HỌA TỪNG BƯỚC CÁC THUẬT TOÁN TRONG MÔN CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

Giảng viên hướng dẫn: Nguyễn Đình Hiển

Sinh viên thực hiện:

Hoàng Minh Tài	MSSV: 6051071102
Nguyễn Hoàng Hiệp	MSSV: 6051071147
Cao Lâm Bảo Khanh	MSSV: 6051071056
Nguyễn Hữu Đại	MSSV: 6051071023
Trương Được	MSSV: 6051071033

Lớp: CQ.60.CNTT

TP Hồ Chí Minh, năm 2022

TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI
PHÂN HIỆU TẠI THÀNH PHỐ HỒ CHÍ MINH

CỘNG HÒA XÃ HỘI CHỦ NGHĨA VIỆT NAM
Độc lập – Tự do – Hạnh phúc

NHIỆM VỤ BÁO CÁO
MÔN: THUẬT TOÁN VÀ ỨNG DỤNG

-------***-------

Họ và tên: Hoàng Minh Tài — MSSV: 6051071102
Họ và tên: Nguyễn Hoàng Hiệp — MSSV: 6051071147
Họ và tên: Cao Lâm Bảo Khanh — MSSV: 6051071056
Họ và tên: Nguyễn Hữu Đại — MSSV: 6051071023
Họ và tên: Trương Được — MSSV: 6051071033
Khóa: 60 — Lớp: CQ.60.CNTT

1. Tên đề tài

   Xây dựng chương trình minh họa từng bước các thuật toán trong môn Cấu trúc dữ liệu và Giải thuật.

2. Mục đích

   Cấu trúc dữ liệu và Giải thuật là môn học quan trọng trong lập trình. Tất cả sinh viên khi theo học công nghệ thông tin sẽ được học cấu trúc dữ liệu và giải thuật. Các vấn đề cần giải quyết trong lập trình đều sử dụng các giải thuật. Những giải thuật cơ bản bao gồm sắp xếp, tìm kiếm, danh sách liên kết, cây nhị phân tìm kiếm… Vì vậy việc minh họa sẽ giúp các sinh viên cũng như ai đang tìm hiểu sẽ hiểu rõ hơn.

3. Các kết quả chính dự kiến sẽ đạt được

   Xây dựng thành công chương trình minh họa từng bước các thuật toán trong môn Cấu trúc dữ liệu và Giải thuật.

4. Giảng viên và cán bộ hướng dẫn

   - Giảng viên: Nguyễn Đình Hiển.
   - Đơn vị công tác: Trường đại học Giao thông Vận tải Phân hiệu tại Thành phố Hồ Chí Minh.

LỜI CẢM ƠN

Lời đầu tiên, em xin gửi lời cảm ơn chân thành đến Trường Đại học Giao Thông Vận Tải - Phân hiệu tại thành phố Hồ Chí Minh đã đưa ngành Công nghệ thông tin vào chương trình đào tạo. Đặc biệt là quý thầy, cô giáo trong Bộ môn Công nghệ thông tin – những người đã dành cả tâm huyết để chỉ dạy và truyền đạt những kiến thức, kinh nghiệm của mình cho chúng em.

Trong những năm học tập tại trường, với những gì thầy cô truyền đạt, bản thân em đã tiếp thu được những kiến thức cơ bản các môn học và ngày càng hiểu rõ về ngành mà em đã lựa chọn. Không những thế, dưới mái trường này, em còn được học những kỹ năng mà có lẽ nó sẽ giúp em không ít trong sự nghiệp tương lai. Để hoàn thành được báo cáo này, em xin bày tỏ lòng biết ơn chân thành và sâu sắc đến thầy Nguyễn Đình Hiển, giảng viên dạy bộ môn Thuật toán và Ứng dụng, người đã trực tiếp hướng dẫn, dìu dắt, giúp đỡ chúng em với những chỉ dẫn khoa học quý giá trong suốt quá trình triển khai, nghiên cứu và hoàn thành báo cáo. Như người ta thường nói, người thầy như một nhà làm vườn, đêm ngày ươm trồng chăm sóc cho hạt giống của mình mong sao chúng có thể lớn nhanh để có ích cho đời. Hạt giống mà thầy cô gieo trồng chính là hạt giống tâm hồn – sự nghiệp trồng người. Cảm ơn thầy đã cho chúng em thứ tài sản vô giá, là hành trang vững chắc để chúng em có thể bước từng bước vào cuộc sống đầy chông gai và thử thách của cuộc sống.

Do kiến thức còn hạn chế và khả năng tiếp thu chưa được hoàn hảo nên chúng em khó tránh được những sai sót trong quá trình làm bài. Mong thầy/cô thông cảm và góp ý thêm cho bài báo cáo nhóm em.

Sau cùng, em xin kính chúc quý thầy cô trong Bộ môn Công nghệ thông tin và toàn thể quý thầy cô đang giảng dạy tại Trường Đại học Giao Thông Vận Tải - Phân hiệu tại thành phố Hồ Chí Minh lời chúc sức khỏe, luôn hạnh phúc và thành công hơn nữa trong công việc cũng như cuộc sống.

Em xin chân thành cảm ơn!

NHẬN XÉT CỦA GIÁO VIÊN

.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....

Tp. Hồ Chí Minh, ngày ... tháng ... năm ...

Giảng viên hướng dẫn

Nguyễn Đình Hiển

MỤC LỤC

Contents

DANH MỤC HÌNH ẢNH

CHƯƠNG I: THUẬT TOÁN VÀ ỨNG DỤNG

1. Bài toán về sắp xếp

Sắp xếp là quá trình bố trí lại các phần tử của một tập đối tượng nào đó theo một thứ tự nhất định. Chẳng hạn như thứ tự tăng dần (hay giảm dần) đối với một dãy số, thứ tự từ điển đối với các từ v.v… Yêu cầu về sắp xếp thường xuyên xuất hiện trong các ứng dụng Tin học với các mục đích khác nhau: sắp xếp dữ liệu trong máy tính để tìm kiếm cho thuận lợi, sắp xếp các kết quả xử lý để in ra trên bảng biểu v.v…

Nói chung, dữ liệu có thể xuất hiện dưới nhiều dạng khác nhau, nhưng ở đây ta quy ước: Một tập các đối tượng cần sắp xếp là tập các bản ghi (records), mỗi bản ghi bao gồm một số trường (fields) khác nhau. Nhưng không phải toàn bộ các trường dữ liệu trong bản ghi đều được xem xét đến trong quá trình sắp xếp mà chỉ là một trường nào đó (hay một vài trường nào đó) được chú ý tới thôi. Trường như vậy ta gọi là khoá (key). Sắp xếp sẽ được tiến hành dựa vào giá trị của khoá này.

Ví dụ: Hồ sơ tuyển sinh của một trường Đại học là một danh sách thí sinh, mỗi thí sinh có tên, số báo danh, điểm thi. Khi muốn liệt kê danh sách những thí sinh trúng tuyển tức là phải sắp xếp các thí sinh theo thứ tự từ điểm cao nhất tới điểm thấp nhất. Ở đây khoá sắp xếp chính là điểm thi.

STT	SBD	Họ và tên	Điểm thi
1	A100	Nguyễn Văn A	20
2	B200	Trần Thị B	25
3	X150	Phạm Văn C	18
4	G180	Đỗ Thị D	21

Khi sắp xếp, các bản ghi trong bảng sẽ được đặt lại vào các vị trí sao cho giá trị khoá tương ứng với chúng có đúng thứ tự đã ấn định. Vì kích thước của toàn bản ghi có thể rất lớn, nên nếu việc sắp xếp thực hiện trực tiếp trên các bản ghi sẽ đòi hỏi sự chuyển đổi vị trí của các bản ghi, kéo theo việc thường xuyên phải di chuyển, copy những vùng nhớ lớn, gây ra những tổn phí thời gian khá nhiều. Thường người ta khắc phục tình trạng này bằng cách xây dựng một bảng khoá: Mỗi bản ghi trong bảng ban đầu sẽ tương ứng với một bản ghi trong bảng khoá. Bảng khoá cũng gồm các bản ghi nhưng mỗi bản ghi chỉ gồm có hai trường:

Trường thứ nhất chứa khoá

Trường thứ hai chứa liên kết tới một bản ghi trong bảng ban đầu, tức là chứa một thông tin đủ để biết bản ghi tương ứng với nó trong bảng ban đầu là bản ghi nào.

Sau đó, việc sắp xếp được thực hiện trực tiếp trên bảng khoá, trong quá trình sắp xếp, bảng chính không hề bị ảnh hưởng gì, việc truy cập vào một bản ghi nào đó của bảng chính vẫn có thể thực hiện được bằng cách dựa vào trường liên kết của bản ghi tương ứng thuộc bảng khoá.

Như ở ví dụ trên, ta có thể xây dựng bảng khoá gồm 2 trường, trường khoá chứa điểm và trường liên kết chứa số thứ tự của người có điểm tương ứng trong bảng ban đầu:

Điểm thi	STT
20	1
25	2
18	3
21	4

Sau khi sắp xếp theo trật tự điểm cao nhất tới điểm thấp nhất, bảng khoá sẽ trở thành:

Điểm thi	STT
25	2
21	4
20	1
18	3

Dựa vào bảng khoá, ta có thể biết được rằng người có điểm cao nhất là người mang số thứ tự 2, tiếp theo là người mang số thứ tự 4, tiếp nữa là người mang số thứ tự 1, và cuối cùng là người mang số thứ tự 3, còn muốn liệt kê danh sách đầy đủ thì ta chỉ việc đối chiếu với bảng ban đầu và liệt kê theo thứ tự 2, 4, 1, 3.

