CSCI 36200

data structures

Gavriil Tsechpenakis
gtsechpe@indiana.edu

books

Mark Allen Weiss, Data structures and algorithm analysis in C++, 4th edition, Pearson, Addison Wesley

T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein,
Introduction to Algorithms, MIT Press, Third Edition, 2009

content

Principles, fundamentals, advanced topics in data structures and algorithms

prereqs

CSCI 230/340, C++

- **Grading**

homework (hw): 40%

midterm exam: 20%

final exam: 40%

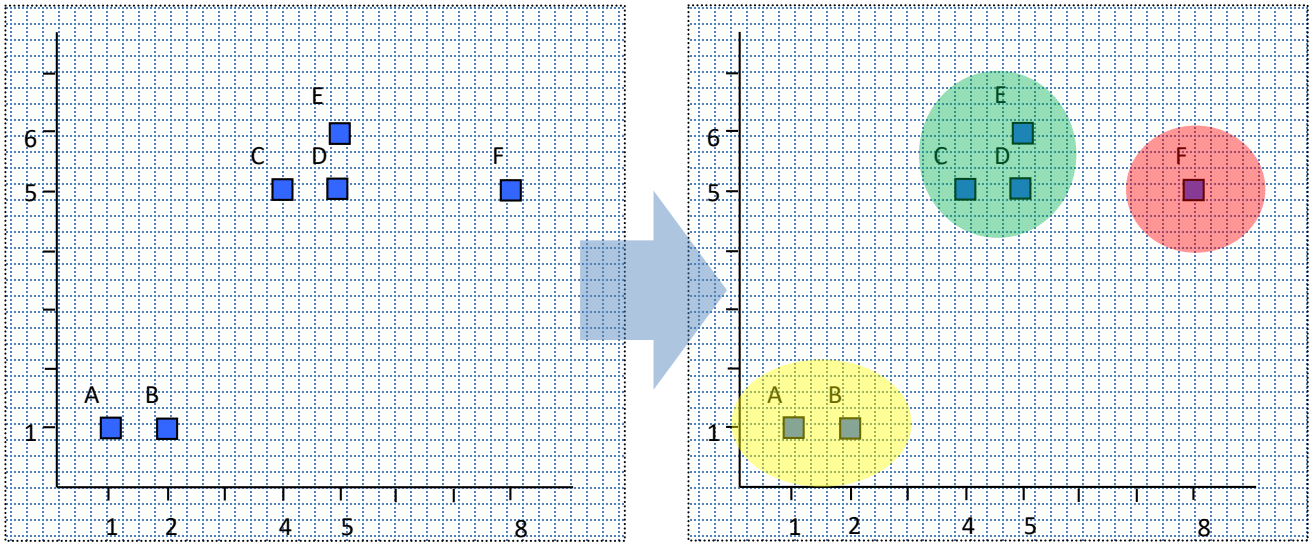
- **Policy**

  Academic integrity and sexual misconduct: please see the Syllabus section on Oncourse site, follow the links, and read carefully the related websites.

“algorithmic” thinking

k-means clustering

- One of the most known, simple, and relatively effective algorithms for clustering



- Numbers of clusters must be chosen in advance
- Strengths
  1. Data can flexibly change clusters during the process
  2. Always converges to a local optimum
  3. Quite fast for most applications
- Weaknesses
  - Global optimum solution not guaranteed

k-means (cnt'd)

1. Set k for number of output clusters

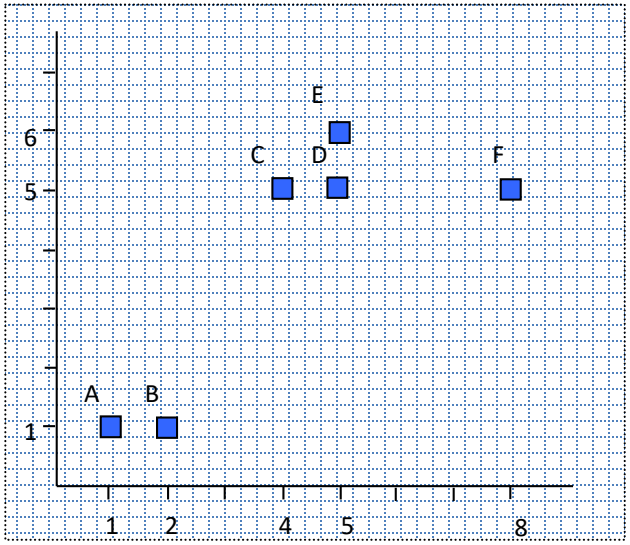
1. Randomly choose k points as centroids of the clusters

