SỞ GIÁO DỤC VÀ ĐÀO TẠO TỈNH NGHỆ AN

TRƯỜNG THPT CON CUÔNG

Tên đề tài:

VẬN DỤNG THUẬT TOÁN SẮP XẾP, TÌM KIẾM VÀO VIỆC BỒI DƯỠNG HỌC SINH GIỎI, THI CHUYÊN PHAN TRÊN NGÔN NGỮ LẬP TRÌNH C++

Người thực hiện: Đặng Văn Hảo

Đơn vị: Trường THPT Con Cuông

Con Cuông – Năm 2021

PHỤ LỤC Trang

PHẦN I: ĐẶT VẤN ĐỀ

PHẦN II. NỘI DUNG NGHIÊN CỨU

PHẦN I: ĐẶT VẤN ĐỀ

I. LÝ DO CHỌN ĐỀ TÀI

Trong cuộc sống của chúng ta có rất nhiều công việc được thực hiện, nhưng quan trọng hơn những công việc đó được chúng ta sắp xếp thực hiện vào thời gian nào là hợp lý và quy trình thực hiện ra sao cũng hết sức quan trọng.

Với việc giải quyết các bài toán trong tin học cũng vậy, chúng ta phải biết lựa chọn cho mình một thuật toán hiệu quả nhất để giải bài toán. Ở phương diện lập trình một bài lập trình hiệu quả nhất khi thời gian chạy nhanh nhất và dung lượng bộ nhớ sử dụng ít nhất hay đó chính là độ phức tạp của thuật toán bé nhất có thể.

Đối với một số bài toán thoạt nhìn thì trông rất khó, phức tạp. Nhưng nếu để ý và quan sát, chúng ta chỉ cần dùng thuật toán sắp xếp, tìm kiếm để chuyển bài toán sang một một hướng mới từ đó có thể giải quyết bài toán một cách đơn giản hơn nhiều. Mặt khác trong tin học sắp xếp lại có rất nhiều phương pháp, đôi khi các chúng ta chỉ chọn cho mình một phương pháp dễ nhớ, dễ hiểu mà không để ý đến thời gian và dung lượng trong chương trình. Nếu là học sinh ở lớp thường thì điều này có thể được nhưng đã là học sinh giỏi và giáo viên bồi dưỡng học sinh giỏi thì phải có suy nghĩ khác.

Để giúp cho các em hiểu thêm và nắm rõ về các phương pháp sắp xếp cũng như tìm kiếm. Tôi đã tìm hiểu đề tài **"Vận dụng thuật toán sắp xếp, tìm kiếm vào việc bồi dưỡng học sinh giỏi, thi chuyên phan trên ngôn ngữ lập trình C++"**.

Với đề tài này các em sẽ biết lựa chọn cho mình một phương pháp sắp xếp, tìm kiếm hợp lý nhất trong việc giải bài toán. Từ đó nâng cao kĩ năng xử lí khi gặp những bài toán mà có thể vận dụng thuật toán sắp xếp, tìm kiếm một cách hiệu quả nhất, độ phức tạp nhỏ nhất và nhanh nhất.

II. ĐỐI TƯỢNG NGHIÊN CỨU

- Đối tượng nghiên cứu là học sinh ôn thi học sinh giỏi, thi vào chuyên tin trường Phan, giáo viên giảng dạy môn Tin trong trường THPT.

- Các thuật toán sắp xếp, tìm kiếm và ứng dụng vào các bài toán.

III. MỤC ĐÍCH NGHIÊN CỨU, ĐÓNG GÓP MỚI CỦA ĐỀ TÀI

Trên cơ sở nghiên cứu, các thuật toán sắp xếp, tìm kiếm còn đưa ra những ứng dụng của nó trong việc giải các bài lớn trên máy tính. Nhằm giúp học sinh, giáo viên có cái nhìn tổng quát hơn phần nào về tầm quan trọng của các thuật toán sắp xếp, tìm kiếm trình bày các bài toán thường gặp, cách giải, cài đặt chương trình bằng NNLT C. Từ đó nâng cao kĩ năng xử lí các bài toán khó, phức tạp có liên quan đến thuật toán sắp xếp và tìm kiếm. Đồng thời hướng dẫn và sử dụng một số hàm có sẵn trong C++.

IV. PHƯƠNG PHÁP NGHIÊN CỨU

Để hoàn thành nhiệm vụ của đề tài, tôi đã nghiên cứu rất nhiều sách và các chuyên đề Tin học dành cho viêc bồi dưỡng học sinh giỏi tin, những vấn đề cơ bản nhất về cấu trúc dữ liệu, thuật toán và cài đặt chương trình.

Lựa chọn một số bài toán giải áp dụng các thuật toán sắp xếp trong chương trình tin học chuyên THPT.

V. CẤU TRÚC CỦA CHUYÊN ĐỀ

Ngoài phần đặt vấn đề và phần kết luận nội dung của đề tài gồm 2 chương:

1. Cơ sở lý luận

Trình bày đầy đủ các thuật toán sắp xếp, tìm kiếm với những quy trình của từng phương pháp và các chương trình cụ thể để bạn đọc dễ hiểu nhất. Qua đó có thể vận dụng để giải quyết các bài toán về sau.

2. Cơ sở thực tiễn

Nêu ra thực trạng vấn đề về học sinh cũng như giáo viên trong việc giải quyết các bài toán lớn, các bài toán ôn thi học sinh giỏi hiện nay. Qua đó giải quyết vấn đề bằng cách liệt kê và đưa ra các bài toán điển hình có sử dụng thuật toán sắp xếp, tìm kiếm để có thể vận dụng lập trình giải các bài toán trong các đề thi hay khi ôn luyện. Trang bị kiến thức cho học sinh và giáo viên để giải quyết tốt khi gặp các bài toán có liên quan đến phương pháp sắp xếp, tìm kiếm một cách hiệu quả nhất.

PHẦN II. NỘI DUNG NGHIÊN CỨU

CHƯƠNG I. CƠ SỞ LÍ LUẬN

1.1 Thuật toán sắp xếp

Sắp xếp là quá trình xử lý một danh sách các phần tử (hoặc các mẫu tin) để đặt chúng theo một thứ tự thỏa mãn một tiêu chuẩn nào đó dựa trên nội dung thông tin lưu giữ tại mỗi phần tử. Hay đơn giản ta có thể hiểu sắp xếp là một quá trình tổ chức lại một dãy các dữ liệu theo một trật tự nhất định.

Tại sao cần phải sắp xếp các phần tử thay vì để nó ở dạng tự nhiên (chưa có thứ tự) vốn có ?

Mục đích của việc sắp xếp là nhằm giúp cho việc tìm kiếm dữ liệu một cách dễ dàng và nhanh chóng. Sắp xếp là một việc làm hết sức cơ bản và được dùng rộng rãi trong các kĩ thuật lập trình nhằm xử lý dữ liệu.

Các thuật toán sắp xếp được phân chia thành hai nhóm chính là:

- Sắp xếp đơn giản

* Sắp xếp chọn (Selection sort).

* Sắp xếp chèn (Insertion sort).

* Sắp xếp nổi bọt (Bubble sort).

-Sắp xếp với độ phức tạp O(nlog(n)).

* Sắp xếp theo phân đoạn (Quick Sort).

* Sắp xếp hòa nhập (merge sort).

* Sắp xếp vun đống (Heap sort).

Sau đây chúng ta sẽ lần lượt tìm hiểu các thuật toán trên.

1.1.1. Sắp xếp chọn (Selection sort).

Ý tưởng: 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 1 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.

Ví dụ cho một dãy số đơn giản với 8 phần tử cần sắp xếp cho nó tăng dần.

Ban đầu	5	89	0	4	7	2	10	41
i=1	0	89	5	4	7	2	10	41
i=2	0	2	5	4	7	89	10	41
i=3	0	2	4	5	7	89	10	41
i=4	0	2	4	5	7	89	10	41
i=5	0	2	4	5	7	89	10	41
i=6	0	2	4	5	7	10	89	41
i=7 dãy đã sắp xếp	0	2	4	5	7	10	41	89

Những ô tô gạch chân là những ô đổi chỗ tại mỗi bước i.

Thuật toán như sau:

- Đầu vào: mảng a[i] (1<=i<= n).
- Đầu ra: Đưa ra dãy a[i] tăng dần.

Bước 1: i=1;

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

Bước 3: Hoán vị a[min] và a[i]

Bước 4: Nếu i<=n-1 thì i=i+1; Lặp lại bước 2

Ngược lại: Dừng đưa ra dãy số đã sắp xếp

***Cài đặt:**

```
void selectionsort()
{   int k, mi;
    for(int i=1;i<n;i++)
   { mi=a[i];
      for(int j=i+1;j<=n;j++)  if(a[j]<mi) {mi=a[j];k=j;}
      if (a[i]!=mi) swap(a[i],a[k]);
   }}
```

Trong C cho phép ta gọi thủ tục hoán đổi trong khi viết chương trình, hoặc tự viết cũng được.

Chương trình cụ thể

```
#include<bits/stdc++.h>
using namespace std;
 int a[100],n;
void selectionsort()
{   int k,mi;
    for(int i=1;i<n;i++)
    {  mi=a[i];
        for(int j=i+1;j<=n;j++) if(a[j]<mi) {mi=a[j];k=j;}
        if (a[i]!=mi) swap(a[i],a[k]);
    }}
int main()
{  freopen("selection.inp","r",stdin);
    freopen("selection.out"."w",stdout);
    cin>>n;
for(int i=1;i<=n;i++) cin>>a[i];
    selectionsort();
   for(int i=1;i<=n;i++)        cout<<a[i]<< " ";
    return 0; }
```

Chú ý: các chương trình sau tương tự chỉ thay đoạn sắp xếp nên tôi chỉ viết thủ tục sắp xếp.