Có thể còn cải tiến tốt hơn dựa vào nhận xét sau: Trong bảng khoá, nội dung của trường khoá hoàn toàn có thể suy ra được từ trường liên kết bằng cách: Dựa vào trường liên kết, tìm tới bản ghi tương ứng trong bảng chính rồi truy xuất trường khoá trong bảng chính. Như ví dụ trên thì người mang số thứ tự 1 chắc chắn sẽ phải có điểm thi là 20, còn người mang số thứ tự 3 thì chắc chắn phải có điểm thi là 18. Vậy thì bảng khoá có thể loại bỏ đi trường khoá mà chỉ giữ lại trường liên kết. Trong trường hợp các phần tử trong bảng ban đầu được đánh số từ 1 tới n và trường liên kết chính là số thứ tự của bản ghi trong bảng ban đầu như ở ví dụ trên, người ta gọi kỹ thuật này là kỹ thuật sắp xếp bằng chỉ số: Bảng ban đầu không hề bị ảnh hưởng gì cả, việc sắp xếp chỉ đơn thuần là đánh lại chỉ số cho các bản ghi theo thứ tự sắp xếp.

2. Thuật toán sắp xếp
   a. Bubble sort

Tư tưởng:

- Xuất phát từ cuối dãy, đổi chỗ các cặp phần tử kế cận để đưa phần tử nhỏ hơn trong cặp phần tử đó về vị trí đầu dãy hiện hành, sau đó sẽ không xét đến nó ở bước tiếp theo.
- Ở lần xử lý thứ i có vị trí đầu dãy là i
- Lặp lại xử lý trên cho đến khi không còn cặp phần tử nào để xét

Trong thuật toán sắp xếp nổi bọt, dãy các khoá sẽ được duyệt từ cuối dãy lên đầu dãy (từ kn về k1), nếu gặp hai khoá kế cận bị ngược thứ tự thì đổi chỗ của chúng cho nhau. Sau lần duyệt như vậy, phần tử nhỏ nhất trong dãy khoá sẽ được chuyển về vị trí đầu tiên và vấn đề trở thành sắp xếp dãy khoá từ k2 tới kn:

```
procedure BubbleSort;
var
  i, j: Integer;
begin
  for i := 2 to n do
    for j := n downto i do {Duyệt từ cuối dãy lên, làm nổi khoá nhỏ nhất trong số k_{i-1}, …, k_n về vị trí i-1}
      if k_j < k_{j-1} then
        <Đảo giá trị k_j và k_{j-1}>
end;
```

Hình 1. Mã giả Bubble sort

Đối với thuật toán sắp xếp nổi bọt, ta có thể coi phép toán tích cực là phép so sánh kj < kj-1. Và số lần thực hiện phép so sánh này là:

$$(n - 1) + (n - 2) + \ldots + 1 = n * (n - 1) / 2$$

Vậy thuật toán sắp xếp nổi bọt cũng có cấp là O(n^2). Bất kể tình trạng dữ liệu vào như thế nào.

b. Insertion sort

Xét dãy khoá k1, k2, …, kn. Ta thấy dãy con chỉ gồm mỗi một khoá là k1 có thể coi là đã sắp xếp rồi. Xét thêm k2, ta so sánh nó với k1, nếu thấy k2 < k1 thì chèn nó vào trước k1. Đối với k3, ta lại xét dãy chỉ gồm 2 khoá k1, k2 đã sắp xếp và tìm cách chèn k3 vào dãy khoá đó để được thứ tự sắp xếp. Một cách tổng quát, ta sẽ sắp xếp dãy k1, k2, …, ki trong điều kiện dãy k1, k2, …, ki-1 đã sắp xếp rồi bằng cách chèn ki vào dãy đó tại vị trí đúng khi sắp xếp.

```
procedure InsertionSort;
var
  i, j: Integer;
  tmp: TKey; {Biến giữ lại giá trị khoá chèn}
begin
  for i := 2 to n do {Chèn giá trị ki vào dãy k1,…, ki-1 để toàn đoạn k1, k2,…, ki trở thành đã sắp xếp}
    begin
      tmp := ki; {Giữ lại giá trị ki}
      j := i - 1;
      while (j > 0) and (tmp < kj) do {So sánh giá trị cần chèn với lần lượt các khoá kj (i-1≥j≥0)}
        begin
          kj+1 := kj; {Đẩy lùi giá trị kj về phía sau một vị trí, tạo ra "khoảng trống" tại vị trí j}
          j := j - 1;
        end;
      kj+1 := tmp; {Đưa giá trị chèn vào "khoảng trống" mới tạo ra}
    end;
end;
```

Hình 2. Mã giả Insertion sort

Đối với thuật toán sắp xếp kiểu chèn, thì chi phí thời gian thực hiện thuật toán phụ thuộc vào tình trạng dãy khoá ban đầu. Nếu coi phép toán tích cực ở đây là phép so sánh tmp < kj thì: Trường hợp tốt nhất ứng với dãy khoá đã sắp xếp rồi, mỗi lượt chỉ cần 1 phép so sánh, và như vậy tổng số phép so sánh được thực hiện là n - 1.

Trường hợp tồi tệ nhất ứng với dãy khoá đã có thứ tự ngược với thứ tự cần sắp thì ở lượt thứ i, cần có i - 1 phép so sánh và tổng số phép so sánh là:

$$(n - 1) + (n - 2) + \ldots + 1 = n * (n - 1) / 2.$$

Trường hợp các giá trị khoá xuất hiện một cách ngẫu nhiên, ta có thể coi xác suất xuất hiện mỗi khoá là đồng khả năng, thì có thể coi ở lượt thứ i, thuật toán cần trung bình i / 2 phép so sánh và tổng số phép so sánh là:

$$(1 / 2) + (2 / 2) + \ldots + (n / 2) = (n + 1) * n / 4.$$

Nhìn về kết quả đánh giá, ta có thể thấy rằng thuật toán sắp xếp kiểu chèn tỏ ra tốt hơn so với thuật toán sắp xếp chọn và sắp xếp nổi bọt. Tuy nhiên, chi phí thời gian thực hiện của thuật toán sắp xếp kiểu chèn vẫn còn khá lớn. Và xét trên phương diện tính toán lý thuyết thì cấp của thuật toán sắp xếp kiểu chèn vẫn là O(n^2).

```
procedure InsertionSortwithBinarySearching;
var
  i, inf, sup, median: Integer;
  tmp: TKey;
begin
  for i := 2 to n do
    begin
      tmp := ki; {Giữ lại giá trị ki}
      inf := 1; sup := i - 1; {Tìm chỗ chèn giá trị tmp vào đoạn từ kinf tới ksup+1}
      repeat {Sau mỗi vòng lặp này thì đoạn tìm bị co lại một nửa}
        median := (inf + sup) div 2; {Xét chỉ số nằm giữa chỉ số inf và chỉ số sup}
        if tmp<k[median] then sup:=median-1
        else inf := median + 1;
      until inf > sup; {Kết thúc vòng lặp thì inf = sup + 1 chính là vị trí chèn}
      <Dịch các phần tử từ kinf tới ki-1 lùi sau một vị trí>
      kinf := tmp; {Đưa giá trị tmp vào "khoảng trống" mới tạo ra}
    end;
end;
```

Hình 3. Insertion sort cải tiến

c. Selection sort

Thuật toán selection sort sắp xếp một mảng bằng cách đi tìm phần tử có giá trị nhỏ nhất(giả sử với sắp xếp mảng tăng dần) trong đoạn đoạn chưa được sắp xếp và đổi cho phần tử nhỏ nhất đó với phần tử ở đầu đoạn chưa được sắp xếp(không phải đầu mảng).

Thuật toán sẽ chia mảng làm 2 mảng con

- Một mảng con đã được sắp xếp
- Một mảng con chưa được sắp xếp

Tại mỗi bước lặp của thuật toán, phần tử nhỏ nhất ở mảng con chưa được sắp xếp sẽ được di chuyển về đoạn đã sắp xếp.

Ý tưởng cơ bản của cách sắp xếp này là:

Ở lượt thứ nhất, ta chọn trong dãy khoá k1, k2, …, kn ra khoá nhỏ nhất (khoá <= mọi khoá khác) và đổi giá trị của nó với k1, khi đó giá trị khoá k1 trở thành giá trị khoá nhỏ nhất.

Ở lượt thứ hai, ta chọn trong dãy khoá k2, …, kn ra khoá nhỏ nhất và đổi giá trị của nó với k2.

Ở lượt thứ i, ta chọn trong dãy khoá ki, ki+1, …, kn ra khoá nhỏ nhất và đổi giá trị của nó với ki.

