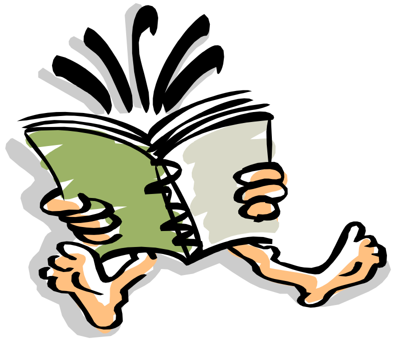
Jiangxi University of Science and Technology

Extra lectures on Sorting /Arrays

Lecture0605 Algorithms and method on Sorting /Arrays

THE
C
PROGRAMMING
LANGUAGE



Sorting

- To arrange a set of items in sequence.
  - It is estimated that 25~50% of all computing power is used for sorting activities.

- Possible reasons:
  - Many applications require sorting;
  - Many applications perform sorting when they don't have to
  - Many applications use inefficient sorting algorithms.

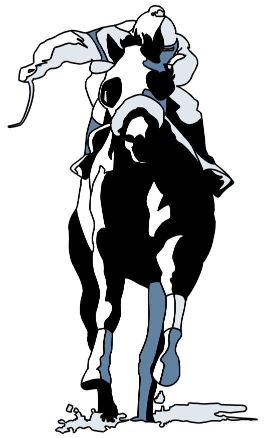
Why Study Sorting Algorithms?

- There are a variety of situations that we can encounter
  - Do we have randomly ordered keys?
  - Are all keys distinct?
  - How large is the set of keys to be ordered?
  - Need guaranteed performance?

- Various algorithms are better suited to some of these situations

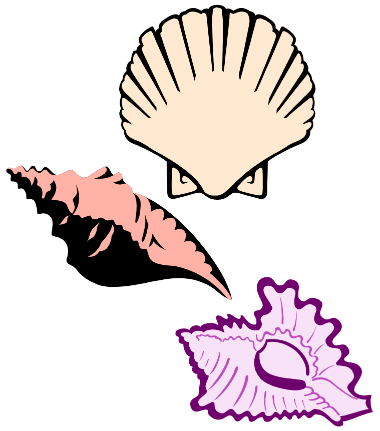
Sorting Applications

- To prepare a list of student ID, names, and scores in a table (sorted by ID or name) for easy checking.
- To prepare a list of scores before letter grade assignment.
- To produce a list of horses after a race (sorted by the finishing times) for payoff calculation.
- To prepare an originally unsorted array for ordered binary searching.



Some Sorting Methods

- Selection sort
- Bubble sort
- Shell sort (a simple but faster sorting method than above; see p.331 of *Numerical Recipes in C*, 2nd ed., by William H. Press *et al*, Cambridge University Press, 1992)
- Quick sort (a very efficient sorting method for most applications; p.332-336, *ibid*.)



- Searching and sorting through arrays is one of the most labor-intensive tasks.
- There are two different approaches to searching through arrays:

  linear or sequential search, and binary search

Basic Sorting Algorithms

- The Sort function sorts the elements in the range in a particular order.
- The different types of sorting methods are Bubble Sort, Selection Sort, Merge Sort and Quick Sort.
- Bubble Sort repeatedly sorts the adjacent elements if they are in wrong order.
- The Selection Sort finds the smallest element in the array and exchanges it with the element in the first position, it then finds the second smallest element and exchanges it with the element in the second position and continues this process until the entire list is sorted.
- The Merge Sort follows the Divide and Conquer rule where in it divides the input array into two halves, sorts the two halves and merges the two sorted halves.
- The Quick Sort selects an element as a pivot and partitions the given array around the pivot.
- The following C programs show the implementation of all the sorting methods mentioned above

Searching

- Linear Search
  - Simplest but Costly
- Binary Search
  - Efficient but assumes the input to be sorted

Bubble sort

- Compare each element (except the last one) with its neighbor to the right
  - If they are out of order, swap them
  - This puts the largest element at the very end
  - The last element is now in the correct and final place
- Compare each element (except the last two) with its neighbor to the right
  - If they are out of order, swap them
  - This puts the second largest element next to last
  - The last two elements are now in their correct and final places
- Compare each element (except the last three) with its neighbor to the right
  - Continue as above until you have no unsorted elements on the left

Bubble Sort

- Bubble Sort is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are in wrong order.

Example:
First Pass:
(**5 1** 4 2 8) –> (**1 5** 4 2 8), Here, algorithm compares the first two elements, and swaps since 5 > 1.
(1 **5 4** 2 8) –> (1 **4 5** 2 8), Swap since 5 > 4
(1 4 **5 2** 8) –> (1 4 **2 5** 8), Swap since 5 > 2
(1 4 2 **5 8**) –> (1 4 2 **5 8**), Now, since these elements are already in order (8 > 5), algorithm does not swap them.