1.1.2. Sắp xếp chèn (Insertion sort).

Ý tưởng: Lấy từng phần tử từ dãy nguồn, chèn vào dãy đích sao cho đảm bảo dãy đích có thứ tự. Chẳng hạn là giả sử ta có dãy a[1]..a[i-1] đã có thứ tự, cần phải xác định vị trí thích hợp của phần tử a[i] trong dãy a[1]..a[i - 1] bằng phương pháp tìm kiếm tuần tự từ a[i - 1] trở về a[1] để tìm ra vị trí thích hợp của a[i]. Ta chèn a[i] vào vị trí này và kết quả là dãy a[1], ..., a[i] có thứ tự. Ta áp dụng cách làm này với i = 2, 3, ..., n.

* Ví dụ:

Ta phải sắp xếp dãy số : 0	3	1	8	5	4	9	0	
Bước 1: 0	3							
Bước 2: 0	1	3						
Bước 3:	0	1	3	8				
Bước 4: 0	1	3	5	8				
Bước 5: 0	1	3	4	5	8			
Bước 6:	0	1	3	4	5	8	9	
Bước 7:	0	0	1	3	4	5	8	9

Thuật toán:

Bước 1: Chèn phần tử a[2] vào danh sách đã có thứ tự a[1] sao cho a[1], a[2] là một danh sách có thứ tự.

Bước i: Chèn phần tử a[i+1] vào danh sách đã có thứ tự a[1], a[2], …, a[i] sao cho a[1], a[2],…, a[i+1] là một danh sách có thứ tự.

Sau n-1 bước thì kết thúc.

*** Cài đặt**

```
void insertionsort()
{   for(int i=2;i<=n;i++)
   {          int j=i;
      while((j>1)&&(a[j]<a[j-1]))
          {  swap(a[j],a[j-1]);    j--; }
}}
```

Độ phức tạp của thuật toán là $O(n^2)$

1.1.3. Sắp xếp nổi bọt (Bubble sort).

Ý tưởng: Phương pháp này sẽ duyệt danh sách nhiều lần, Giả sử có mảng n phần tử. Chúng ta sẽ tiến hành duyệt từ cuối lên đầu, so sánh 2 phần tử kề nhau, nếu chúng bị ngược thứ tự thì đổi vị trí, việc duyệt này bắt đầu từ cặp phần tử thứ n-1 và n. Tiếp theo là so sánh cặp phần tử thứ n-2 và n-1, … cho đến khi so sánh và đổi chỗ cặp phần tử thứ nhất và thứ hai. Sau bước này phần tử nhỏ nhất đã được nổi lên vị trí trên cùng (nó giống như hình ảnh của các "bọt" khí nhẹ hơn được nổi lên trên). Tiếp theo tiến hành với các phần tử từ thứ 2 đến thứ n.

Ví dụ cho một dãy số đơn giản với 8 phần tử và sắp xếp cho nó tăng dần.

Ban đầu	Bước 1	Bước 2	Bước 3	Bước 4	Bước 5	Bước 6	Bước 7	Kết quả
19	0	0	0	0	0	0	0	0
7	19	1	1	1	1	1	1	1
0	7	19	1	1	1	1	1	1
1	1	7	19	3	3	3	3	3
3	1	1	7	19	7	7	7	7
9	3	3	3	7	19	9	9	9
10	9	9	9	9	9	19	10	10
1	10	10	10	10	10	10	19	19

Thuật toán

Bước 1: Xét các phần tử từ a[n] đến a[2], với mỗi phần tử a[j] so sánh nó với phần tử a[j1]. Nếu phần tử a[j] nhỏ hơn phần tử a[j-1] thì đổi chỗ a[j] với a[j-1].

Bước 2: Xét các phần tử từ a[n] đến a[3], làm tương tự như bước 1.

Bước i: Xét các phần tử từ a[n] đến a[i+1], làm tương tự như bước 1.

Sau n-1 bước ta được dãy đã có thứ tự

***Cài đặt:**

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

Độ phức tạp của thuật toán là $O(n^2)$

1.1.4. Sắp xếp theo phân đoạn (Quick Sort).

Ý tưởng: Xét mảng a có các phần tử a[1], a[2],…,a[n].

Khi đó chọn phần tử x của mảng làm chốt (x) để so sánh. Ta phân hoạch mảng thành 2 phần bằng cách chuyển tất cả các phần có giá trị lớn hơn chốt sang phải chốt, các thành phần có giá trị bé hơn hoặc bằng chốt sang trái chốt.

- Dãy con thứ nhất gồm phần tử có giá trị nhỏ hơn chốt x.

- Dãy con thứ hai gồm các phần tử có giá trị lớn hơn chốt x.

Sau đó áp dụng giải thuật phân hoạch này cho dãy con thứ nhất nhất và dãy con thứ 2, nếu các dãy con có nhiều hơn một phần tử (Đệ qui).

Cụ thể là xét một đoạn của dãy từ thành phần l đến thành phần thứ r (từ trái sang phải).

- Lấy giá trị của thành phần thứ (l+r) div 2 gán vào biến X.

- Cho i ban đầu là l.

- Cho j ban đầu là r.

- Lắp lại.

* Chừng nào còn A[i] < X thì tăng i.

* Chừng nào còn A[j] > X thì giảm j.

* i<=j thì

  + Hoán vị A[i] và A[j]

  + Tăng i

  + Giảm j

Cho đến khi i>j

  + Sắp xếp đoạn từ A[L] đến A[j]

+ Sắp xếp đoạn từ A[i] đến A[R]

* Ví dụ:

xếp dãy số (n=8): 20 10 45 28 24 32 69 5

Lần đầu: l=1; r=8; X=A[4]=28

Sau hai lần đổi chỗ ta được 20 10 5 24 28 32 69 45

(những con ký hiệu giống nhau đổi chỗ cho nhau)

Khi đó chia dãy số ban đầu thành hai dãy 20 10 5 24 28 32 69 45

Lại tiếp tục với dãy thứ nhất (phần tô nền)

l=1, r=4, X=A[2]=10

Sau một lần đổi chỗ ta lại có dãy 5 10 20 24 28 32 69 45

Lại xử lý với l=3, r=4 nhưng không có sự đổi chỗ nào nên ta thu được dãy a như sau:

5 10 20 24 28 32 69 45

Lại tiếp tục xử lý đoạn sau lần đầu: 5 10 20 24 28 32 69 45

Với l=5, r=8 X=a[6]=32, có dãy: 5 10 20 24 28 32 69 45

Với l=7, r=8, X=a[7]=69, có dãy: 5 10 20 24 28 32 45 69

Đây chính là kết quả cần tìm.

***Cài đặt**

```
void quicksort(int l,int r)
{   int x,i,j;
    i=l; j=r;
    x=a[(i+j)/2];
    while (i<=j)
   { while (a[i]<x) i++;
      while (a[j]>x)j--;
      if (i<=j)
      {   swap(a[i],a[j]);    i++;   j--;      }
    }      if (i<r)quicksort(i,r);
     if (j>l) quicksort(l,j);
}
```

* Nhận xét, so sánh

- Độ phức tạp của thuật toán là O(nlog(n)).
- Quick Sort phức tạp hơn Bubble Sort nhưng hiệu quả hơn.
- Quick Sort thích hợp cho danh sách ban đầu chưa có thứ tự.
- Quick Sort kém hiệu quả khi danh sách ban đầu gần có thứ tự. Đặc biệt với danh sách đã có thứ tự (lớn dần hoặc nhỏ dần) lại là trường hợp xấu nhất của giải thuật Quick Sort.

1.1.5. Sắp xếp trộn (Merge sort).

Sắp xếp trộn (merge sort) cùng với sắp xếp nhanh là hai thuật toán sắp xếp dựa vào tư tưởng "chia để trị" (divide and conquer). Thủ tục cơ bản là việc trộn hai danh sách đã được sắp xếp vào một danh sách mới theo thứ tự.

Giả sử dãy A[1],...,A[i],...,A[n] có hai đoạn đã được sắp xếp không giảm là A[1], ..., A[i] và A[i+1], ..., A[n], ta tiến hành tạo ra dãy mới bằng cách lần lượt lấy hai phần tử đầu tiên của mỗi đoạn và so sánh với nhau. Phần tử nào nhỏ hơn thì lấy ra dãy mới và bỏ nó ra khỏi đoạn đang xét. Cứ tiến hành như vậy cho tới khi một dãy bị vét hết (không còn phần tử nào). Lấy toàn bộ phần tử còn lại của đoạn không xét hết nối vào sau dãy đích. Như vậy dãy mới là một dãy đã được sắp xếp.

Trong mọi trường hợp, có thể coi dãy gồm duy nhất 1 phần tử là đã được sắp xếp. Lợi dụng việc này, ta phân chia một dãy ra thành nhiều dãy con chỉ gồm một phần tử rồi lần lượt hòa nhập từng đôi một với nhau.

Điểm khác biệt giữa sắp xếp nhanh và sắp xếp trộn là trong sắp xếp trộn việc xác định thứ tự được xác định khi "trộn", tức là trong khâu tổng hợp lời giải sau khi các bài toán con đã được giải, còn sắp xếp nhanh đã quan tâm đến thứ tự các phần tử khi phân chia một danh sách thành hai danh sách con.