Làm tới lượt thứ n - 1, chọn trong hai khoá kn-1, kn ra khoá nhỏ nhất và đổi giá trị của nó với kn-1.

```
procedure SelectionSort;
var
  i,j,jmin:Integer;
begin
  for i := 1 to n - 1 do {Làm n - 1 lượt}
    begin
      {Chọn trong số các khoá từ ki tới kn ra khoá kjmin nhỏ nhất}
      jmin := i;
      for j := i + 1 to n do
        if kj<kjmin then jmin:=j;
      if jmin ≠ i then
        <Đảo giá trị của kjmin cho ki>
    end;
end;
```

Hình 4. Mã giả selection sort

Đối với phương pháp kiểu lựa chọn, ta có thể coi phép so sánh (kj < kjmin) là phép toán tích cực để đánh giá hiệu suất thuật toán về mặt thời gian. Ở lượt thứ i, để chọn ra khoá nhỏ nhất bao giờ cũng cần n - i phép so sánh, số lượng phép so sánh này không hề phụ thuộc gì vào tình trạng ban đầu của dãy khoá cả. Từ đó suy ra tổng số phép so sánh sẽ phải thực hiện là:

$$(n - 1) + (n - 2) + \ldots + 1 = n * (n - 1) / 2$$

Vậy thuật toán sắp xếp kiểu chọn có cấp là O(n^2).

d. Quick sort

QuickSort là một phương pháp sắp xếp tốt nhất, nghĩa là dù dãy khoá thuộc kiểu dữ liệu có thứ tự nào, QuickSort cũng có thể sắp xếp được và không có một thuật toán sắp xếp nào nhanh hơn QuickSort về mặt tốc độ trung bình (theo tôi biết). Người sáng lập ra nó là C.A.R. Hoare đã mạnh dạn đặt tên cho nó là sắp xếp "NHANH".

Ý tưởng chủ đạo của phương pháp có thể tóm tắt như sau: Sắp xếp dãy khoá k1, k2, …, kn thì có thể coi là sắp xếp đoạn từ chỉ số 1 tới chỉ số n trong dãy khoá đó. Để sắp xếp một đoạn trong dãy khoá, nếu đoạn đó có <= 1 phần tử thì không cần phải làm gì cả, còn nếu đoạn đó có ít nhất 2 phần tử, ta chọn một khoá ngẫu nhiên nào đó của đoạn làm "chốt" (pivot). Mọi khoá nhỏ hơn khoá chốt được xếp vào vị trí đứng trước chốt, mọi khoá lớn hơn khoá chốt được xếp vào vị trí đứng sau chốt. Sau phép hoán chuyển như vậy thì đoạn đang xét được chia làm hai đoạn khác rỗng mà mọi khoá trong đoạn đầu đều <= chốt và mọi khoá trong đoạn sau đều >= chốt. Hay nói cách khác: Mỗi khoá trong đoạn đầu đều <= mọi khoá trong đoạn sau. Và vấn đề trở thành sắp xếp hai đoạn mới tạo ra (có độ dài ngắn hơn đoạn ban đầu) bằng phương pháp tương tự.

```
procedure QuickSort;

  procedure Partition(L, H: Integer); {Sắp xếp đoạn từ kL, kL+1, …, kH}
  var
    i, j: Integer;
    Pivot: TKey; {Biến lưu giá trị khoá chốt}
  begin
    if L≥H then Exit; {Nếu đoạn chỉ có ≤ 1 phần tử thì không phải làm gì cả}
    Pivot :=kRandom(H-L+1)+L; {Chọn một khoá ngẫu nhiên trong đoạn làm khoá chốt}
    i := L; j := H; {i := vị trí đầu đoạn; j := vị trí cuối đoạn}
    repeat
      while ki < Pivot do i := i + 1; {Tìm từ đầu đoạn khoá ≥khoá chốt}
      while kj > Pivot do j := j - 1; {Tìm từ cuối đoạn khoá ≤khoá chốt}
      {Đến đây ta tìm được hai khoá ki và kj mà ki ≥ key ≥ kj}
      if i≤j then
        begin
          if i < j then {Nếu chỉ số i đứng trước chỉ số j thì đảo giá trị hai khoá ki và kj}
            <Đảo giá trị ki và kj> {Sau phép đảo này ta có: ki ≤key ≤kj }
          i := i + 1; j := j - 1;
        end;
    until i>j;
    Partition(L, j); Partition(i, H); {Sắp xếp hai đoạn con mới tạo ra}
  end;

begin
  Partition(1, n);
end;
```

Hình 5. Mã giả quick sort

Ta thử phân tích xem tại sao đoạn chương trình trên hoạt động đúng: Xét vòng lặp repeat…until trong lần lặp đầu tiên, vòng lặp while thứ nhất chắc chắn sẽ tìm được khoá ki >= khoá chốt bởi chắc chắn tồn tại trong đoạn một khoá bằng khóa chốt. Tương tự như vậy, vòng lặp while thứ hai chắc chắn tìm được khoá kj <= khoá chốt. Nếu như khoá ki đứng trước khoá kj thì ta đảo giá trị hai khoá, cho i tiến và j

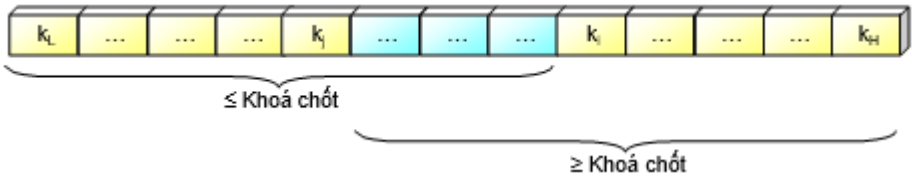
lùi. Khi đó ta có nhận xét rằng mọi khoá đứng trước vị trí i sẽ phải <= khoá chốt và mọi khoá đứng sau vị trí j sẽ phải >= khoá chốt.

Hình 6. Vòng lặp trong của quick sort

Điều này đảm bảo cho vòng lặp repeat…until tại bước sau, hai vòng lặp while…do bên trong chắc chắn lại tìm được hai khoá ki và kj mà ki >= khoá chốt >= kj, nếu khoá ki đứng trước khoá kj thì lại đảo giá trị của chúng, cho i tiến về cuối một bước và j lùi về đầu một bước. Vậy thì quá trình hoán chuyển phần tử trong vòng lặp repeat…until sẽ đảm bảo tại mỗi bước:

- Hai vòng lặp while…do bên trong luôn tìm được hai khoá ki, kj mà ki >= khoá chốt >= kj. Không có trường hợp hai chỉ số i, j chạy ra ngoài đoạn (luôn luôn có L <= i, j <= H).
- Sau mỗi phép hoán chuyển, mọi khoá đứng trước vị trí i luôn <= khoá chốt và mọi khoá đứng sau vị trí j luôn >= khoá chốt.

Vòng lặp repeat …until sẽ kết thúc khi mà chỉ số i đứng phía sau chỉ số j.



Hình 7. Trạng thái trước khi gọi đệ quy

Xét về độ phức tạp tính toán:

Trường hợp tồi tệ nhất, là khi chọn khoá chốt, ta chọn phải khoá nhỏ nhất hay lớn nhất trong đoạn, khi đó phép phân đoạn sẽ chia thành một đoạn gồm n - 1 phần tử và đoạn còn lại chỉ có 1 phần tử. Có thể chứng minh trong trường hợp này, thời gian thực hiện giải thuật T(n) = O(n^2) Trường hợp tốt nhất, phép phân đoạn tại mỗi bước sẽ chia được thành hai đoạn bằng nhau. Tức là khi chọn khoá chốt, ta chọn đúng trung vị của dãy khoá. Có thể chứng minh trong trường hợp này, thời gian thực hiện giải thuật T(n) = O(nlog2n).

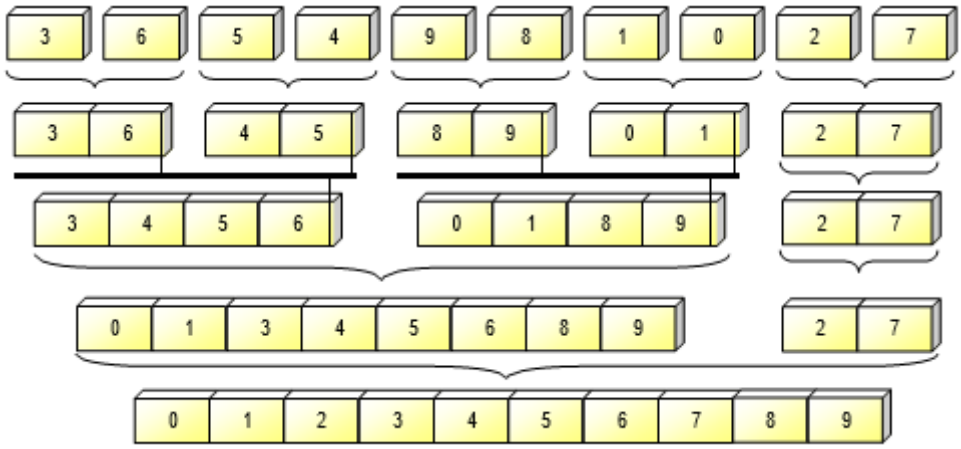
Trường hợp các khoá được phân bố ngẫu nhiên, thì trung bình thời gian thực hiện giải thuật cũng là T(n) = O(nlog2n).

Việc tính toán chi tiết, đặc biệt là khi xác định T(n) trung bình, phải dùng các công cụ toán phức tạp, ta chỉ công nhận những kết quả trên.

e. Merge sort

Phép trộn 2 đường là phép hợp nhất hai dãy khoá đã sắp xếp để ghép lại thành một dãy khoá có kích thước bằng tổng kích thước của hai dãy khoá ban đầu và dãy khoá tạo thành cũng có thứ tự sắp xếp. Nguyên tắc thực hiện của nó khá đơn giản: so sánh hai khoá đứng đầu hai dãy, chọn ra khoá nhỏ nhất và đưa nó vào miền sắp xếp (một dãy khoá phụ có kích thước bằng tổng kích thước hai dãy khoá ban đầu) ở vị trí thích hợp. Sau đó, khoá này bị loại ra khỏi dãy khoá chứa nó. Quá trình tiếp tục cho tới khi một trong hai dãy khoá đã cạn, khi đó chỉ cần chuyển toàn bộ dãy khoá còn lại ra miền sắp xếp là xong.