2. Pass through all datapoints and "assign" to each cluster its closest points – distance from the corresponding centroid

3. Re-calculate the centroid positions, by averaging the positions of the assigned points

4. Repeat 3-4 until convergence (points stop changing clusters)

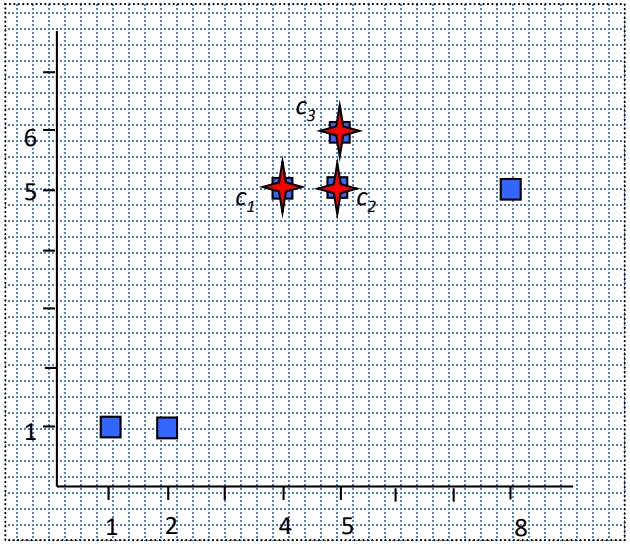
k-means (cnt'd)



Dataset:

X: a set of *N* data points (can be numbers, vectors, sets of vectors etc.)

N = 6 (dataset size)



Clusters

Random initial centroids

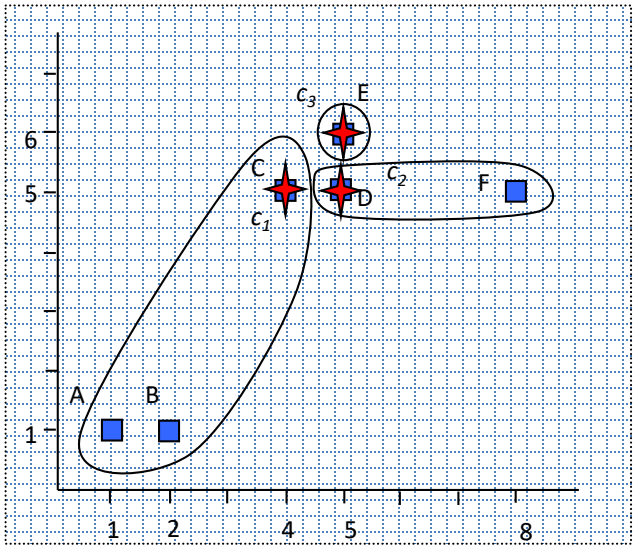
Initial codebook:

c_1 = C, c_2 = D, c_3 = E

C_i: initialized *k* cluster centroids

k = 3

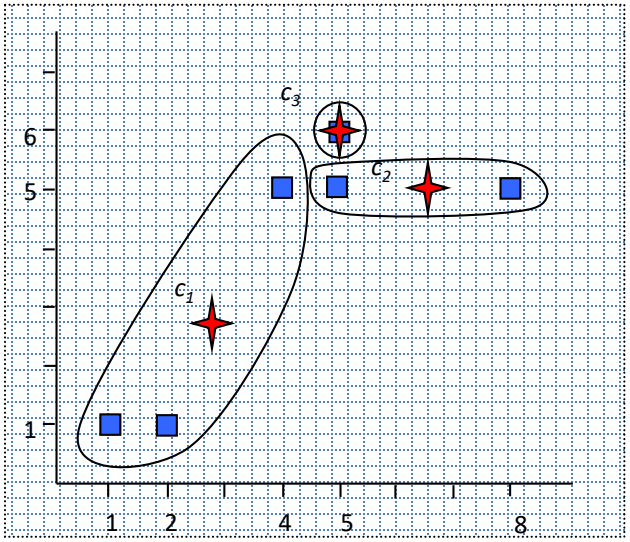
k-means (cnt'd)