Xét ví dụ 1: A={8, 3, 6, 5, 2, 4, 9, 7}

Dãy ban đầu là: 8 3 6 5 2 4 9 7

Từ dãy ban đầu chia làm hai đoạn con {8, 3, 6, 5} và {2, 4, 9, 7}

Chia tiếp làm các đoạn con A1={8, 3}, A2={6,5}, A3={2, 4} và A4={9, 7};

Merge từng đoạn ta được A1={3, 8}, A2={5 ,6}, A3={2, 4} và A4={ 7, 9};

Merge(A1+A2), Merge(A3+A4) ta được A5= {3, 5, 6, 8}và A6={2, 4, 7, 9}

Merge(A5+A6) ta được 2, 3, 4, 5, 6, 7, 8, 9

Cuối cùng ta có dãy 2, 3, 4, 5, 6, 7, 8, 9 đã sắp xếp

Với cách làm như trên ta còn gọi nó là trộn trực tiếp tức việc phân hoạch thành các dãy con đơn giản là chỉ tách dãy gồm n phần tử thành n dãy con. Đòi hỏi của thuật toán là tính có thứ tự của các dãy con luôn được thõa trong cách phân hoạch này vì dãy gồm 1 phần tử luôn có thứ tự, cứ mỗi lần tách rồi trộn, chiều dài của các dãy con sẽ được nhân đôi. Khi đó ta có thuật toán sau.

Thuật toán

Bước 1: k=1;//chiều dài của dãy con trong bước hiện hành

Bước 2: Tách A_1, A_2,…, A_n thành 2 dãy B và C theo quy tắc luân phiên từng nhóm (n div 2) phần tử

B=A1,…,A2, A3,…,A(n div 2).

C= A(n div 2)+1,…, An-1, An.

Bước 3: trộn từng cặp dãy con gồm k phần tử của dãy B, C vào dãy A

Bước 4: k=k*2;

Nếu k<n thì trở lại bước 2

Ngược lại thì dừng

Ta thấy rằng số lần lặp của bước 2 và bước 3 trong thuật toán MergeSort bằng logn do sau mỗi lần lặp giá trị của k tăng lên gấp đôi. Dễ thấy, chi phí thực hiện bước 2 và bước 3 tỉ lệ thuận với n. Như vậy, chi phí thực hiện của giải thuật MergeSort sẽ là O(nlogn). Do không sử dụng thông tin nào về đặc tính của dãy cần sắp xếp, nên trong mọi trường hợp của thuật toán chi phí là không đổi. Đây cũng chính là một trong những nhược điểm lớn của thuật toán

***Cài đặt**

```
#include<bits/stdc++.h>
using namespace std;
 int a[100], n;
void meger(int a[],int l,int m,int r)
{    int i=l,j=m;
     int nb=r-l;
     int b[nb];
     int k=l;
        while ((i<=m-1)&&(j<=r))
     {   if (a[i]>a[j])
     {   b[k++]=a[j++];}
```

```
        else {b[k++]=a[i++];}
      }    while (i<=m-1)
     {
b[k++]=a[i++];
}
     while (j<=r) { b[k++]=a[j++]; }
     for(k=0;k<=nb;k++,r--) {a[r]=b[r]; }
     }
void megersort(int a[],int l,int r){
     if (l<r)
     {  int m=(int)(r+l)/2;
      megersort(a,l,m);
      megersort(a,m+1,r);
      meger(a,l,m+1,r);
}}
     main()
     {   freopen("meger.inp","r",stdin);
         freopen("meger.out","w",stdout);
         cin>>n;
```

1.1.6. Sắp xếp vun đống (Heap sort).

Sắp xếp vun đống (heapsort) là một trong các phương pháp sắp xếp chọn. Ở mỗi bước của sắp xếp chọn ta chọn phần tử lớn nhất đặt vào cuối danh sách, sau đó tiếp tục với phần còn lại của danh sách. Thông thường sắp xếp chọn chạy trong thời gian $O(n^2)$. Nhưng heapsort đã giảm độ phức tạp này bằng cách sử dụng một cấu trúc dữ liệu đặc biệt được gọi là đống (Heap). Đống (Heap) là cây nhị phân mà trọng số ở mỗi đỉnh cha lớn hơn hoặc bằng trọng số các đỉnh con của nó. Một khi danh sách dữ liệu đã được vun thành đống, gốc của nó là phần tử lớn nhất, thuật toán sẽ giải phóng nó khỏi đống để đặt vào cuối danh sách. Do đó trong Heap ta có:

+ Nút gốc có khóa lớn nhất

+Dãy khóa nhận được khi đi theo một đường bất kì là dãy có thứ tự tăng dần.

Định nghĩa: Một cây nhị phân nếu xét từ trên xuống dưới, từ trái sang phải là khoá của một nút luôn ≥ nút con thì gọi là đống.

Cần lưu ý:

- Không nói gì, ngầm định là đống không tăng theo định nghĩa.
- Không nói gì, ngầm định là đống được lưu trữ vào dạng mảng danh sách. Lần lượt từ trái sang phải, hết mức nọ đến mức kia.
- Về chỉ số: Nếu i chỉ ra nút bất kì, tuy nhiên i ≤ n/2, thì :
- Con trái có chỉ số là: 2i
- Con phải có chỉ số là: 2i+1
- Khi nói con chung chung, ngầm định là con trái.

* Một cây nhị phân, được gọi là đống cực đại nếu khóa của mọi nút không nhỏ hơn khóa các con của nó. Khi biểu diễn một mảng a[] bởi một cây nhị phân theo thứ tự tự nhiên điều đó nghĩa là a[i] ≥ a[2*i] và a[i] ≥ a[2*i +1] với mọi i =1..int(n/2). Ta cũng sẽ gọi mảng như vậy là đống. Như vậy trong đống a[1] (ứng với gốc của cây) là phần tử lớn nhất. Mảng bất kỳ chỉ có một phần tử luôn luôn là một đống.

* Vun đống

Việc sắp xếp lại các phần tử của một mảng ban đầu sao trở thành đống được gọi là vun đống.

*Vun đống tại đỉnh thứ i.

Nếu hai cây con gốc 2*i và 2*i +1 đã là đống thì để cây con gốc i trở thành đống chỉ việc so sánh giá trị a[i] với giá trị lớn hơn trong hai giá trị a[2*i] và a[2*i + 1], nếu a[i] nhỏ hơn thì đổi chỗ chúng cho nhau. Nếu đổi chỗ cho a[2*i], tiếp tục so sánh với con lớn hơn trong hai con của nó cho đến khi hoặc gặp đỉnh lá.

• Vun một mảng thành đống

Để vun mảng a[1..n] thành đống ta vun từ dưới lên, bắt đầu từ phần tử a[j] với j =int(n/2) ngược lên tới a[1]. (Thủ tục MakeHeap trong giải mã dưới đây).

* Sắp xếp bằng vun đống

Đổi chỗ (Swap):mảng a[1..n] đã là đống, lấy phần tử a[1] trên đỉnh của đống ra khỏi đống đặt vào vị trí cuối cùng n, và chuyển phần tử thứ cuối cùng a[n] lên đỉnh đống thì phần tử a[n] đã được đứng đúng vị trí.

*Vun lại: Phần còn lại của mảng a[1..n-1] chỉ khác cấu trúc đống ở phần tử a[1]. Vun lại mảng này thành đống với n-1 phần tử.

Lặp: Tiếp tục với mảng a[1..n-1]. Quá trình dừng lại khi đống chỉ còn lại một phần tử.

Áp dụng thuật toán trên cho ví dụ sau:

Ví dụ: cho dãy số: 2 3 7 6 4 1 5

Vun gốc A[3] được mảng A={2, 3, 7, 6, 4, 1, 5}

Vun gốc A[2] được mảng A={2, 6, 7, 3, 4, 1, 5}

Vun gốc A[1] được mảng A={7, 6, 5, 3, 4, 1, 2}

bây giờ A={7, 6, 5, 3, 4, 1, 2} đã là đống

Sắp xếp

Đổi chỗ A[1] với A[7] ta có A={2, 6, 5, 3, 4, 1, 7} vun lại mảng A[1..6] ta được A={ 6, 4, 5, 3, 2, 1, 7}

Đổi chỗ A[1] với A[6] ta có A={ 1, 4, 5, 3, 2, 6, 7} vun lại mảng A[1..5] ta được A ={ 5, 4, 1, 3, 2 , 6, 7}

Đổi chỗ A[1] với A[5] ta có A={ 2, 4, 1, 3, 5, 6, 7} vun lại mảng A[1..4] ta được A={ 4, 3, 1, 2, 5, 6, 7}

Đổi chỗ A[1] với A[4] ta có A={ 2, 3, 1, 4, 5, 6, 7} vun lại mảng A[1..3] ta được A={ 3, 2, 1, 4, 5, 6, 7}

Đổi chỗ A[1] với A[3] ta có A={ 1, 2, 3, 4, 5, 6, 7} vun lại mảng A[1..2] ta được A={ 2, 1, 3, 4, 5, 6, 7}

Đổi chỗ A[1] với A[2] ta có A={ 1, 2, 3, 4, 5, 6, 7} vun lại mảng ta được dãy số đã sắp xếp xong.

***Cài đặt**

```
#include<iostream>
using namespace std;
int a[10],n;
//hoan vi nut cha thu i phai lon hon nut con
void heapity(int a[], int n, int i)
{   int l=2*(i+1)-1;
    int r=2*(i+1);
    int largest;
    if ((l<n)&&(a[l]>a[i])) largest=l;
    else largest=i;
```

```
    if((r<n)&&(a[r]>a[largest])) largest=r;
    if(i!=largest) {swap(a[i],a[largest]); heapity(a,n,largest);
    }   }
   //xay dung heap sao cho moi nut cha luon lon hon nut con tren cay
   void builheap(int a[], int n)
   {       for(int i=int(n/2)-1;i>=0;i--) heapity(a,n,i);   }
    void heapsort(int a[], int n)
    {   builheap(a,n);
       for(int  i=n-1;i>0;i--) {swap(a[0],a[i]);
       heapity(a,i,0);
       }      }
   void xuat(int a[], int n)
   {  for (int i=0;i<n;i++) cout<<a[i]<<"  ";
   }
int main()
{       freopen("heap.inp","r",stdin);
        freopen("heap.out","w",stdout);
      cin>>n;
      for(int i=0;i<n;i++) cin>>a[i];
      heapsort(a,n);
      xuat(a, n);
```

*Độ phức tập của thuật toán là O(nlogn).