Ta có thể coi mỗi khoá trong dãy khoá k1, k2, …, kn là một mạch với độ dài 1, dĩ nhiên các mạch độ dài 1 có thể coi là đã được sắp. Nếu trộn hai mạch liên tiếp lại thành một mạch có độ dài 2, ta lại được dãy gồm các mạch đã được sắp. Cứ tiếp tục như vậy, số mạch trong dãy sẽ giảm dần sau mỗi lần trộn.



Hình 8. Thuật toán sắp xếp trộn

Về độ phức tạp của thuật toán, ta thấy rằng trong thủ tục Merge, phép toán tích cực là thao tác đưa một khoá vào miền sắp xếp. Mỗi lần gọi thủ tục MergeByLength, tất cả các phần tử trong dãy khoá được chuyển hoàn toàn sang miền sắp xếp, nên độ phức tạp của thủ tục MergeByLength là O(n). Thủ tục MergeSort có vòng lặp thực hiện không quá log2n + 1 lời gọi MergeByLength bởi biến len sẽ được tăng theo cấp số nhân công bội 2. Từ đó suy ra độ phức tạp của MergeSort là O(nlog2n) bất chấp trạng thái dữ liệu vào.

Cùng là những thuật toán sắp xếp tổng quát với độ phức tạp trung bình như nhau, nhưng không giống như QuickSort hay HeapSort, MergeSort có tính ổn định. Nhược điểm của MergeSort là nó phải dùng thêm một vùng nhớ để chứa dãy khoá phụ có kích thước bằng dãy khoá ban đầu.

Người ta còn có thể lợi dụng được trạng thái dữ liệu vào để khiến MergeSort chạy nhanh hơn: ngay từ đầu, ta không coi mỗi phần tử của dãy khoá là một mạch mà coi những đoạn đã được sắp trong dãy khoá là một mạch. Bởi một dãy khoá bất kỳ có

thể coi là gồm các mạch đã sắp xếp nằm liên tiếp nhau. Khi đó người ta gọi phương pháp này là phương pháp trộn hai đường tự nhiên.

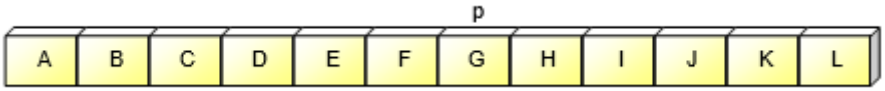
Tổng quát hơn nữa, thay vì phép trộn hai mạch, người ta có thể sử dụng phép trộn k mạch, khi đó ta được thuật toán sắp xếp trộn k đường.

3. Danh sách liên kết

Khi cài đặt danh sách bằng một mảng, thì có một biến nguyên n lưu số phần tử hiện có trong danh sách. Nếu mảng được đánh số bắt đầu từ 1 thì các phần tử trong danh sách được cất giữ trong mảng bằng các phần tử được đánh số từ 1 tới n.

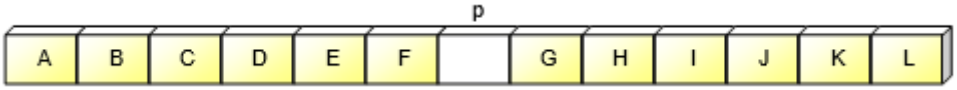
Chèn phần tử vào mảng:

Mảng ban đầu:

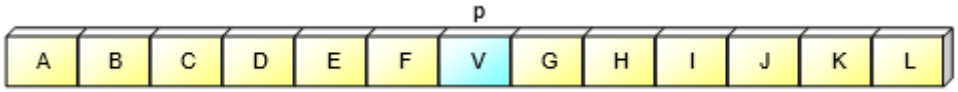


Nếu muốn chèn một phần tử V vào mảng tại vị trí p, ta phải:

Dồn tất cả các phần tử từ vị trí p tới tới vị trí n về sau một vị trí:



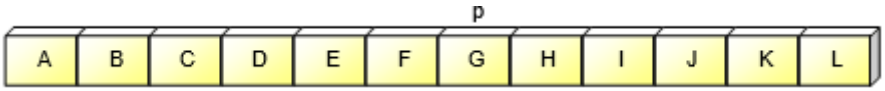
Đặt giá trị V vào vị trí p:



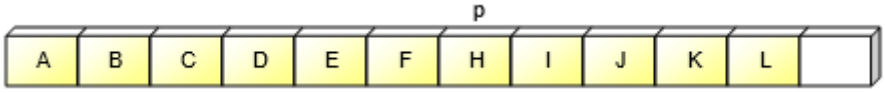
Tăng n lên 1

Xoá phần tử khỏi mảng

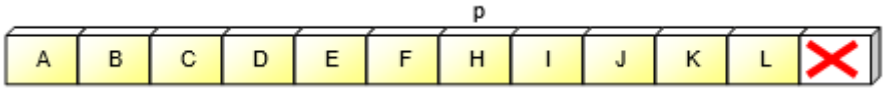
Mảng ban đầu:



Muốn xoá phần tử thứ p của mảng mà vẫn giữ nguyên thứ tự các phần tử còn lại, ta phải: Dồn tất cả các phần tử từ vị trí p + 1 tới vị trí n lên trước một vị trí:



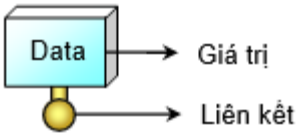
Giảm n đi 1



Trong trường hợp cần xóa một phần tử mà không cần duy trì thứ tự của các phần tử khác, ta chỉ cần đảo giá trị của phần tử cần xóa cho phần tử cuối cùng rồi giảm số phần tử của mảng (n) đi 1.

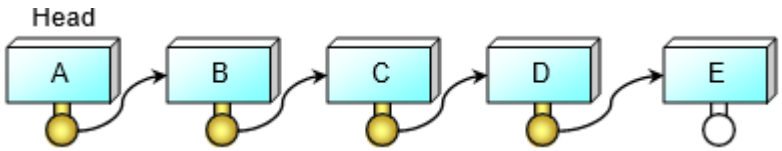
Cài đặt bằng danh sách liên kết đơn

Danh sách nối đơn gồm các nút được nối với nhau theo một chiều. Mỗi nút là một bản ghi (record) gồm hai trường:
Trường thứ nhất chứa giá trị lưu trong nút đó
Trường thứ hai chứa liên kết (con trỏ) tới nút kế tiếp, tức là chứa một thông tin đủ để biết nút kế tiếp nút đó trong danh sách là nút nào, trong trường hợp là nút cuối cùng (không có nút kế tiếp), trường liên kết này được gán một giá trị đặc biệt.



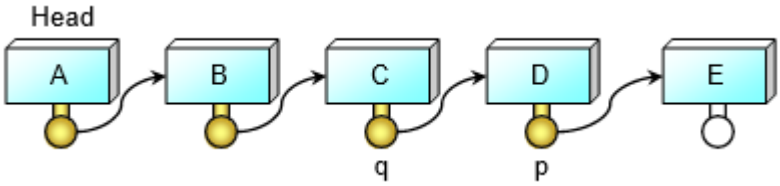
Hình 9. Cấu trúc nút của danh sách liên kết đơn

Nút đầu tiên trong danh sách được gọi là chốt của danh sách nối đơn (Head). Để duyệt danh sách nối đơn, ta bắt đầu từ chốt, dựa vào trường liên kết để đi sang nút kế tiếp, đến khi gặp giá trị đặc biệt (duyệt qua nút cuối) thì dừng lại.

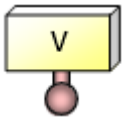


Hình 10. Danh sách liên kết đơn

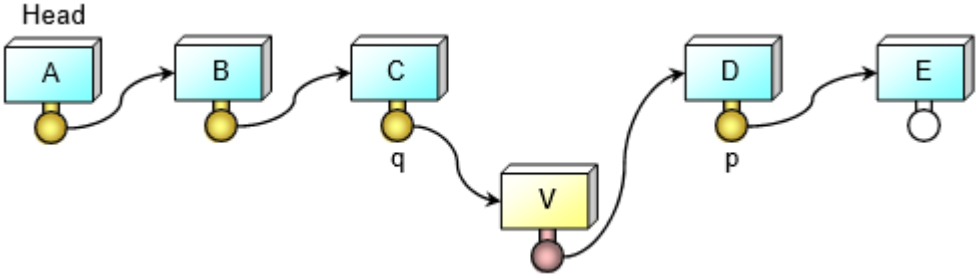
Chèn phần tử vào danh sách nối đơn:
Danh sách ban đầu:



Muốn chèn thêm một nút chứa giá trị V vào vị trí của nút p, ta phải:
Tạo ra một nút mới NewNode chứa giá trị V:



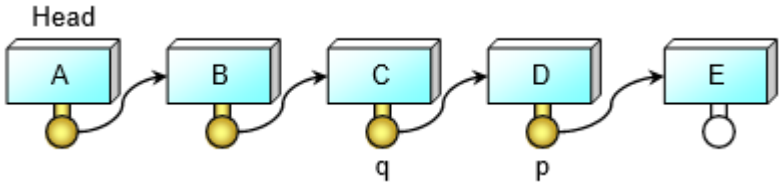
Tìm nút q là nút đứng trước nút p trong danh sách (nút có liên kết tới p).
Nếu tìm thấy thì chỉnh lại liên kết: q liên kết tới NewNode, NewNode liên kết tới p.