Generate optimal partitions

Distance matrix (Euclidean distance)

	A	B	C	D	E	F
c_1	5	4,5	0	1	1,4	4
c_2	5,7	5	1	0	1	3
c_3	6,4	5,8	1,4	1	0	3,2



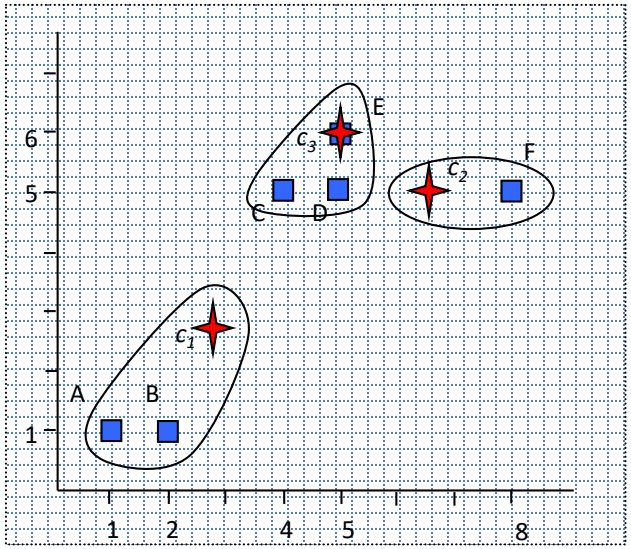
Generate optimal centroids

$$c_1 = \left(\frac{1+2+4}{3}, \frac{1+1+5}{3}\right) = (2.3, 2.3)$$

$$c_2 = \left(\frac{5+8}{2}, \frac{5+5}{2}\right) = (6.5, 5)$$

$$c_3 = (5, 6)$$

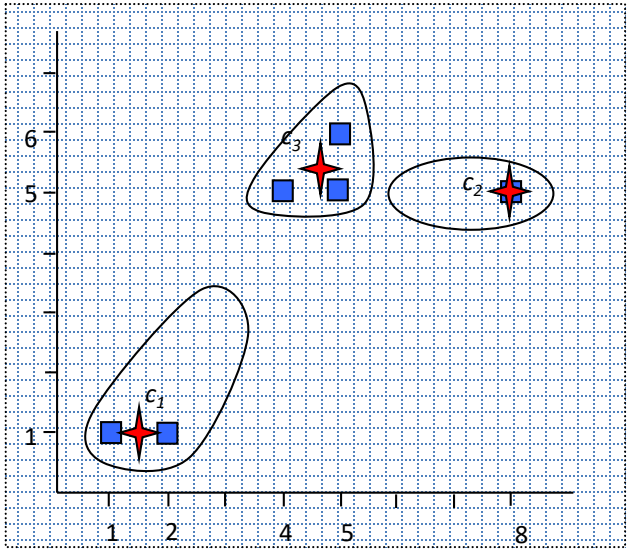
k-means (cnt'd)



Generate optimal partitions

Distance matrix (Euclidean distance)

	A	B	C	D	E	F
c_1	***1,9***	***1,4***	3,1	3,8	4,5	6,3
c_2	6,8	6	2,5	1,5	1,8	***1,5***
c_3	6,4	5,8	***1,4***	***1***	***0***	3,2



