Sort books via insertion sort

Create a class book with title, author name, access number, price. Modify the insertion sort in the template code to

(i) Sort books in ascending order as per access number

(ii) Use vector

(iii) do minimal changes to the given code

```
#include<iostream>
#include<vector>
using namespace std;
class book
{
    public:
    string title,author_name;
    int access_no;
    float price;
};
int main()
{
    int i,j,n;
    book key;
    vector<book> elements;
    //cout<<"Enter number of elements";
    cin>>n;
    for(i=0;i<n;i++)
    {
        cin>>key.title;
        cin>>key.author_name;
        cin>>key.access_no;
        cin>>key.price;
        elements.push_back(key);
    }
    for(j=1;j<n;j++)
    {
        key = elements[j];
        i = j-1;
        while((i>=0)&&(elements[i].access_no>key.access_no))
        {
            elements[i+1] = elements[i];
            i = i-1;
        }
        elements[i+1] = key;
    }
   for(i=0;i<n;i++)
    {

cout<<elements[i].title<<"\n"<<elements[i].author_name<<"\n"<<elements[i].access_no<<"\n"<<
elements[i].price<<"\n";
    }
    cout<<"\nabc";
}
```

Bidirectional Selection Sort Students

Selection sort technique has the core idea of choosing the apt element for the ith position. This technique will be having 'i' sorted elements from the beginning of the array after 'i' iterations. In each iteration, this technique basically find the minimum element from the unsorted array. Bidirectional Selection sort is a variation of selection sort which sort the elements from both the directions. We will have two indices pointing to the extreme ends and we move them in opposite direction till they cross each other. The algorithm works as below:

1. Let i = 1 and j = n

2. Repeat step 2 to 10 when i < j

3. Initialize min = max = ith element

4. Initialize min_index and max_index to i

5. Let k = i

6. while k<=j do steps 7 and 8

7. if element in kth position is greater than max then make max as kth element and max_index as k

8. Else if the element in kth position is less than max then make min as kth element and min_index as k

9. swap element at ith position and min

10. swap eleeent at jth position and max

Consider a vector of student records as shown in the post code and sort them based on roll no using Bidirectional Selection sort. Understand the precode of Bidirectional Selection sort given and complete

Note:

1. Use getline(cin,s) to read string in this code

2. Use cin.ignore() to clear one or more characters from the input buffer if normal cin was used to read previous input

Input Format

First line cotains the number of students, n

Next 3*n lines contain details of students in the order name, rollno and marks of each student

Output Format

Print details of students sorted as per rollnp in ascending order print name, rollno and marks of each student in order

```
#include<iostream>
using namespace std;
#include<vector>
class student
{
     string name;
     int rollno;
     int marks;
     public:
     friend istream& operator>>(istream&, student&);
     friend ostream& operator<<(ostream&,student&);
     bool operator<(student&);
     bool operator>(student&);
};
// values are passed by reference
```

```
void swap(int &a, int & b)
{
          int temp;
          temp = a;
          a = b;
          b = temp;

  }

// Vector is passed by reference
void bidirectional_selection_sort(vector<student>& elements)
{
          int n = elements.size(),i=0,j=n-1,min_i,max_i,k,p;
          student max,min;
          //Initialize min and max to first elements of unsorted array
          // Move i in forward and j in reverse direction
          while(i<=j)
          {
                    //Initialize min and max to first elements in the same for loop
                    min = elements[i];
                    max = elements[i];
                    //min index and max index to 0
                    min_i = i;
                    max_i = i;
                    // Move k from index i to index j (inclusive)
                    k=i;
                    while(k<=j)
                    {
                              if(elements[k]>max)
                              {
                                        max = elements[k];
                           max_i = k;
                       }
                              else if(elements[k]<min)
                              {
             min = elements[k];
             min_i = k;
                              }
                              k++;
                    }
                    /*swap(elements[i], elements[min_i]);
                    swap(elements[j], elements[max_i]);*/
                    student temp;
```

```
                // swap element in the first position of unsorted array to minimum element
                temp=elements[i];
                elements[i]=elements[min_i];
                elements[min_i]=temp;

                // and element in last position of unsorted array to maximum element
                temp=elements[j];
                elements[j]=elements[max_i];
                elements[max_i]=temp;

                i++;
                j--;
        }
        cout<<"\nabc\n\n";
}
istream& operator>>(istream& in,student& s)
{
   cin.ignore();
   getline(in,s.name);
   in>>s.rollno>>s.marks;
   return in;
}
ostream& operator<<(ostream& out,student& s)
{
   out<<s.name<<"\n"<<s.rollno<<"\n"<<s.marks<<endl;
   return out;
}
bool student::operator<(student& s)
{
   return (rollno<s.rollno);
}
bool student::operator>(student& s)
{
   return (rollno>s.rollno);
}

int main()
{
        vector<student> elements;
        int i,n;
        student ele;
        cin>>n;
```

```
    for(i=0;i<n;i++)
    {
         cin>>ele;
         elements.push_back(ele);
    }
    bidirectional_selection_sort(elements);
    for(i=0;i<n;i++)
         cout<<elements[i];

}
```

Insertion Sort and Binary Search

One of the interpretation of insertion sort is that when we try to insert an element at position 'i', elements from position 0 to i-1 are in sorted order. So let's use binary search to find to find the position of insertion.

(i) Normal binary to search for an element in a sorted array is given, this code will return index of element if it is present in the array and returns -1 otherwise. Modify the code to return the position to insert the search element in the sorted array. That is for example, if the original array contains five elements 12, 3, 45, 33, 37 and if we are trying to insert 33 then at that point the array will look as 3, 12, 45, 33 and 37, binary search should return index 2 (human index 3)

(ii) Binary search function given here work with an integer array, whereas a vector is used in the main function, use appropraite function of the container such that binary search shall be used for the process

(iii) Fill in the blanks appropraitely so that binary search will identify the position to insert the jth element to the left part of the array

Input Format

First line contains the number of elements, n

Next 'n' lines contain the elements in the array

Output Format

}

Print the elements of the array in the ascending order

```
#include<iostream>

#include<vector>

using namespace std;

int binary_search_pos(int *elements, int low, int high, int
s)

{

     int mid;

     mid = (low+high)/2;

    if(low>high)

    {

        return low;

    }

    if (elements[mid]<s)

    {

        return binary_search_pos(elements,mid+1,high,s);

    }

    else

    {

        return binary_search_pos(elements,low,mid-1,s);

    }

}
```

```
int main()

{

   vector<int> elements(20,0);

   int key,i,j,n,pos;

   cin>>n;

   for(i=0;i<n;i++)

   {

       cin>>elements[i];

   }

   int* ele=&elements[0];

   for(j=1;j<n;j++)

   {

       int* ele=&elements[0];

       key = elements[j];

       pos = binary_search_pos(ele,0,j-1,key);

       for(i=j;i>=pos;i--)

       {

           elements[i] = elements[i-1];

       }

       elements[pos]= key;

    }

for(i=0;i<n;i++)
```

```
    {
          cout<<elements[i]<<" ";
    }
}
```