1.2. Thuật toán tìm kiếm

Có hai giải thuật tìm kiếm thường gặp :

+ Tìm kiếm tuyến tính: thường thực hiện với các mảng chưa được sắp xếp thứ tự.

+ Tìm kiếm nhị phân: thường được thực hiện với các mảng được sắp xếp thứ tự.

Xét bài toán: cho mảng a có n phần tử bất kỳ chưa sắp xếp và một số x bất kỳ. Hãy cho biết x có trong mảng a không, nếu có thì ở vị trí nào?

Ví dụ: tìm phần tử x=10 trong a_n (n=10).

16	25	1	32	2	10	11	20	34	22

Chúng ta sẽ cùng nhau tìm kiếm x thông qua 2 thuật toán tìm kiếm trên

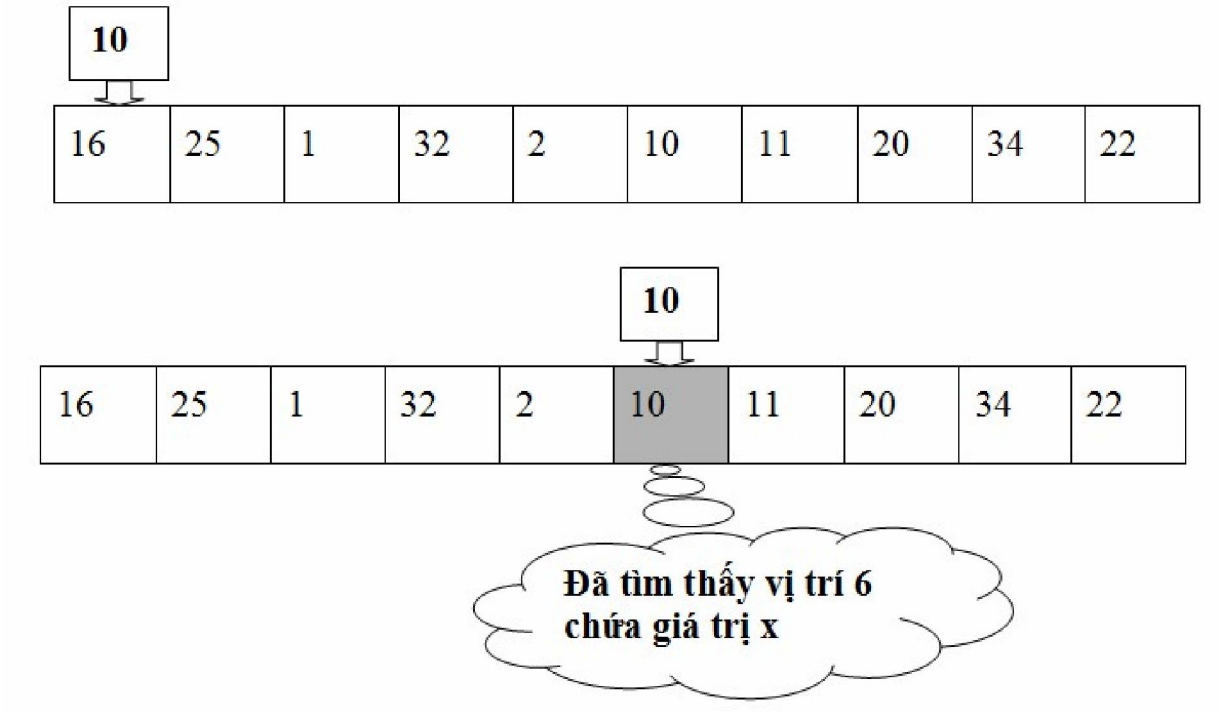
1.2.1. Tìm kiếm tuyến tính

Ý tưởng: tiến hành so sánh x lần lượt với các phần tử thứ nhất, thứ hai… của mảng a cho đến khi gặp được phần tử có giá trị x cần tìm hoặc đã tìm hết mảng mà không thấy x.

Xét ví dụ: cho dãy số: 16 25 1 32 2 10 11 20 34 22

Và giá trị X=10.

Bắt đầu đi tìm



Ta có thuật toán tìm kiếm tuyến tính như sau:

Bước 1: i=1; // bắt đầu từ phần tử đầu tiên của dãy

Bước 2: So sánh a[i] với x, có 2 khả năng:

- a[i]=x: Tìm thấy. Dừng thông báo vị trí
- a[i]!=x: Sang bước 3.

Bước 3. i=i+1; // xét tiếp phần tử kế trong mảng

- Nếu i>n: Hết mảng, không tìm thấy. Dừng thông báo

Ngược lại: Lặp lại bước 2.

***Cài đặt**

```
int kt(int x)
{  for(int i=0;i<n;i++ )
   If (a[i]==x)  return i;
 }
```

Độ phức tạp trong trường hợp xấu nhất không tìm thấy x thì khi đó độ phức tạp là O(n).

1.2.2. Tìm Kiếm nhị phân

Ý tưởng : Chỉ được thực hiện khi mảng đã được sắp xếp với ý tưởng như sau

- Tìm kiếm kiểu "tra từ điển"
- Giải thuật tìm cách giới hạn phạm vi tìm kiếm sau mỗi lần so sánh x với một phần tử trong dãy đã được sắp xếp.
- Tại mỗi bước, so sánh x với phần tử nằm ở vị trí giữa của dãy tìm kiếm hiện hành :
    - Nếu x nhỏ hơn thì sẽ tìm kiếm ở nửa trước của dãy
    - Ngược lại, tìm ở nửa sau của dãy.

Cho dãy số: 16 25 1 32 2 10 11 20 34 22

Và giá trị X=10.

Sắp xếp ta được: 1 2 10 11 16 20 22 25 32 34

Ta tiến hành tìm kiếm như sau:

Ban đầu: left=1, right=10;

mid=(1+10)/2=5; //lấy phần nguyên

so sánh a[mid]=16 > x=10

khi đó right=mid-1=4, left=1;

mid=(4+1)/2=2 //lấy phần nguyên

so sánh a[mid]=2<x=10

khi đó left=mid+1=3, right=4, mid=[3+4]/2=3;

so sánh a[mid]=10=x=10 thông báo tìm thấy

Thuật toán

Bước 1: left=1; right=n; //tìm kiếm trên tất cả các phần tử

Bước 2: mid=(left+right)/2; //lấy mốc so sánh

so sánh a[mid] với x, có 3 khả năng:

- a[mid]=x; //tìm thấy. Dừng
- a[mid]>x: //tìm tiếp x trong dãy con $a_{left}..a_{mid-1}$

  right=mid-1;
- a[mid]<x: //tìm tiếp x trong dãy con $a_{mid-1}..a_{right}$

  left=mid+1;

Bước 3: Nếu left<=right // còn phần tử chưa xét tìm tiếp. Lặp lại bước 2.

Ngược lại: Dừng // Đã xét hết tất cả các phần tử.

***Cài đặt**

```
int kt(int a[],int x)
{  int l=1,r=n;
    int mid;
     while (l<=r)
    {   mid=int(r+l)/2;
        if(a[mid]==x) return mid;
           else
                if (a[mid]>x) r=mid-1;
         else l=mid+1;
  }}
```

Sau mỗi phép so sánh, số lượng phần tử trong danh sách cần xét giảm đi một nửa. Thuật toán kết thúc khi tìm thấy hoặc số lượng phần tử còn không quá 1. Vì vậy thời gian thực thi của thuật toán là O(logn).

CHƯƠNG II: CƠ SỞ THỰC TIỄN

2.1. Thực trạng của học sinh và giáo viên trước khi chưa áp dụng đề tài

Như chúng ta biết từ nhiều năm nay, trong chương trình sách giáo khoa THPT không đi sâu vào các thuật toán sắp xếp, tìm kiếm mà chỉ dừng lại ở một hay vài phương pháp:

- Các bài học trong sách giáo khoa

Trong chương trình tin học lớp 10 kiến thức chỉ chủ yếu làm quen với môi trường lập trình bằng các thuật toán đơn giản. Chỉ trình bày phương pháp sắp xếp bằng phương pháp tráo đổi.

Trong chương trình tin học lớp 11 kiến thức chỉ chủ yếu đi lập trình giải các bài toán đơn giản trên máy tính.

- Các tài liệu dành cho học sinh yêu thích môn Tin học hay những học sinh chuyên tin thì có rất nhiều, những tài liệu ấy đi sâu vào việc giải các bài toán thực tế với các thuật toán phức tạp hơn nhiều.

Nghiên cứu, xem xét kĩ lưỡng các tài liệu trên, tôi nhận thấy:

- Các tài liệu viết một cách chung chung. Để tìm một tài liệu chi tiết, đầy đủ, dễ đọc và áp dụng là rất khó. Mặt khác các thuật toán tương đối khó nên phải dành khá nhiều thời gian cho việc ngồi trên máy tính để lập trình. Trong khi đó các em còn rất nhiều kiến thức phải học các môn khác.

- Đặc biệt việc sử dụng ngôn ngữ lập trình C đối với các em còn mới và ít tài liệu hơn so với Pascal.

Mặt khác xuất phát là một huyện miền núi nên việc bồi dưỡng học sinh giỏi càng khó khăn hơn. Từ chất lượng học sinh cho đến tài liệu bồi dưỡng ôn tập còn nhiều hạn chế. Để các em lập trình giải được các bài toán đơn giản là một vấn đề khó. Chưa nói đến các kĩ năng vận dụng và xử lí khi gặp một số bài toán lớn, phức tạp. Việc lựa chọn được học sinh để bồi dưỡng thi học sinh giỏi là một việc hết sức khó khăn đối với mỗi giáo viên bồi dưỡng.