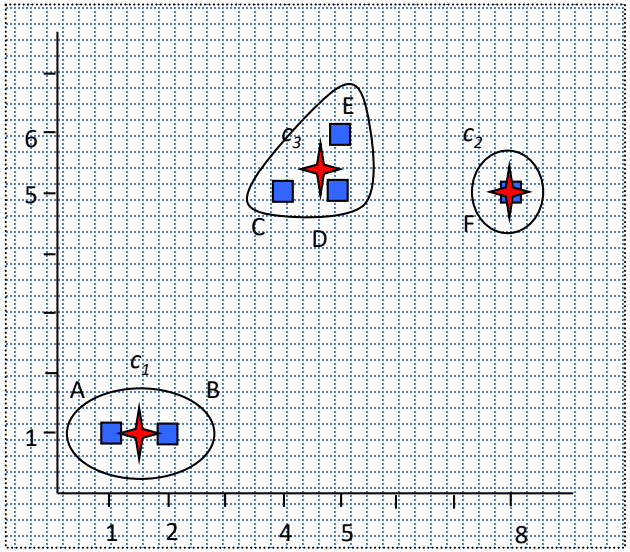
Generate optimal centroids

$$c_1 = \left(\frac{1+2}{2}, \frac{1+1}{2}\right) = (1.5,1)$$

$$c_2 = (8,5)$$

$$c_3 = \left(\frac{4+5+5}{3}, \frac{5+5+6}{3}\right) = (4.7,5.3)$$

k-means (cnt'd)



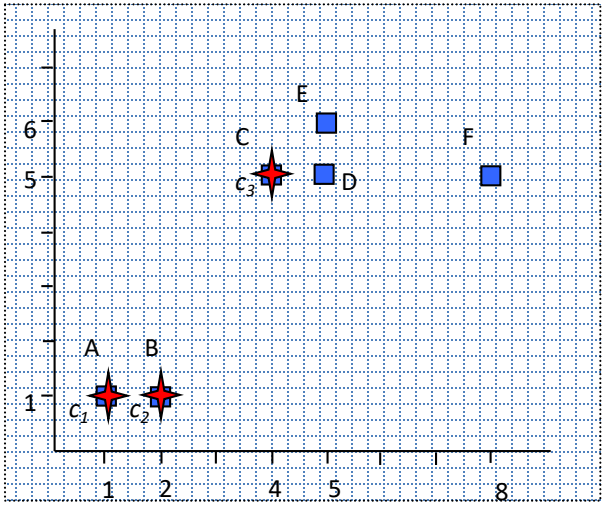
Generate optimal partitions

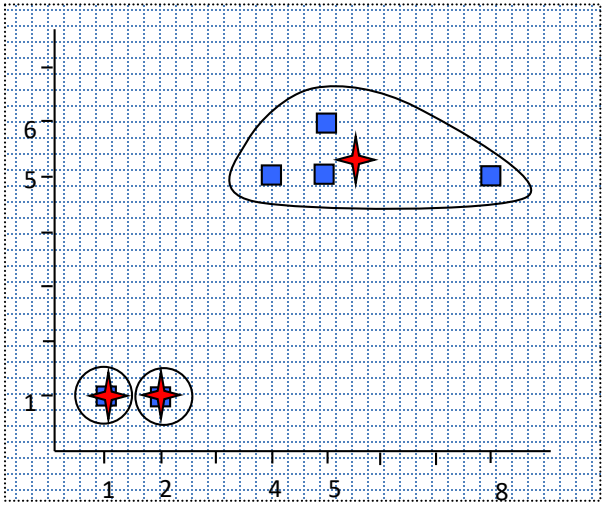
Distance matrix (Euclidean distance)

	A	B	C	D	E	F
c_1	***0,5***	***0,5***	4,7	5,3	6,1	7,6
c_2	8,1	7,2	4	3	3,2	***0***
c_3	5,7	5,1	***0,7***	***0,5***	***0,7***	3,3

No points move – stop

counter-example





Initial codebook:

$c_1 = A$, $c_2 = B$, $c_3 = C$

What is an alrorithm?

What is an alrorithm?