Nếu không có nút đứng trước nút p trong danh sách thì tức là p = Head, ta chỉnh lại liên kết: NewNode liên kết tới Head (cũ) và đặt lại Head = NewNode

Xoá phần tử khỏi danh sách nối đơn:

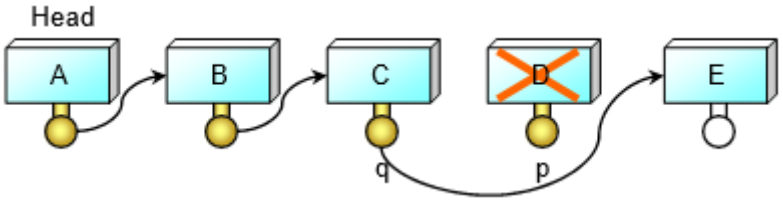
Danh sách ban đầu:



Muốn huỷ nút p khỏi danh sách nối đơn, ta phải:

Tìm nút q là nút đứng liền trước nút p trong danh sách (nút có liên kết tới p)

Nếu tìm thấy thì chỉnh lại liên kết: q liên kết thẳng tới nút liền sau p, khi đó quá trình duyệt danh sách bắt đầu từ Head khi duyệt tới q sẽ nhảy qua không duyệt p nữa. Trên thực tế khi cài đặt bằng các biến động và con trỏ, ta nên có thao tác giải phóng bộ nhớ đã cấp cho nút p.

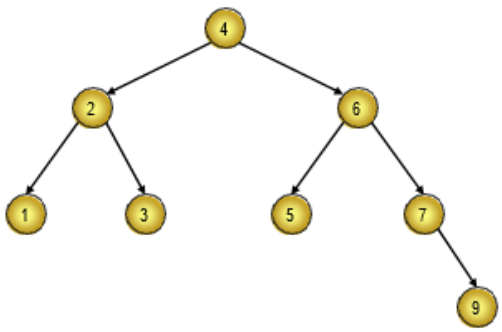


Nếu không có nút đứng trước nút p trong danh sách thì tức là p = Head, ta chỉ việc đặt lại Head bằng nút đứng kế tiếp Head (cũ) trong danh sách. Sau đó có thể giải phóng bộ nhớ cấp cho nút p (Head cũ).

4. Cây nhị phân tìm kiếm

Cho n khoá k1, k2, …, kn, trên các khoá có quan hệ thứ tự toàn phần. Cây nhị phân tìm kiếm ứng với dãy khoá đó là một cây nhị phân mà mỗi nút chứa giá trị một khoá trong n khoá đã cho, hai giá trị chứa trong hai nút bất kỳ là khác nhau. Đối với mọi nút trên cây, tính chất sau luôn được thoả mãn:

- Mọi khoá nằm trong cây con trái của nút đó đều nhỏ hơn khoá ứng với nút đó.
- Mọi khoá nằm trong cây con phải của nút đó đều lớn hơn khoá ứng với nút đó.



Hình 11. Cây nhị phân tìm kiếm

Thuật toán tìm kiếm trên cây có thể mô tả chung như sau:
Trước hết, khoá tìm kiếm X được so sánh với khoá ở gốc cây, và 4 tình huống có thể xảy ra: Không có gốc (cây rỗng): X không có trên cây, phép tìm kiếm thất bại.
X trùng với khoá ở gốc: Phép tìm kiếm thành công.
X nhỏ hơn khoá ở gốc, phép tìm kiếm được tiếp tục trong cây con trái của gốc với cách làm tương tự.
X lớn hơn khoá ở gốc, phép tìm kiếm được tiếp tục trong cây con phải của gốc với cách làm tương tự.

```
{Hàm tìm kiếm trên BST, nó trả về nút chứa khoá tìm kiếm X nếu tìm thấy, trả về nil nếu không tìm thấy}
function BSTSearch(X: TKey): PNode;
var
  p: PNode;
begin
  p := Root; {Bắt đầu với nút gốc}
  while p ≠nil do
    if X=p^.Info then Break;
    else
      if X<p^.Info then p :=p^.Left
      else p :=p^.Right;
  BSTSearch := p;
end;
```

Thuật toán dựng cây nhị phân tìm kiếm từ dãy khoá k1, k2, …, kn cũng được làm gần giống quá trình tìm kiếm. Ta chèn lần lượt các khoá vào cây, trước khi chèn, ta tìm xem khoá đó đã có trong cây hay chưa, nếu đã có rồi thì bỏ qua, nếu nó chưa có thì ta thêm nút mới chứa khoá cần chèn và nối nút đó vào cây nhị phân tìm kiếm.
Trường hợp trung bình, thì các thao tác tìm kiếm, chèn, xoá trên BST có độ phức tạp là O(log2n). Còn trong trường hợp xấu nhất, cây nhị phân tìm kiếm bị suy biến thì các thao tác đó đều có độ phức tạp là O(n), với n là số nút trên cây BST.
Nếu ta mở rộng hơn khái niệm cây nhị phân tìm kiếm như sau: Giá trị lưu trong một nút lớn hơn hoặc bằng các giá trị lưu trong cây con trái và nhỏ hơn các giá trị lưu trong cây con phải. Thì chỉ cần sửa đổi thủ tục BSTInsert một chút, khi chèn lần lượt vào cây n giá trị, cây BST sẽ có n nút (có thể có hai nút chứa cùng một giá trị). Khi đó nếu ta duyệt các nút của cây theo kiểu trung thứ tự (inorder traversal), ta sẽ liệt kê được các giá trị lưu trong cây theo thứ tự tăng dần. Phương pháp sắp xếp này người ta gọi là Tree Sort. Độ phức tạp tính toán trung bình của Tree Sort là O(nlog2n).
Phép tìm kiếm trên cây BST sẽ kém hiệu quả nếu như cây bị suy biến, người ta có nhiều cách xoay xở để tránh trường hợp này. Đó là phép quay cây để dựng cây nhị phân cân đối AVL, hay kỹ thuật dựng cây nhị phân tìm kiếm tối ưu. Những kỹ thuật này ta có thể tham khảo trong các tài liệu khác về cấu trúc dữ liệu và giải thuật.

CHƯƠNG II: ĐỘ PHỨC TẠP THUẬT TOÁN

1. Sự cần thiết phải phân tích thuật toán

Trong khi giải một bài toán chúng ta có thể có một số giải thuật khác nhau, vấn đề là cần phải đánh giá các giải thuật đó để lựa chọn một giải thuật tốt (nhất). Thông thường thì ta sẽ căn cứ vào các tiêu chuẩn sau:

- Giải thuật đúng đắn
- Giải thuật đơn giản
- Giải thuật thực hiện nhanh

Với yêu cầu (1), để kiểm tra tính đúng đắn của giải thuật chúng ta có thể cài đặt giải thuật đó và cho thực hiện trên máy với một số bộ dữ liệu mẫu rồi lấy kết quả thu được so sánh với kết quả đã biết. Thực ra thì cách làm này không chắc chắn bởi vì có thể giải thuật đúng với tất cả các bộ dữ liệu chúng ta đã thử nhưng lại sai với một bộ dữ liệu nào đó. Vả lại cách làm này chỉ phát hiện ra giải thuật sai chứ chưa chứng minh được là nó đúng. Tính đúng đắn của giải thuật cần phải được chứng minh bằng toán học. Tất nhiên điều này không đơn giản và do vậy chúng ta sẽ không đề cập đến ở đây.

Khi chúng ta viết một chương trình để sử dụng một vài lần thì y ê u cầu (2) là quan trọng nhất. Chúng ta cần một giải thuật dễ viết chương trình để nhanh chóng có được kết quả, thời gian thực hiện chương trình không được đề cao vì dù sao thì chương trình đó cũng chỉ sử dụng một vài lần mà thôi.

Tuy nhiên khi một chương trình được sử dụng nhiều lần thì thì yêu cầu tiết kiệm thời gian thực hiện chương trình lại rất quan trọng đặc biệt đối với những chương trình mà khi thực hiện cần dữ liệu nhập lớn do đó yêu cầu (3) sẽ được xem xét một cách kĩ càng. Ta gọi nó là hiệu quả thời gian thực hiện của giải thuật.

2. Thời gian thực hiện của chương trình

Một phương pháp để xác định hiệu quả thời gian thực hiện của một giải thuật là lập trình nó và đo lường thời gian thực hiện của hoạt động trên một máy tính xác định đối với tập hợp được chọn lọc các dữ liệu vào. Thời gian thực hiện không chỉ phụ thuộc vào giải thuật mà còn phụ thuộc vào tập các chỉ thị của máy tính, chất lượng của máy tính và kĩ xảo của người lập trình. Sự thi hành cũng có thể điều chỉnh để thực hiện tốt trên tập đặc biệt các dữ liệu vào được chọn. Để vượt qua các trở ngại này, các nhà khoa học máy tính đã chấp nhận tính phức tạp của thời gian được tiếp cận như một sự đo lường cơ bản sự thực thi của giải thuật. Thuật ngữ tính hiệu quả sẽ đề cập đến sự đo lường này và đặc biệt đối với sự phức tạp thời gian trong trường hợp xấu nhất.

Thời gian thực hiện chương trình.