Muốn đạt kết quả cao trong các kì thi học sinh giỏi tỉnh thì giáo viên ngoài cần phải trang bị đầy đủ, chi tiết kiến thức phần lí thuyết. Còn phải đưa ra các bài tập phù hợp để cũng cố, trang bị kiến thức, cũng như nâng cao kĩ năng vận dụng trong những bài cụ thể. Trong chương 1 tôi đã trình bày đầy đủ chi tiết về thuật toán. Ở chương 2 này tôi đưa ra những bài tập cơ bản nhằm cũng cố thêm về phương pháp cũng như nâng cao về mặt kĩ năng xử lí những bài toán có liên quan đến phương pháp sắp xếp và tìm kiếm.

2.2. Cách sử dụng thư viện có sẵn trong C++

Những bài tập sau được lập trình bằng ngôn ngữ lập trình C++, ở C++ có cho phép việc vận dụng các hàm có sẵn sẽ làm cho chương trình chạy nhanh, và giảm đi một phần công việc khi viết lập trình. Vì vậy nếu đã hiểu được các cách sắp xếp, tìm kiếm ở trên thì chúng ta có thể sử dụng một số hàm sau:

+ swap(a,b);// đổi chổ a và b

+ max(a,b); min(a,b); //tìm giá trị lớn và nhỏ của a và b

+ sort(a,a+n); //sắp xếp mảng a từ a[0..n-1] theo phương pháp Quick sort

+ sort(a+1,a+n+1); //sắp xếp mảng a từ a[1..n]

+sort_heap(a,a+n); //sắp xếp mảng a từ a[0..n-1] theo Heap sort.

Các hàm tìm kiếm chú ý hàm này chỉ hoạt động khi dãy số được sắp xếp tăng dần:

+ hàm lower_bound(first, last, value) : trả về vị trí đầu tiên lớn hơn hoặc bằng value trong dãy.

+ hàm upper_bound(first, last, value) : trả về vị trí đầu tiên lớn hơn value trong dãy.

Ví dụ: q = lower_bound(a+1, a+n+1, p) - a; // a[1..n], a[q]>=p

q = upper_bound(a+1, a+n+1, p) - a; // a[1..n], a[q]>p

+ Hàm binary_seach(a,a+n,k);// trả về giá trị 0 nếu không tìm thấy k và ngược lại trả về 1.

+ Hàm max_element(a,a+n);// trả về địa chỉ đạt giá trị lớn nhất trong mảng a.

Nếu muốn lấy giá trị lớn nhất thì ta có thể viết *max_element(a,a+n); Ngoài những hàm trên trong C++ còn có rất nhiều các hàm khác.

Các bài tập dưới đây có những chương trình tôi viết cụ thể đoạn sắp xếp, nhưng có những chương trình tôi sẽ sử dụng những thư viện có sẵn.

2.3. Bài tập ứng dụng

2.3. 1. Bài tập có lời giải

Bài 1. Dựng cọc

Có n cái cọc được đánh số thứ tự từ 1 đến n, biết di là chiều dài của cọc i (i = 1 .. n). (cho các cọc có độ dài không giống nhau). Hãy dựng các cọc trên 1 đường thẳng sao cho tổng độ lệch giữa 2 cọc liên tiếp là nhỏ nhất.

Dữ liệu vào: Từ file văn bản **COC.INP** gồm:

- Dòng đầu tiên là số n (n < 100 000)
- Từ dòng 2 trở đi ghi n giá trị di (i = 1..n)

Dữ liệu ra: Ghi vào file **COC.OUT** gồm:

- Dòng đầu tiên là tổng độ lệch nhỏ nhất.
- Từ dòng 2 trở đi ghi chỉ số các cọc được dựng theo thứ tự tìm được.

Ví dụ:

COC.INP	**COC.OUT**
6 1 8 3 12	11 2 4 1 3 5

Sắp xếp dãy a. độ chênh lệch chính là for(int i=2-n) S=s+a[i]-a[i-1];

Đưa ra vị trí ban đầu các cọc chính là mảng chỉ số ban đầu khi chưa sắp.

Code:

```
#include<bits/stdc++.h>
using namespace std;
const int maxn=10000;
int n,a[maxn],b[maxn],S[maxn],s=0;
int main()
{   freopen("coc.inp","r",stdin);

  // freopen("coc.out","w",stdout);
    cin>>n;
    for(int i=1; i<=n; i++) cin>>a[i];
```

```
for(int i=1; i<=n; i++) b[i]=1;
for(int i=1;i<n;i++)
    for(int j=i+1;j<=n; j++)
      { if(a[i]>a[j]) b[i]++;
        else b[j]++;}
//for(int i=1;i<=n; i++) cout<<b[i]<<" ";
// cout<<"\n";
for(int i=1;i<=n; i++) S[b[i]]=a[i];
for(int i=2;i<=n;i++) s=s+S[i]-S[i-1];
cout<<s<<"\n";
for(int i=1;i<=n;i++) cout<<S[i]<<" ";
return 0;
}
```

Bài 2. Xếp việc

Có N công việc cần thực hiện trên một máy tính, mỗi việc đòi hỏi đúng 1 giờ máy. Với mỗi việc ta biết thời hạn phải nộp kết quả thực hiện sau khi hoàn thành việc đó và tiền thưởng thu được nếu nộp kết quả trước hoặc đúng thời điểm quy định. Chỉ có một máy tính trong tay, hãy lập lịch thực hiện đủ N công việc trên máy tính sao cho tổng số tiền thưởng thu được là lớn nhất và thời gian hoạt động của máy là nhỏ nhất. Giả thiết rằng máy được khởi động vào đầu ca, thời điểm t = 0 và chỉ tắt máy sau khi đã hoàn thành đủ N công việc.

Dữ liệu vào: tệp văn bản **VIEC.INP**:

- Dòng đầu tiên là số N.
- N dòng tiếp theo: mỗi việc được mô tả bằng hai số tự nhiên, số thứ nhất là thời hạn giao nộp, số thứ hai là tiền thưởng. Các số cách nhau bởi dấu cách.

Dữ liệu ra: tệp văn bản **VIECC.OUT**:

- N dòng đầu tiên, dòng thứ t ghi một số tự nhiên i cho biết việc thứ i được làm trong giờ t.
- Dòng cuối cùng ghi tổng số tiền thu được.

Với thí dụ trên, tệp **VIEC.OUT** sẽ như sau:

Ví dụ:

VIEC.INP	VIEC.OUT
4	4
1 15	2
3 10	3
5 100	1
1 27	137

Ý tưởng: Ta ưu tiên cho những việc có tiền thưởng cao, do đó ta sắp các việc giảm dần theo tiền thưởng. Với mỗi việc k ta đã biết thời hạn giao nộp việc đó là h = t[k]. Ta xét trục thời gian b. Nếu giờ h trên trục đó đã bận do việc khác thì ta tìm từ thời điểm h trở về trước một thời điểm có thể thực hiện được việc k đó. Nếu tìm được một thời điểm m như vậy, ta đánh dấu bằng mã số của việc đó trên trục thời gian b, b[m]= k. Sau khi xếp việc xong, có thể trên trục thời gian còn những thời điểm rỗi, ta dồn các việc đã xếp về phía trước nhằm thu được một lịch làm việc trù mật, tức là không có giờ trống. Cuối cùng ta xếp tiếp những việc trước đó đã xét nhưng không xếp được. Đây là những việc phải làm nhưng không thể nộp đúng hạn nên sẽ không có tiền thưởng. Với thí dụ đã cho, N = 4, thời hạn giao nộp t = (1, 3, 5, 1) và tiền thưởng a = (15, 10, 100, 27) ta tính toán như sau:

- Khởi trị: trục thời gian với 5 thời điểm ứng với Tmax = 5 là thời điểm muộn nhất phải nộp kết quả, Tmax = max { thời hạn giao nộp }, b = (0, 0, 0, 0,0).
- Chọn việc 3 có tiền thưởng lớn nhất là 100. Xếp việc 3 với thời hạn t[3] = 5 vào h: h[5] = 3. Ta thu được h = (0, 0, 0, 0, 3).
- Chọn tiếp việc 4 có tiền thưởng 27. Xếp việc 4 với thời hạn t[4] = 1 vào h: h[1] = 4. Ta thu được h = (4, 0, 0, 0, 3).
- Chọn tiếp việc 1 có tiền thưởng 15. Xếp việc 1 với thời hạn t[1] = 1 vào h: Không xếp được vì từ thời điểm 1 trở về trước trục thời gian h[1..1] đã kín. Ta thu được h = (4, 0, 0, 0, 3).
- Chọn nốt việc 2 có tiền thưởng 10. Xếp việc 2 với thời hạn t[2] = 3 vào h: h[3] = 2.
- Ta thu được h = (4, 0, 2, 0, 3).
- Dồn việc trên trục thời gian h, ta thu được h = (4, 2, 3, 0, 0).
- Xếp nốt việc phải làm mà không có thưởng, ta thu được h = (4, 2, 3, 1).
- Ca làm việc kéo dài đúng N = 4 giờ.

- Dùng mảng cs để lưu lại vị trí công việc.