- Computational procedure that takes an INPUT and produces an OUTPUT
- A tool for solving a problem
- A simple example: sort the numbers $<a_1, a_2, a_3,..., a_{1573}>$
- What's wrong with this problem statement?
- A solution should be SCALABLE: with the same tool you should be able to sort any other set of numbers
- A correct algorithm terminates and produces correct output for all valid input instances
- How well does an algorithm scale up? (or down?)

Sorting

Given a set of n numbers, A[1:n], obtain a permutation of the numbers such that A(i) ≤ A(j), for all (i, j): i ≤ j

Searching

Given a set of n numbers A[1:n] and a number x, return an index j, such that A[j] = x or NIL if x is not found in A

Shortest path

Given a graph G(V,E) and a vertex s in V, find the shortest path from s to any vertex u in V\s

Why know algorithms?

- Learning analytical skills: how to solve a problem in a systematic, methodological way
- Analyze difficulty level of a problem
- How to find and utilize existing tools, software, math, data to solve a problem
- Algorithms: the spine of CS

…in and outside CS

Algorithm knowledge is necessary for many other research domains within (and outside) the field of computer science

- Graph Algorithms (computer networks)
- Spatial algorithms (computational geometry, computer graphics, computer vision and robotics)
- Resource optimization (in management, finance, manufacturing, marketing)
- Data manipulation (database design, information retrieval)
- String Algorithms (in bioinformatics)
- ….

data

- What do data typically look like? set of objects, a sequence of objects, or a group of objects that may have pair-wise relation, a collection of set-of-objects, or a set of key-value pairs

- Data can be dynamic, for example,
  - in a set of objects, the set can grow or shrink;
  - in a sequence of objects, the order of objects can change based on some priority value, or
  - Two sets in a set-of-objects can merge

- To obtain a desired output from an input data, the data objects may require to be accessed in a defined order, for example,
  - From a set of data objects, we may want to only access the newest data object or the oldest data object
  - from a sequence of data objects, we may want to access the objects by following the sequence, or randomly, or based on the priority assigned on each object

recursion

- Defining a function in terms of itself
  - Base case
  - Making progress through recursion
  - Reduce all cases towards the base case

```
void f()
{
... f() ...
}
```

- Example

```
for(int i=0;  i<10; i++)
{
    std::cout << "The number is: " << i << std::endl;
}
```

```
#include <iostream>
using namespace std;

void numberFunction(int i)
{
    cout << "The number is: “<< i << endl;
}

int main()
{
for(int i=0; i<10; i++)
     {
         numberFunction(i);
     }
return 0;
}
```

NO

```
#include <iostream>
using namespace std;

void numberFunction(int i)
{
    cout << "The number is: " << i << endl;
    i++;
    if(i<10)
     {
        numberFunction(i);
     }
}

int main()
{
int i = 0;
numberFunction(i);
return 0;
}
```

YES

recursion (cnt’d)

does it make the code run faster?
NO

does it use less memory?
NO

can it make the code more complicated?
YES

is it easy to implement it wrong?
YES

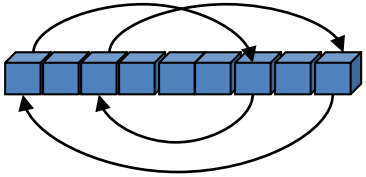
then why recursion?
sometimes it makes the code simpler (see sorting, e.g., Quicksort, later)

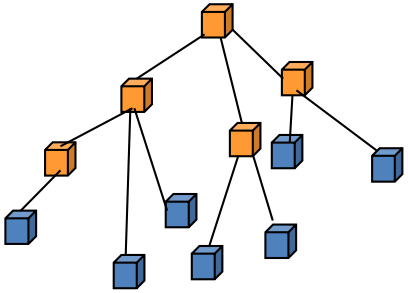
What is a data structure?

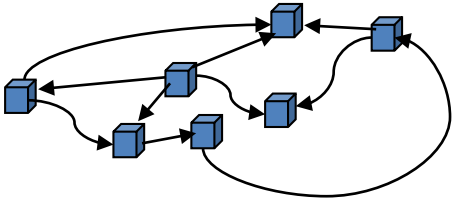
What is a data structure?