Thời gian thực hiện một chương trình là một hàm của kích thước dữ liệu vào, ký hiệu T(n) trong đó n là kích thước (độ lớn) của dữ liệu vào. Chương trình tính tổng của n số

có thời gian thực hiện là T(n) = cn trong đó c là một hằng số. Thời gian thực hiện chương trình là một hàm không âm, tức là T(n) >= 0 với mọi n >= 0.

Đơn vị đo thời gian thực hiện.

Đơn vị của T(n) không phải là đơn vị đo thời gian bình thường như giờ, phút giây… mà thường được xác định bởi số các lệnh được thực hiện trong một máy tính lý tưởng. Khi ta nói thời gian thực hiện của một chương trình là T(n) = Cn thì có nghĩa là chương trình ấy cần Cn chỉ thị thực thi.

Thời gian thực hiện trong trường hợp xấu nhất.

Nói chung thì thời gian thực hiện chương trình không chỉ phụ thuộc vào kích thước mà còn phụ thuộc vào tính chất của dữ liệu vào. Nghĩa là dữ liệu vào có cùng kích thước nhưng thời gian thực hiện chương trình có thể khác nhau. Chẳng hạn chương trình sắp xếp dãy số nguyên tăng dần, khi ta cho vào dãy có thứ tự thì thời gian thực hiện khác với khi ta cho vào dãy chưa có thứ tự, hoặc khi ta cho vào một dãy đã có thứ tự tăng thì thời gian thực hiện cũng khác so với khi ta cho vào một dãy đã có thứ tự giảm. Vì vậy thường ta coi T(n) là thời gian thực hiện chương trình trong trường hợp xấu nhất trên dữ liệu vào có kích thước n, tức là: T(n) là thời gian lớn nhất để thực hiện chương trình đối với mọi dữ liệu vào có cùng kích thước n.

3. Cách tính độ phức tạp của giải thuật

Cách tính độ phức tạp của một giải thuật bất kỳ là một vấn đề không đơn giản. Tuy nhiên ta có thể tuân theo một số nguyên tắc sau:

Quy tắc bỏ hằng số

Nếu đoạn chương trình P có thời gian thực hiện T(n) = O(c1.f(n)) với c1 là một hằng số dương thì có thể coi đoạn chương trình đó có độ phức tạp tính toán là O(f(n)).

Quy tắc cộng – lấy max

Nếu T1(n) và T2(n) là thời gian thực hiện của hai đoạn chương trình P1 và P2; và T1(n)=O(f(n)), T2(n)=O(g(n)) thì thời gian thực hiện của đoạn hai chương trình đó nối tiếp nhau là T(n)=O(max(f(n),g(n)))

VD: Lệnh gán x:=15 tốn một hằng thời gian hay O(1), Lệnh đọc dữ liệu readln(x) tốn một hằng thời gian hay O(1). Vậy thời gian thực hiện cả hai lệnh trên nối tiếp nhau là O(max(1,1))=O(1)

Quy tắc nhân

Nếu T1(n) và T2(n) là thời gian thực hiện của hai đoạn chương trình P1và P2 và T1(n) = O(f(n)), T2(n) = O(g(n)) thì thời gian thực hiện của đoạn hai đoạn chương trình đó lồng nhau là T(n) = O(f(n).g(n))

Quy tắc tổng quát để phân tích một chương trình

- Thời gian thực hiện của mỗi lệnh gán, READ, WRITE là O(1).
- Thời gian thực hiện của một chuỗi tuần tự các lệnh được xác định bằng qui tắc cộng. Như vậy thời gian này là thời gian thi hành một lệnh nào đó lâu nhất trong chuỗi lệnh.
- Thời gian thực hiện cấu trúc IF là thời gian lớn nhất thực hiện lệnh sau THEN hoặc sau ELSE và thời gian kiểm tra điều kiện. Thường thời gian kiểm tra điều kiện là O(1).
- Thời gian thực hiện vòng lặp là tổng (trên tất cả các lần lặp) thời gian thực hiện thân vòng lặp. Nếu thời gian thực hiện thân vòng lặp không đổi thì thời gian thực hiện vòng lặp là tích của số lần lặp với thời gian thực hiện thân vòng lặp.

VD: Tính thời gian thực hiện của thủ tục sắp xếp "nổi bọt"

```
01  procedure Bubble (var a: array[1..n] of integer);
02  var i,j,temp: integer;
03  begin
04      for i:=1 to n-1 do                  {1}
05          for j:=n downto i+1 do              {2}
06              if a[j-1]>a[j] then          {3}
07              begin
08                  temp:=a[j-1];           {4}
09                  a[j-1]:=a[j];           {5}
10                  a[j]:=temp;             {6}
11              end;
12  end;
```

Về giải thuật sắp xếp nổi bọt, chúng ta sẽ bàn kĩ hơn trong chương 2. Ở đây, chúng ta chỉ quan tâm đến độ phức tạp của giải thuật.

Ta thấy toàn bộ chương trình chỉ gồm một lệnh lặp {1}, lồng trong lệnh {1} là lệnh {2}, lồng trong lệnh {2} là lệnh {3} và lồng trong lệnh {3} là 3 lệnh nối tiếp nhau {4}, {5} và {6}. Chúng ta sẽ tiến hành tính độ phức tạp theo thứ tự từ trong ra.

Trước hết, cả ba lệnh gán {4}, {5} và {6} đều tốn O(1) thời gian, việc so sánh a[j-1] > a[j] cũng tốn O(1) thời gian, do đó lệnh {3} tốn O(1) thời gian. Vòng lặp {2} thực hiện (n-i) lần, mỗi lần O(1) do đó vòng lặp {2} tốn O((n-i).1) = O(n-i).Vòng lặp {1} lặp có i chạy từ 1 đến n-1 nên thời gian thực hiện của vòng lặp {1} và cũng là độ phức tạp của giải thuật là:

$$T(n) = \sum_{i=1}^{n-1}(n-i) = \frac{n.(n-1)}{2} = O(n^2)$$

Chú ý: Trong trường hợp vòng lặp không xác định được số lần lặp thì chúng ta phải lấy số lần lặp trong trường hợp xấu nhất.

Tìm kiếm tuần tự. Hàm tìm kiếm Search nhận vào một mảng a có n số nguyên và một số nguyên x, hàm sẽ trả về giá trị logic TRUE nếu tồn tại một phần tử a[i] = x, ngược lại hàm trả về FALSE.

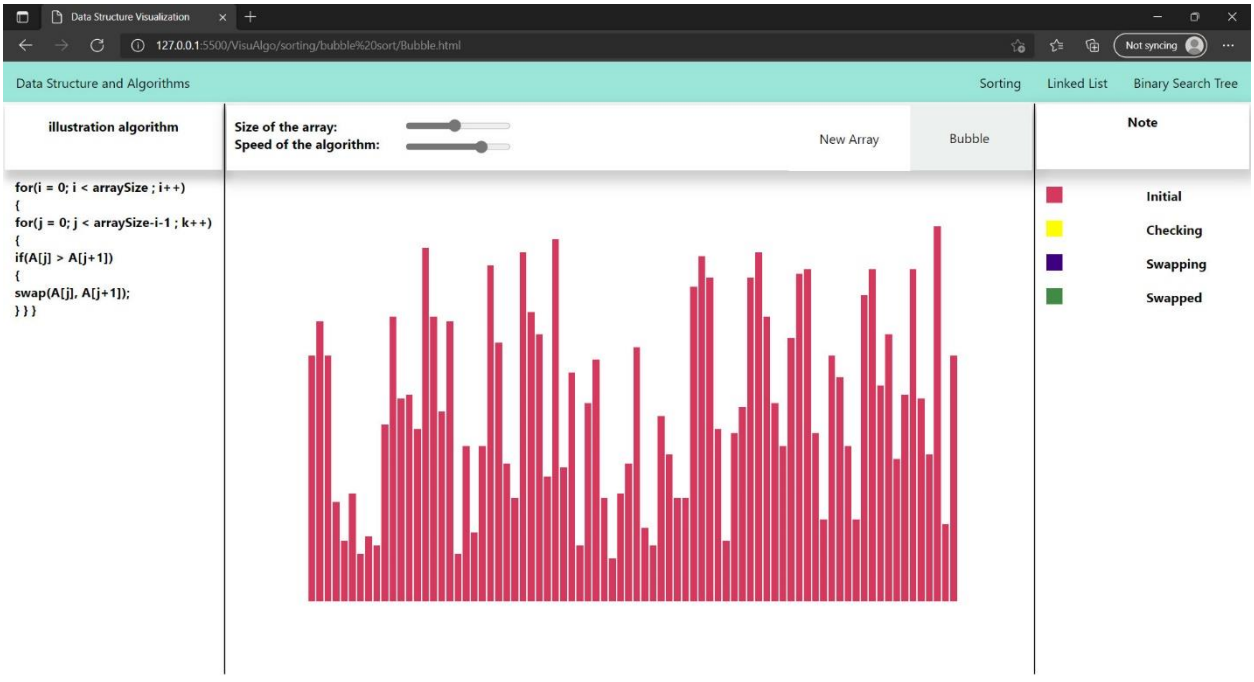
Giải thuật tìm kiếm tuần tự là lần lượt so sánh x với các phần tử của mảng a, bắt đầu từ a[1], nếu tồn tại a[i] = x thì dừng và trả về TRUE, ngược lại nếu tất cả các phần tử của a đều khác X thì trả về FALSE.

```
01  FUNCTION Search(a:ARRAY[1..n] OF Integer;x:Integer):Boolean;
02  VAR i:Integer;
03      Found:Boolean;
04  BEGIN
05      i:=1;                                          {1}
06      Found:=FALSE;                                  {2}
07      WHILE(i<=n) AND (not Found) DO                  {3}
08          IF A[i]=X THEN Found:=TRUE ELSE i:=i+1;    {4}
09      Search:=Found;                                 {5}
10  END;
```