***Code:**

```
#include<iostream> using
namespace std;
int a[100],b[100],cs[100],d[100],h[1000],n,l=0,tt=0;
void nhap()
{   cin>>n;
    for(int i=1;i<=n;i++) { cin>>a[i]>>b[i]; cs[i]=i;}
    fill(h+1,h+n+1,0);
      }
void qsort(int b[],int l,int r)
{   int i=l, j=r;
       int m=int(i+j)/2;
       while (i<=j)
       {       while (b[i]>b[m]) i++;
               while (b[j]<b[m]) j--;
               if (i<=j)
          {  swap(b[i],b[j]);
             swap(cs[i],cs[j]);
             i++; j--;
          }   }
       if (i<r) qsort(b,i,r);
       if (j>l) qsort(b,l,j);
    }
    void xv()
    {    int v;
       for(int i=1;i<=n;i++)
       {  v=cs[i];
          for(int k=a[v];k>0;k--)
          if (h[k]==0)
              {h[k]=v; l++; d[l]=v; tt=tt+b[i]; break; }
   } }
```

```
void xuat()
{   for(int i=l;i>0;i--) cout<<d[i]<<" ";
cout<<tt<<"    "<< l<< "\n";
}
int main()
{       freopen("viec.inp","r",stdin);
        freopen("viec.out","w",stdout);nhap();
        qsort(b,1,n);    xv();    xuat();    return 0;
}
```

Bài 3. Đua ngựa

Ở thời xuân thu, vua Tề và Điền Kỳ thường hay tổ chức đua ngựa từng cặp với nhau. Vua và Điền Kỳ mỗi người đua ra N con ngựa, đánh số từ 1 đến N. mỗi một con ngựa có một hệ số khác nhau. Một trận đấu mỗi người đưa ra một con ngựa để thi đấu. Trong cuộc đua, con ngựa vào có hệ số cao hơn thì sẽ thắng, nếu cùng hệ só thì Điền kỳ bí mật nhường cho vua thắng. mỗi một con ngựa chỉ được đấu đúng một trận. Ai có tổng số trận thắng nhiều hơn thì sẽ thắng chung cuộc. Bạn hãy giúp Điền Kỳ sắp xếp các lượt đấu để đạt số trận thắng cao nhất có thể.

Dữ liệu vào: Horse.inp

- Dòng đầu là số lượng ngựa (n<100000).
- Dòng thứ hai có n số, số thứ I là hệ số của con ngựa thứ I của Điền Kỳ.
- Dòng thứ ba có n số, số thứ I là hệ số của con ngựa thứ I của Vua Tề.

Kết quả ra: horse.out

- Ghi duy nhất một số là số trận thắng tối đa đạt được của Điền Kỳ.

Ví dụ:

Horse.inp	Horse.out
3 4 6 2 9 3 5	2

Hạn chế thời gian: 1s,

Cách giải:

- Ban đầu ta chỉ cần sắp xếp các hệ số con ngựa của Vua Tề và Điền Kỳ
- Sau đó so sánh lần lượt từng số hiệu con ngựa của Điền Kỳ với số hiệu con ngựa của Vua

Tề, nếu con thứ 1 của Điền Kỳ có số hiệu lớn hơn con thứ j của Vua Tề thì số trận thắng của Điền Kỳ tăng lên.

- Sau đó tiếp tục so sánh số hiệu con thứ i+1 của Điền Kỳ với số hiệu con thứ j+1 của Vua , nếu Không lớn hơn thì lại so sánh con thứ i+2 của Điền kỳ với số hiệu con thứ j+1 của Vua.

Và cứ như vậy cho đến khi hết số ngựa của Điền Kỳ.

Code:

```
#include<iostream>;
using namespace std;
int a[100],b[100],n;
void nhap(int c[],int k)
{   for(int i=0;i<k;i++)  cin>>c[i];}
    int main()
    {  cin>>n;       nhap(a,n);       sort(a,a+n);
       nhap(b,n);       sort(b,b+n);       int d=0;
       int i=0, j=0;
       while ((i<n)&&(j<n))
       {if (a[i]>b[j]) {d++; i++; j++;}
          else     i++;
       }    cout<<d;
       return 0;
  }
```

Ta thấy bài toán trên độ phức tạp của đoạn code chính chỉ có O(n). Nhưng độ phức tạp của đoạn sắp xếp lại có thể là $O(n^2)$ hoặc O(nlogn) do ta lựa chọn thuật toán. Chính vì vậy độ phức tạp của thuật toán là phụ thuộc vào đoạn sắp xếp.

- Chúng ta lựa chọn thư viện sort() để sắp xếp khi độ độ phức tạp chính phụ thuộc vào đoạn này.

Bài 4. Đoạn gối

Cho N đoạn thẳng trên trục số với các điểm đầu a_i và điểm cuối b_i là những số nguyên trong khoảng −1000..1000, $a_i < b_i$. Liệt kê tối đa K đoạn thẳng gối nhau liên tiếp. Hai đoạn thẳng [a,b] và [c,d] được gọi là gối nhau nếu xếp chúng trên cùng một trục số thì điểm đầu đoạn này trùng với điểm cuối của đoạn kia, tức là

c = b hoặc d = a.

Dữ liệu vào: tệp văn bản **DOAN.INP**:

- Dòng đầu ghi số N
- N dòng tiếp theo, mỗi dòng ghi 2 số a_i và b_i

Dữ liệu ra: tệp văn bản **DOAN.OUT** Ghi số tự nhiên K.

Ví dụ:

DOAN.INP	**DOAN.OUT**
5 2 7 1 3 7 9 3 4 4 5	3

Ý tưởng: ta áp dụng phương pháp quy hoạch động + Tham lam.

Giả sử các đoạn được sắp tăng theo đầu phải b. Kí hiệu c(i) là số lượng tối đa các đoạn thẳng gối nhau tạo thành một dãy nhận đoạn i làm phần tử cuối dãy (khi khảo sát các đoạn từ

1..i).

Ta có: c(1) =1,

Với i = 2...N: $c(i) = \max \{ c(j) \mid 1 \le j < i, \text{Đoạn j kề trước đoạn i: } b_j = a_i \} + 1$.

Lợi dụng các đoạn sắp tăng theo đầu phải b, với mỗi đoạn i ta chỉ cần duyệt ngược các đoạn j đứng trước đoạn i cho đến khi phát hiện bất đẳng thức $b_j < a_i$.

Kết quả: $K = \max \{ c(i) \mid i = 1...n \}$.

Độ phức tạp: cỡ N^2 vì với mỗi đoạn ta phải duyệt tối đa tất cả các đoạn đứng trước đoạn đó.

Code:

```
#include<iostream>
#include<fstream>
using namespace std;
ifstream fi ("doangoi.inp");
ofstream fo ("doangoi.out");
int a[100],b[100],n, c[100],cs[100],t[100],k;
void nhap()
{    cin>>n;
for(int i=1;i<=n;i++) { cin>>a[i]>>b[i]; cs[i]=i;} }
void qsort(int l, int r)
{    int i=l,j=r;
   int m=int((i+j)/2);
   while (i<=j)
   {  while (b[i]<b[m]) i++;
      while (b[j]>b[m]) j--;
       if (i<=j)
      {  swap(b[i],b[j]); swap(cs[i],cs[j]);
       swap(a[i],a[j]);
        i++;j--;
      }
   }    if (i<r) qsort(i,r);
    if (j>l) qsort(l,j);
 }
void xuli()
{   c[1]=1;  t[1]=1;
    for (int i=2;i<=n;i++)
   {   c[i]=0;
       for (int j=i-1;j>=1;j--)    {f (a[i]>b[j]) (t[i]=i; break;}
```

```
        if (a[i]==b[j])
         if (c[j]>c[i]) {c[i]=c[j]; t[i]=j;}
         }c[i]++;
   } }
void xuat()
{   int  imax=1;
        for(int i=2;i<=n;i++)if (c[imax]<c[i]) { c[imax]=c[i]; imax=i;}
         cout<<c[imax]<<endl;            }
}
int main()
{   nhap();    qsort(1,n);    xuli();    xuat();     Return 0;
 }
```

Bài 5. Băng nhạc

Người ta cần ghi N bài hát, được mã số từ 1 đến N, vào một băng nhạc có thời lượng tính theo phút đủ chứa toàn bộ các bài đã cho. Với mỗi bài hát ta biết thời lượng phát của bài đó. Băng sẽ được lắp vào một máy phát nhạc đặt trong một siêu thị. Khách hàng muốn nghe bài hát nào chỉ việc nhấn phím ứng với bài đó. Để tìm và phát bài thứ i trên băng, máy xuất phát từ đầu cuộn băng, quay băng để bỏ qua i – 1 bài ghi trước bài đó. Thời gian quay băng bỏ qua mỗi bài và thời gian phát bài đó được tính là như nhau. Tính trung bình, các bài hát trong một ngày được khách hàng lựa chọn với số lần (tần suất) như nhau. Hãy tìm cách ghi các bài trên băng sao cho tổng thời gian quay băng trong mỗi ngày là ít nhất.

Dữ liệu vào: được ghi trong tệp văn bản tên **BANGNHAC.INP.**

- Dòng đầu tiên là số tự nhiên N cho biết số lượng bài hát.
- Tiếp đến là N số nguyên dương thể hiện dung lượng tính theo phút của mỗi bài.

Mỗi đơn vị dữ liệu cách nhau qua dấu cách.

Dữ liệu ra: được ghi trong tệp văn bản **BANGNHAC.OUT** theo dạng thức sau:

- N dòng đầu tiên thể hiện trật tự ghi bài hát trên băng: mỗi dòng gồm hai số nguyên dương j và d cách nhau bởi dấu cách, trong đó j là mã số của bài hát cần ghi, d là thời gian tìm và phát bài đó theo trật tự ghi này.

- Dòng thứ n + 1 ghi tổng số thời gian quay băng nếu mỗi bài hát được phát một lần trong ngày.

Ví dụ:

BANGNHAC.INP	**BANGNHAC.OUT**
3 7 2 3	2 2 3 5 1 12 19

Ý tưởng: Để có phương án tối ưu ta chỉ cần ghi băng theo trật tự tăng dần của thời lượng. Bài toán được cho với giả thiết băng đủ lớn để ghi được toàn bộ các bài. Dễ dàng sửa chương trình để vận dụng cho trường hợp dung lượng tăng hạn chế trong M phút. Chương trình sắp xếp dữ liệu theo chỉ dẫn.