- Objects that store data so that they can be accessed and modified easily
- For each kind of data, there are well known data structure
  - Set of objects (Array or linklist, depending on the data access requirements)
  - Set of key-value pairs (hash table)
  - Set of objects with pair-wise relations (graph)
  - Sequence (sorted array, binary search tree)
  - Partial-Sequence (Heap, priority query)
- Study of data structures goes along with the study of algorithm, because
  - sometimes designing efficient algorithm becomes easy when we can obtain the most appropriate data structure for that task
  - For most of the data structures, the access/modification cost is well-known
- Most of the modern programming language (C++, Java) have built-in data structure objects

…in a nutshell







Algorithms: I/O =>

=> I/O = data =>

=> data = numbers, sequences, matrices, characters, strings, sets, “mixed” sets (see numerical vs categorical), independent / correlated, static / dynamic =>

=> Structures [arrays, tables, trees, priority queries etc) =>

=> storage, processing, access =>

=> Algorithms: (1) depending on algorithm, build structure; (2) depending on data structure, build algorithm

Example

Given a set of n numbers, A[1:n],
obtain a permutation of the numbers
such that $A(i) \leq A(j)$, for all (i, j): $i \leq j$

- Data structure: a sequence of numbers stored in an **array**
- The numbers are also called **keys**
- The way of solving this problem is expressed as an algorithm
- think => pseudocode => code

insertion sort

a good way of sorting short sequences

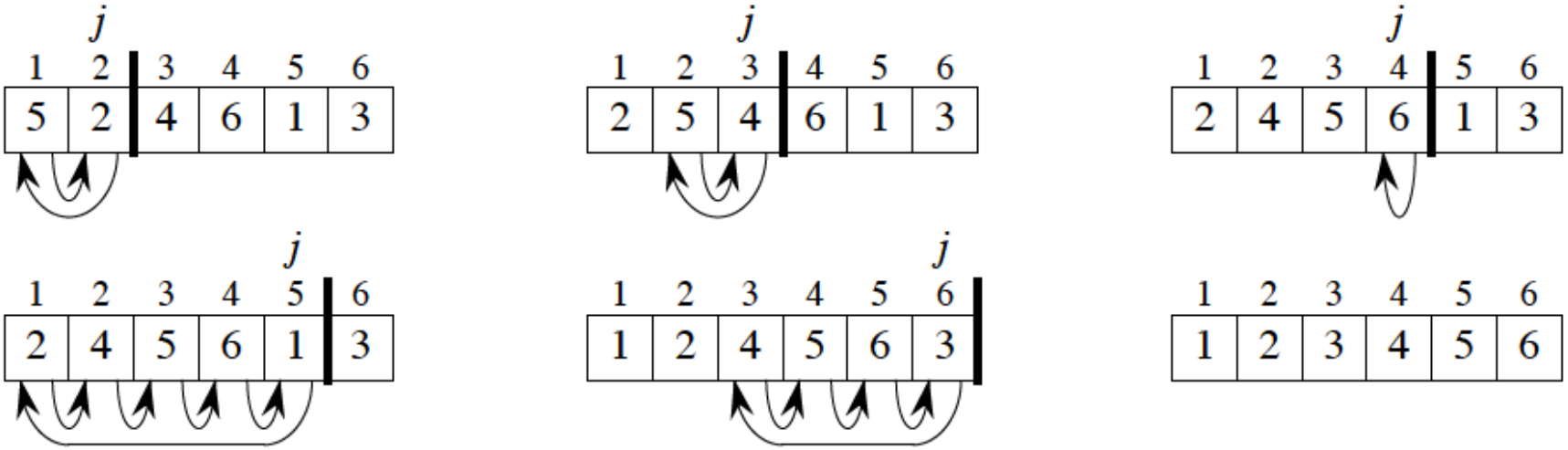
sorting “in-place” = within the same array

Principle: playing cards

Start with empty hand and all cards face-down on the table

Pick one card at a time and insert it into the correct position in hand

To find the correct position, compare it with each of the cards already in hand, from right to left