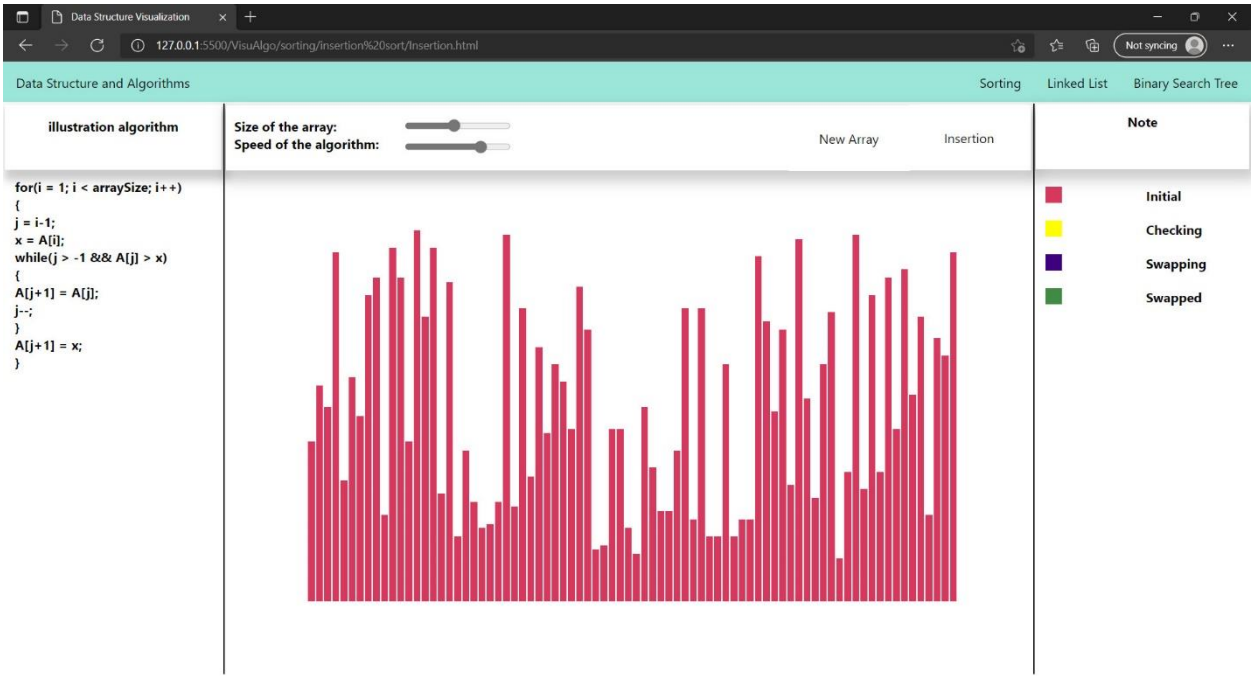
Ta thấy các lệnh {1}, {2}, {3} và {5} nối tiếp nhau, do đó độ phức tạp của hàm Search chính là độ phức tạp lớn nhất trong 4 lệnh này. Dễ dàng thấy rằng ba lệnh {1}, {2} và {5} đều có độ phức tạp O(1) do đó độ phức tạp của hàm Search chính là độ phức tạp của lệnh {3}. Lồng trong lệnh {3} là lệnh {4}. Lệnh {4} có độ phức tạp O(1). Trong trường hợp xấu nhất (tất cả các phần tử của mảng a đều khác x) thì vòng lặp {3} thực hiện n lần, vậy ta có T(n) = O(n).