Code:

```
#include<iostream>
#include<fstream>
 using namespace std;
ofstream fo("bd.out");
ifstream fi("bd.inp");
 int a[100],cs[100],n,t=0,tt=0;
void nhap() //tu viet
void qsort(int l,int r)
{    int i=l,j=r;
     int m=((i+j)/2);
     while (i<=j)
     {   while (a[m]>a[i]) i++;
         while (a[m]<a[j]) j--;
         if (i<=j)
         { swap(a[i],a[j]);
          swap(cs[i],cs[j]);
          i++; j--;
             }    }
```

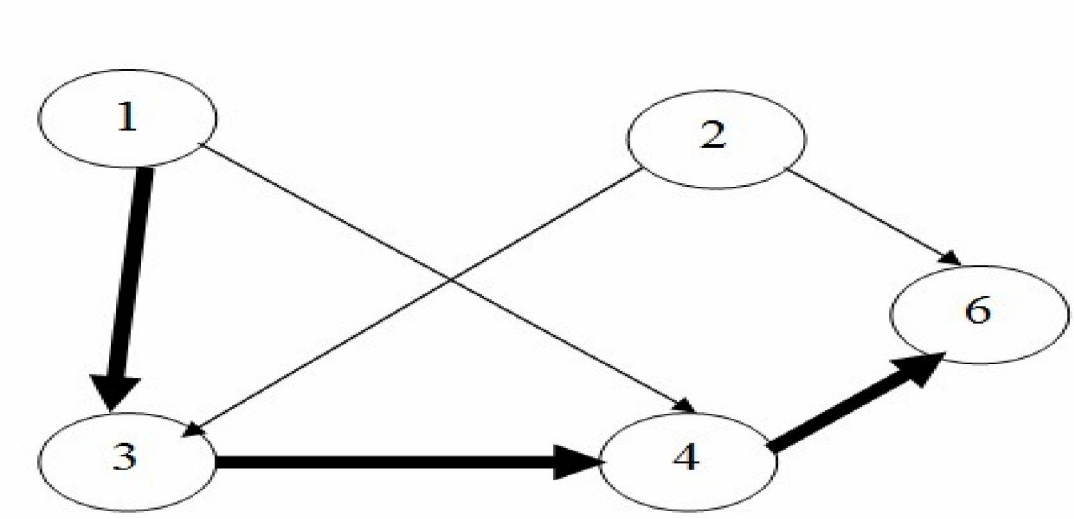
```
if(i<r) qsort(i,r);
   if(j>l) qsort(l,j);
}
void xuat()
{   qsort(1,n);
    for(int i=1;i<=n;i++)
   {   t=t+a[i]; fo<<cs[i]<<"  "<<t<<"\n";
       tt=tt+t;
   }    fo<<tt<<"  ";
```

Vậy độ phức tạp của bài toán trên phụ thuộc vào đoạn sắp xếp.

Bài 6. Lo cò

Nhảy lò cò là trò chơi dân gian của Việt Nam, Người trên hành tinh X cũng rất thích trò chơi này và họ đã cải biên trò chơi này như sau: trên mặt phẳng vẽ n vòng tròn được đánh số từ 1 đến n. Tại vòng tròn i người ta điền số nguyên dương a_i. hai số trên hai vòng tròn tùy ý không phân biệt nhất thiết phải khác nhau. Tiếp đến đến người ta vẽ các mũi tên là: nếu có ba số a_i, a_j, a_k thõa mãn $a_k = a_i+a_j$ thì vẽ mũi tên hướng từ vòng tròn i đến vòng tròn k và mũi tên hướng từ vòng tròn j đến vòng tròn k. Người chơi chỉ được di chuyển từ một vòng tròn đến một vòng tròn khác nếu có mũi tên xuất phát từ một trong số các vòng tròn, di chuyển theo cách mũi tên đã vẽ để đi đến các vòng tròn khác. Người thắng cuộc sẽ là người tìm được cách di chuyển qua nhiều vòng tròn nhất.

Ví dụ: với 5 vòng tròn và các số trong vòng tròn là 1, 2, 6, 4, 3, trò chơi được trình bày trong hình dưới đây:



Khi đó có thể di chuyển được nhiều nhất qua 4 vòng tròn (tương ứng với đường di chuyển được tô đậm trên hình vẽ).

Yêu cầu: Hãy xác định xem trong trò chơi mô tả ở trên nhiều nhất có thể di chuyển được qua bao nhiêu vòng tròn?

Dữ liệu vào:

- Dòng đầu chứa số nguyên n (3<=n<=1000);

- Dòng thứ hai chứa dãy số nguyên dương $a_1, a_2, \ldots a_n$ (a_i <=109, i=1, 2, …n). Hai số liên tiếp trên một dòng được ghi cách nhau bởi dấ u cách.

Dữ liệu ra:

Ghi ra số lượng vòng tròn trên đường di chuyển tìm được Rằng buộc: 60 % tests ứng với 60% số điểm của bài có 3<=n<=100

Ví dụ:

INPUT	**OUTPUT**
5 1 2 6 4 3	4

Ý tưởng: từ đường tròn j để di chuyển sang vòng tròn i thì a[i]>a[j] , nên khi đó việc nhập dữ liệu không ảnh hưởng đến giá trị tối ưu của bài toán.

- Ta sắp xếp các giá trị trong mảng a theo thứ tự tăng dần.

- Với quy tắc di chuyển này mỗi đơn vị dữ liệu là một vòng tròn, ứng với một số trong dãy đã sắp xếp khi xét đến đường tròn thứ i để nó là điểm cuối của dãy nhiều vòng tròn nhất ta thêm nó vào sau dãy con dài nhất có thể đi đến các vòng tròn phía trước.

Code:

```
#include<bits/stdc++.h>
using namespace std;
int n,a[1000],f[1000];
void xl()
{   for(int i=0;i<n;i++) f[i]=1;
    for (int i=2;i<n;i++)
     {for (int j=i-1;j>=1;j--)
      if (binary_search(a,a+i,a[i]-a[j])) f[i]=max(f[i],f[j]+1);
     } }
```

```
void xuat()
{   cout<<*max_element(f,f+n); //đưa ra giá trị max trong mảng f
}
int main()
{   freopen("loco.inp","r",stdin);
    freopen("loco.out","w",stdout);
    cin>>n;
    for(int i=0;i<n;i++) cin>>a[i];
    sort(a,a+n);    xl();   xuat();   return 0;
}
```

2.3.2. Bài tập tự giải

Bài 1. Bước nhảy xa nhất

Cho dãy A gồm N số nguyên không âm A_1, A_2,…,A_N. Một bước nhảy từ phần tử A_i đền phần A_j được gọi là bước nhảy xa nhất của dãy nếu thỏa mãn các điều kiện sau:

+$1 \le i < j \le N$.

+$A_j - A_i \ge P$.

+j-i lớn nhất(Khi đó j-i được gọi là độ dài bước nhảy xa nhất của dãy).

Yêu cầu: Tìm độ dài bước nhảy xa nhất của dãy A.

Dữ liệu vào: Từ tệp **JUMP.INP** có cấu trúc như sau:

-Dòng 1: Gồm hai số nguyên N và P($1 \le N \le 10^5$; $0 \le\le P \le 10^9$).

-Dòng 2 : Gồm N số nguyên $A_1,A_2,...,A_n$($0 \le A_i \le 10^9$ với $1 \le i \le N$).

Kết quả: Ghi vào tệp **JUMP.OUT** gồm một số nguyên dương duy nhất là độ dài của bước nhảy xa nhất của dãy (Nếu không có bước nhảy nào thỏa mãn thì ghi kết quả bằng 0).

Ví dụ:

JUMP.INP	JUMP.OUT
6 3 4 3 7 2 6 4	3

Bài 2. Khoảng Cách

Yêu cầu: Cho n số nguyên , hãy tính khoảng cách nhỏ nhất giữa 2 số nguyên bất kì.

Dữ liệu vào: Từ tệp **KHOANGCACH.INP** gồm

- Dòng đầu tiên ghi số nguyên dương n ($n \leq 100000$).
- Dòng tiếp theo ghi n số nguyên a[i] ($|a[i]| \leq 1e6$).

Dữ liệu ra: ghi vào tệp **KHOANGCACH.OUT** là:

- Kết quả bài toán.

Ví dụ:

KHOANGCACH.INP	KHOANGCACH.OUT
3 1 5 2	1

Bài 3. Bài toán cái túi (với số lượng vật không hạn chế)

Cho một cái túi có thể đựng một trọng lượng không quá W và n đồ vật, mỗi đồ vật i có trọng lượng là a_i và giá trị là c_i , tất cả các đồ vật đều có số lượng không hạn chế. Tìm một cách lựa chọn các đồ vật đựng vào ba lô, chọn các đồ vật nào, mỗi loại lấy bao nhiêu, sao cho tổng trọng lượng không vượt quá W và tổng giá trị là lớn nhất.

Dữ liệu vào: từ file **CAITUI.INP**

- Dòng đầu ghi n (n<1000), W
- N dòng tiếp theo, mỗi dòng ghi hai số a_i và c_i (i=1..n).

Dữ liệu ra: từ file **CAITUI.OUT** các đồ vật và số lượng các đồ vật được lấy

Ví dụ:

CAITUI.INP	CAITUI.OUT
4 37 15 30 10 25 2 2 4 6	2 3 3 1 4 1

Bài 4. Đoạn bao nhau

Cho N đoạn thẳng trên trục số với các điểm đầu a_i và điểm cuối b_i là những số nguyên trong khoảng −1000..1000, $a_i < b_i$, i = 1..n. Tìm K là số lượng nhiều đoạn nhất tạo thành một dãy các đoạn bao nhau liên tiếp. Hai đoạn [a,b] và [c,d] được gọi là bao nhau nếu đoạn này nằm lọt trong đọan kia, tức là $a \le c < d \le b$ hoặc $c \le a < b \le d$.

Dữ liệu vào: tệp văn bản **DOAN.INP**:

- Dòng đầu ghi số N

- N dòng tiếp theo, mỗi dòng ghi hai số ai và bi.

Dữ liệu ra: tệp văn bản **DOAN.OUT** Chứa duy nhất một số tự nhiên K.