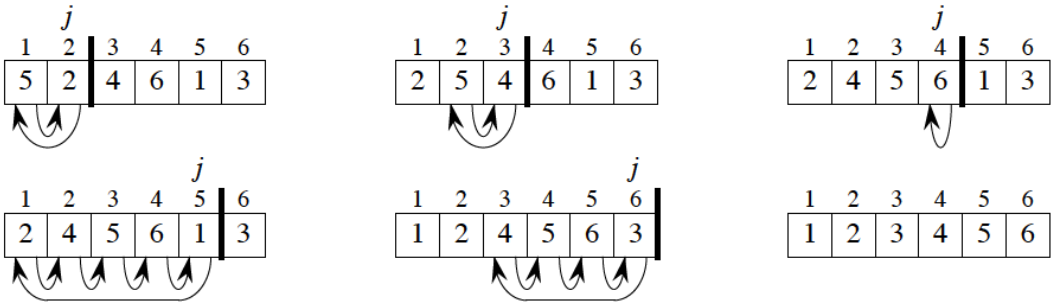
INSERTION-SORT(A, n)

for $j = 2$ **to** n
  $key = A[j]$
  // Insert $A[j]$ into the sorted sequence $A[1 \,..\, j-1]$
  $i = j - 1$
  while $i > 0$ and $A[i] > key$
    $A[i+1] = A[i]$
    $i = i - 1$
  $A[i+1] = key$

```
Enter length of array (max 100): 5

Enter the array keys
element 1 : 3
element 2 : 1
element 3 : 6
element 4 : 4
element 5 : 8

Sorted in ascending order

element 1 : 1
element 2 : 3
element 3 : 4
element 4 : 6
element 5 : 8
```

Target Output

```
#include <iostream>

void Insertion_sort(int AR[],int n)
{
    int temp,j;
    AR[0] = INT_MIN;
    for (int i=1; i<=n; i++)
    {
        temp = AR[i];
        j = i-1;
        while ( temp < AR[j])
        {
            AR[j+1] = AR[j];
            j--;
        }
        AR[j+1] = temp;
    }
}

int main()
{
    int arr[100];
    int size;
    std::cout<<" Enter length of array (max 100): ";
    std::cin>>size;
    std::cout<<"\n Enter the array keys \n ";

    for (int i=1; i<=size; i++)
    {
        std::cout<<"element "<<i<<" : ";
        std::cin>>arr[i];
    }
    Insertion_sort(arr,size);

    std::cout<<" \n Sorted in ascending order \n ";
    for (int i=1; i<=size; i++)
    {
        std::cout<<" \n element "<<i<<" : "<<arr[i];
    }

    return 0;
}
```



correctness

Use a **loop invariant** (instance) of the algorithm to show the correctness:

> Insertion-sort: in each iteration (for a given j), the subset A[1,…,j-1] consists of numbers originally in A[1,…,j-1], but sorted in accending order

We must show three factors:

Initialization: it is true prior to the first iteration

Maintenance: if it is true before an iteration, it remains true before next iteration

Termination: when loop terminates, we get a property that holds

insertion sort

```
INSERTION-SORT(A, n)
for j = 2 to n
    key = A[j]
    // Insert A[j] into the sorted sequence A[1 . . j − 1]
    i = j − 1
    while i > 0 and A[i] > key
        A[i + 1] = A[i]
        i = i − 1
    A[i + 1] = key
```

Initialization: *it is true prior to the first iteration*

Just before the first iteration, j=2 => A[1,…,j-1] is a single element array A[1], which is sorted

Maintenance: *if it is true before an iteration, it remains true before next iteration*

We should also prove a loop invariant for the *while* loop. In such simple cases, it is sufficient to describe what it does, i.e., moving elements by one position to right until proper position for key is found.

Termination: *when loop terminates, we get a property that holds*

The outer loop ends when j > n, i.e., when j=n+1 => j-1 = n. Replacing j-1 with n in the loop invariant, the subarray A[1,…n] consists of all elements originally in A[1,…,n] but in sorted order.