CHƯƠNG III: CHƯƠNG TRÌNH

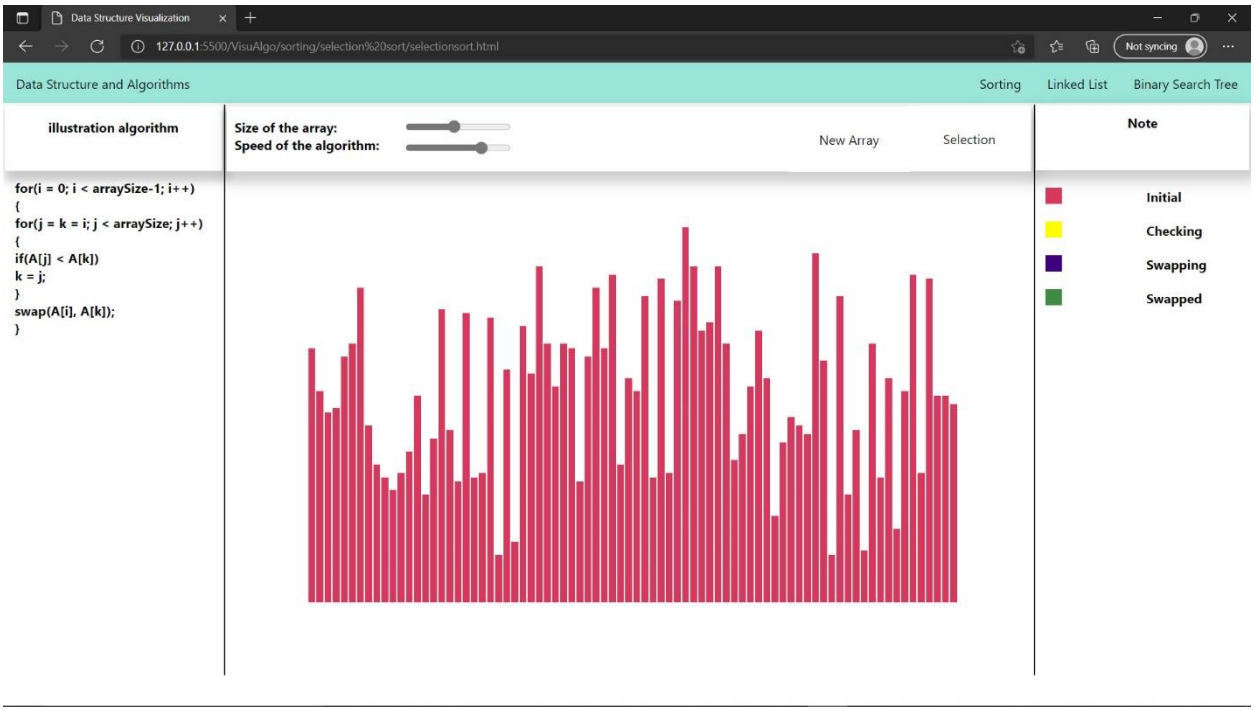
Một số hình ảnh của chương trình



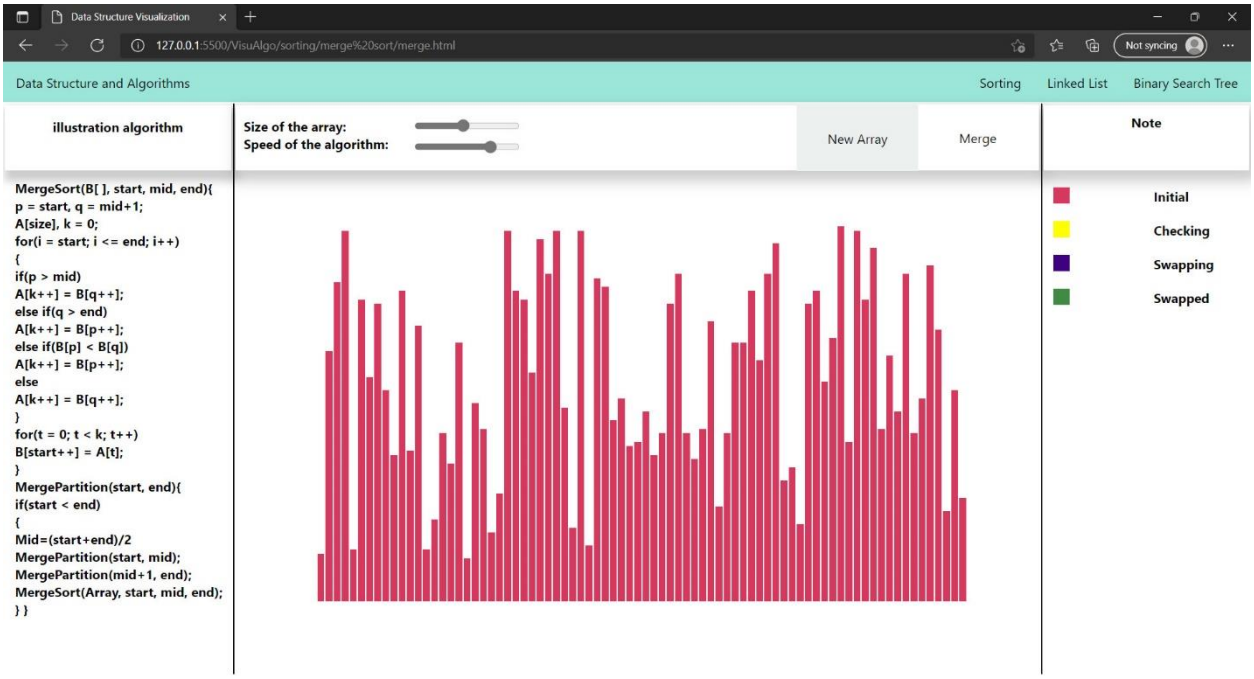
Hình 12. Bubble sort



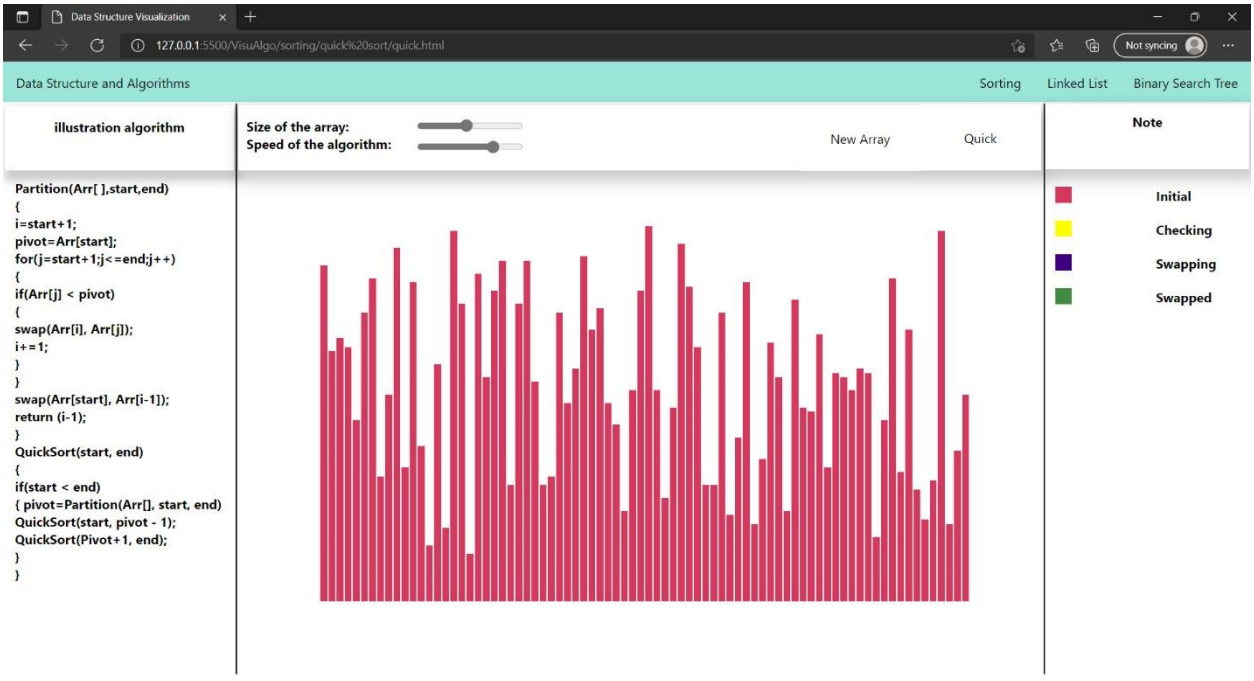
Hình 13. Insertion sort



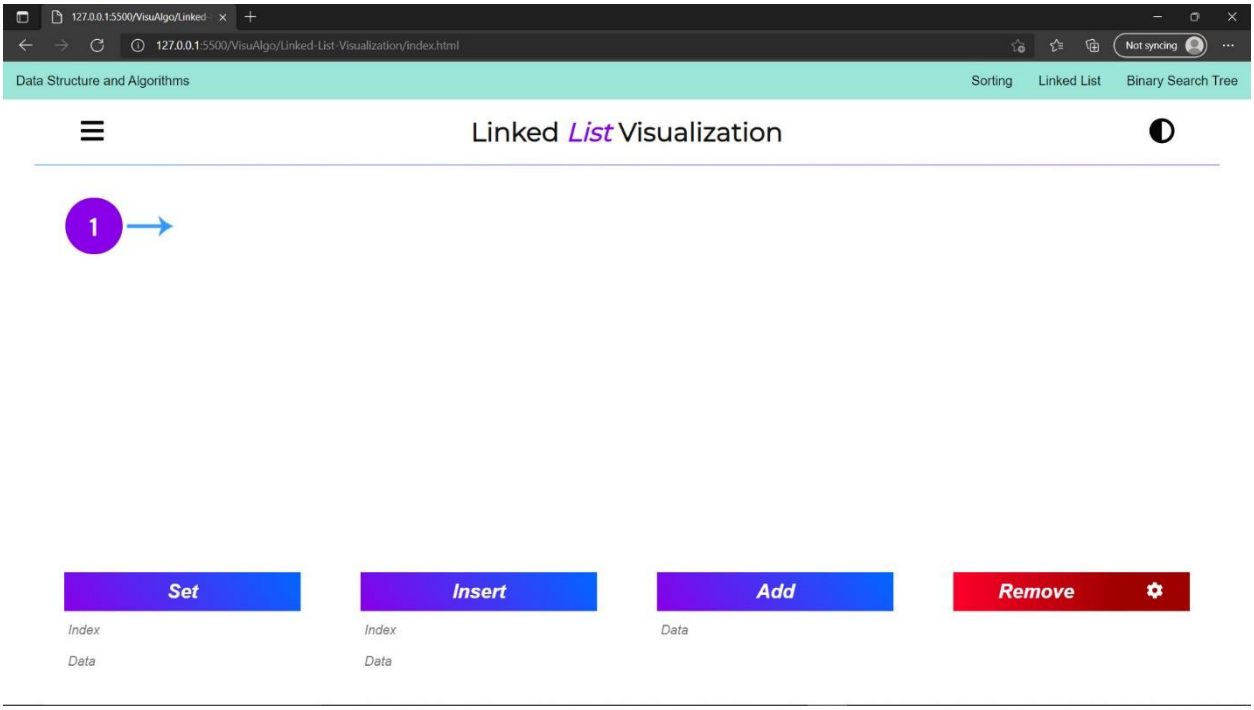
Hình 14. Selection sort



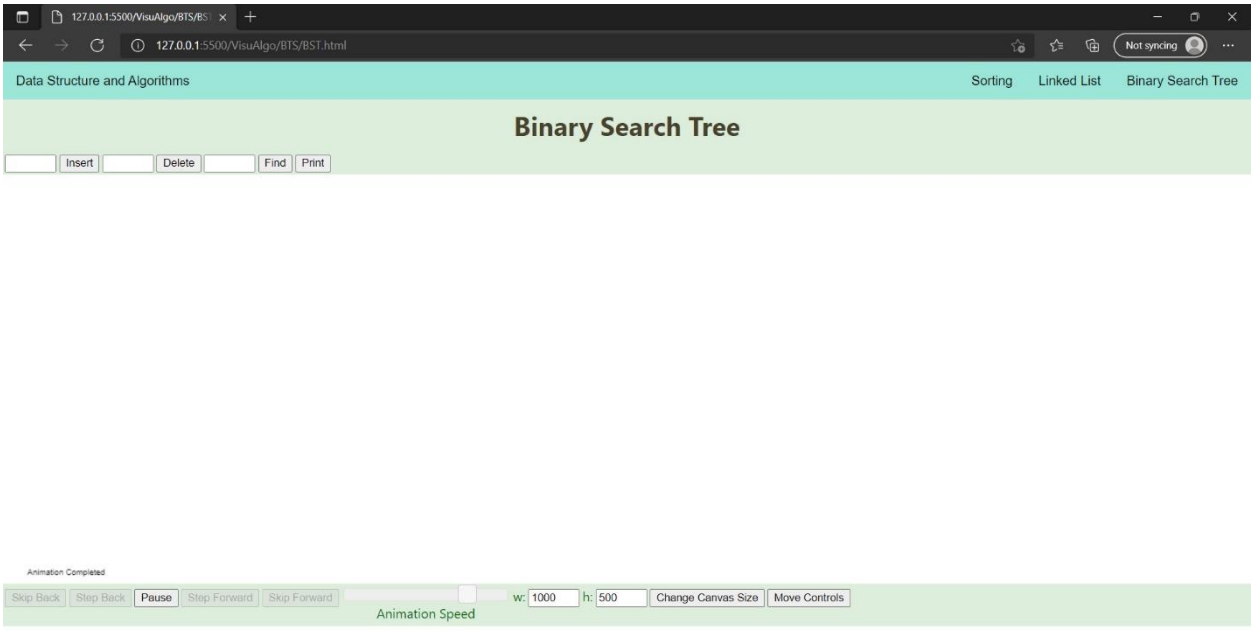
Hình 15. Merge sort



Hình 16. Quick sort



Hình 17. Linked list



Hình 18. Binary search tree

CHƯƠNG IV: TỔNG KẾT

1. Kết quả đạt được
   - Xây dựng thành công chương trình minh họa từng bước các thuật toán trong môn cấu trúc dữ liệu và giải thuật.

   Ưu điểm:

   - Dễ nhìn, sinh động.

- Phù hợp với những người học thuật toán.

Nhược điểm:

- Hạn chế về số lượng các thuật toán
- Bố cục chưa đẹp.

2. Phương hướng phát triển
   - Đa dạng hơn về thuật toán.
   - Thiết kế lại bố cục.
3. Tài liệu tham khảo
   - https://visualgo.net/en
   - Nguồn từ GitHub.
   - https://opendsa-server.cs.vt.edu/

Bảng phân công công việc

Họ và tên	Nhiệm vụ	Hoàn thành
Hoàng Minh Tài		
Nguyễn Hoàng Hiệp		
Cao Lâm Bảo Khanh		
Nguyễn Hữu Đại		
Trương Được		