Ví dụ:

DOAN.INP	**DOAN.OUT**
6 1 12 8 10 8 11 2 7 17 18 13 20	3

Bài 5. Phủ đoạn

Cho N đoạn thẳng trên trục số với các điểm đầu a_i và điểm cuối b_i là những số nguyên trong khoảng −1000..1000, $a_i < b_i$. Hãy chỉ ra ít nhất K đoạn thẳng sao cho khi đặt chúng trên trục số thì có thể phủ kín đoạn [x, y] với tọa độ nguyên cho trước.

Dữ liệu vào: tệp văn bản **DOAN.INP**: xem bài trước

- Dòng 1 chứa N,

- N dòng tiếp theo mỗi dòng chứa hai số a_i và b_i là điểm đầu và điểm cuối.

Dữ liệu ra: tệp văn bản **DOAN.OUT**

- Dòng đầu tiên: số K, nếu vô nghiệm K = 0.

Tiếp theo là K số tự nhiên biểu thị chỉ số của các đoạn thẳng phủ kín đoạn [x, y].

Ví dụ:

DOAN.INP	DOAN.OUT
5 3 23 1 15 3 10 8 20 17 25 2 7	3 1 3 4

Bài 6. Số nguyên nhỏ nhất

Cho dãy gồm N (N ≤ 30000) số tự nhiên không vượt quá 10^9, tìm số tự nhiên nhỏ nhất không xuất hiện trong dãy.

Dữ liệu vào: Từ file **SN.INP** gồm*:*

- Dòng đầu là số nguyên N**.**
- Dòng thứ hai gồm N số.

Dữ liệu ra: file **SN.OUT** có dạng: số tự nhiên nhỏ nhất không xuất hiện trong dãy.

Ví dụ:

SN.INP	SN.OUT
5 5 0 3 1 4	2

Bài 7. Cỏ Dại

Tèo có một vườn rau nhưng dạo gần đây cỏ dại mọc lên ùn ùn nhỗ mãi không hết. Không biết làm thế nào, Tèo bèn nhờ Tí, một kĩ sư nghiên cứu về cỏ dại. Tí cho hay vườn của Tèo có hai cây cỏ dại vương – cây này phân phát hạt mầm, bảo vệ các cây cỏ dại khác, chỉ cần Tèo tìm được nó và nhổ nó lên thì đám cỏ dại kia sẽ từ từ biến mất. Nghe thấy vậy, Tèo liền về nhà đi tìm cây cỏ dại vương.

Vườn nhà Tèo là một đường thẳng có *N* cây cỏ dại, cây cỏ dại được đánh số từ 1 đến *N*, cây cỏ dại thứ *i* có độ dẻo dai *Wi*. Hai cây *x, y* là cỏ dại vương nếu |*Wx – Wy*| là lớn nhất. Bạn hãy giúp Tèo tìm |*Wx – Wy*| lớn nhất.

Dữ liệu vào: Từ tệp **Codai.inp** gồm:

• Dòng đầu tiên, chứa một số nguyên dương N ($1 \le N \le 10^6$)

• Dòng thứ 2, chứ N số nguyên Wi ($1 <= i <= N$) là độ dẻo dai của N cây cỏ. ($1 \le Wi \le 10^9$.)

Dữ liệu ra: ghi vào tệp **Codai.out**

• In ra một số nguyên duy nhất là kết quả của bài toán.

Ví dụ:

Codai.inp	**Codai.out**
3 1 2 3	2

Bài 8. Dãy số

Cho dãy số nguyên $a_1, a_2, \ldots a_n$. Số a_p ($1 \le e \le n$) được gọi là một số trung bình cộng trong dãy nếu tồn tại 3 chỉ số i, j, k ($1 \le i, j, k \le n$) đôi một khác nhau, sao cho $a_p = (a_i + a_j + a_k)/3$

Yêu cầu: Cho n và dãy số $a_1, a_2, \ldots a_n$. Hãy tìm số lượng các số trung bình cộng trong dãy.

Dữ liệu vào: Từ tệp **Dayso.inp** gồm:

- Dòng đầu ghi số nguyên dương n ($3 \le n \le 1000$).
- Dòng thứ hai chứa n số nguyên a_i ($|a_i| < 10^8$).

Dữ liệu ra: Lưu vào tệp **Dayso.out** là:

- Số lượng các số trung bình cộng trong dãy.

Ví dụ:

Dayso.inp	**Dayso.out**
5 4 3 6 3 5	2

Bài 9. Đoạn rời

Cho N đoạn thẳng với các điểm đầu a_i và điểm cuối b_i là những số nguyên trong khoảng −1000..1000, $a_i < b_i$. Liệt kê số lượng tối đa các đoạn thẳng rời nhau. Hai đoạn được xem là rời nhau nếu chúng không có điểm chung. Các đoạn có dạng như trong bài Phủ đoạn 2.

Dữ liệu vào: tệp văn bản **DOAN.INP**

- Dòng đầu tiên: số tự nhiên N > 1.

. - N dòng tiếp theo, mỗi dòng ghi 2 số a_i và bi.

Dữ liệu ra: tệp văn bản **DOAN.OUT**

-Dòng đầu tiên: số K.

-Tiếp theo là K số tự nhiên biểu thị chỉ số của các đoạn thẳng rời nhau cách nhau bằng dấu cách.

Ví dụ:

DOAN.INP	DOAN.OUT
8 -2 3 3 5 8 12 13 15 3 9 4 5 5 8 7 15	5 1 2 7 3 4

2.4. Kết quả sau khi áp dụng đề tài

Đề tài đã được triển khai và áp dụng trong việc bồi dưỡng học sinh giỏi tỉnh trong các năm học vừa qua. Sau khi đề tài triển khai, ngoài trang bị kiến thức, kĩ năng xử lí các bài toán phức tạp cho học sinh. Qua đó cũng cố thêm kiến thức cho giáo viên bồi dưỡng và đồng nghiệp. Kết quả giành được nhiều tín hiệu tích cực từ nhiều phía. Đặc biệt là trong các đợt thi học sinh giỏi tỉnh luôn giành được các giải cao và luôn đứng đầu bảng B. Trong kì thi học sinh giỏi tỉnh năm học 2020-2021 vừa qua. Em học sinh: Nguyễn Mạnh Đạt giải nhất bảng B với 16 điểm.

PHẦN III. KẾT LUẬN

1. Với mục tiêu đề ra đề tài đã làm được

Đề tài được trình bày trong 2 phần chính. Nêu ra thực trạng, những vấn đề cần giải quyết hiện nay mà một giáo viên bồi dưỡng học sinh giỏi cần phải chú ý. Trình bày cụ thể, đầy đủ, chi tiết về mặt lí thuyết của phương pháp sắp xếp và tìm kiếm. Ngoài ra còn lấy những bài toán cơ bản nhằm nâng cao kĩ năng cũng như vận dụng phương pháp một cách hiệu quả.

2. Hướng phát triển của đề tài

Trong thời gian tới tôi sẽ tập trung phát triển đề tài hơn nữa, xây dựng lại hệ thống lý thuyết một cách chặt chẽ, hợp lý hơn. Đồng thời xây dựng chỉnh sửa lại, mở rộng nội dung, dễ triển khai, thân thiện hơn. Nhằm mang lại nhiều kết quả cao trong các kì thi học sinh giỏi cũng như thi vào lớp 10 chuyên tin trường Phan Bội Châu. Ngoài ra chia sẽ đề tài một cách rộng rãi đến các giáo viên trong và ngoài nhà trường. Giúp giáo viên bồi dưỡng có thêm lựa chọn về nội dung, phương pháp trong việc ôn tập và bồi dưỡng học sinh giỏi một cách dễ dàng và hiệu quả hơn.

3. Kiến nghị và đề xuất

Để áp dụng đề tài hiệu quả trong quá trình bồi dưỡng giáo viên cần bồi dưỡng nhiều hơn nữa về lí thuyết sắp xếp và tìm kiếm. Xem và làm các bài toán từ cơ bản, điển hình cho đến nâng cao. Một số thao tác xử lí bằng phương pháp sắp xếp và các phương pháp khác để ứng dụng phương pháp này cũng như kết hợp các phương pháp với phương pháp này một cách hiệu quả nhất.

Hướng dẫn, khuyến khích học sinh tham gia giải bài, tìm kiếm các bài tập có phân dạng về xử lí bằng phương pháp sắp xếp và tìm kiếm trên các trang mạng giải bài trực tuyến (đặc biệt trên trang vn.spoj.com; laptrinhphothong.vn) và các đề thi học sinh giỏi qua các năm.

Do thời gian cũng như năng lực cá nhân có hạn, nên đề tài còn có một số thiếu sót, xây dựng các nội dung, phương pháp và cách thức mới chỉ đáp ứng được yêu cầu cơ bản của đề tài đưa ra. Chính vì vậy rất mong nhận được những lời góp ý, nhận xét từ cán bộ quản lí nhà trường, từ Ban Giám khảo và từ các bạn đồng nghiệp, để phát huy những mặt mạnh, điều chỉnh, khắc phục những thiếu sót cho đề tài được hoàn thiện hơn.

Lời cuối cho phép tôi xin chân thành cảm ơn ban giám hiệu nhà trường, các thầy cô bộ môn, các em học sinh trên địa bàn đã giúp đỡ và tạo điều kiện để tôi hoàn thành và triển khai đề tài một cách hiệu quả nhất.

TÀI LIỆU THAM KHẢO

1. Hồ Sĩ Đàm_Tài liệu giáo khoa chuyên tin quyển 1
2. Hồ Sĩ Đàm _Sách giáo khoa Tin học 11
3. Lê Minh Hoàng_ Chuyên đề tin
4. Đỗ Xuân Lôi - Cấu trúc dữ liệu và giải thuật
5. Nguyễn Thanh Thủy_Ngôn ngữ lập trình C
6. Các trang Internet
    - http://vnoi.info/
    - http://vn.spoj.com
    - http://laptrinhphothong.vn