Second Pass:
(1 4 2 5 8) –> (1 4 2 5 8)
(1 4 2 5 8) –> (1 2 4 5 8), Swap since 4 > 2
(1 2 4 5 8) –> (1 2 4 5 8)
(1 2 4 5 8) –> (1 2 4 5 8)
Now, the array is already sorted, but our algorithm does not know if it is completed. The algorithm needs one whole pass without any swap to know it is sorted.

Third Pass:
(1 2 4 5 8) –> (1 2 4 5 8)
(1 2 4 5 8) –> (1 2 4 5 8)
(1 2 4 5 8) –> (1 2 4 5 8)
(1 2 4 5 8) –> (1 2 4 5 8)

bubble sort

- In a **bubble sort**, you swap neighbours; the larger items drop down while the smaller ones bubble up, in n-1 passes through the array
- C program for bubble sort: C programming code for bubble sort to sort numbers or arrange them in ascending order.

```
/* Bubble sort code */
 #include <stdio.h>
 int main(){
  int array[100], n, c, d, swap;
   printf("Enter number of elements\n");
  scanf("%d", &n);
   printf("Enter %d integers\n", n);
   for (c = 0; c < n; c++)
    scanf("%d", &array[c]);
  for (c = 0 ; c < n - 1; c++)
  {
    for (d = 0 ; d < n - c - 1; d++)
    {
if (array[d] > array[d+1]) /* For decreasing order use < */
      {
        swap       = array[d];
        array[d]   = array[d+1];
        array[d+1] = swap;
      }
    }
  }
  printf("Sorted list in ascending order:\n");
  for (c = 0; c < n; c++)
     printf("%d\n", array[c]);
   return 0;
}
```

Selection sort algorithm

C program for selection sort to sort numbers.

This code implements selection sort algorithm to arrange numbers of an array in ascending order.

```c
#include <stdio.h>
int main(){
  int array[100], n, c, d, position, swap;
  printf("Enter number of elements\n");
  scanf("%d", &n);
  printf("Enter %d integers\n", n);
   for (c = 0; c < n; c++)
    scanf("%d", &array[c]);
  for (c = 0; c < (n - 1); c++)   {
     position = c;

     for (d = c + 1; d < n; d++)
     {
       if (array[position] > array[d])
         position = d;
     }
     if (position != c)
     {
       swap = array[c];
       array[c] = array[position];
       array[position] = swap;
     }
   }

   printf("Sorted list in ascending order:\n");
   for (c = 0; c < n; c++)
     printf("%d\n", array[c]);
    return 0;
}
```

Insertion Sort

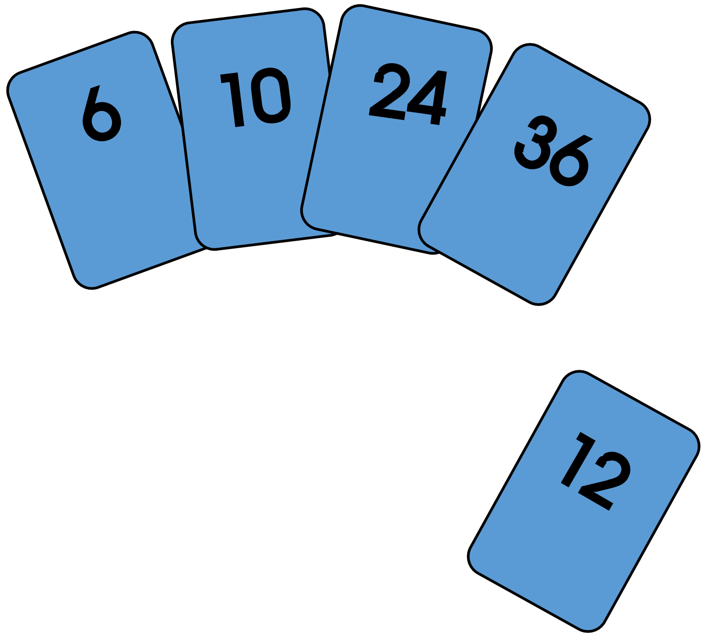
- Idea: like sorting a hand of playing cards
  - Start with an empty left hand and the cards facing down on the table.
  - Remove one card at a time from the table, and insert it into the correct position in the left hand
  - compare it with each of the cards already in the hand, from right to left
  - The cards held in the left hand are sorted
    - these cards were originally the top cards of the pile on the table

- Advantages
  - Good running time for “almost sorted” arrays Θ(n)
- Disadvantages
  - $\Theta(n^2)$ running time in worst and average case
  - $\approx n^2/2$ comparisons and exchanges

Insertion Sort

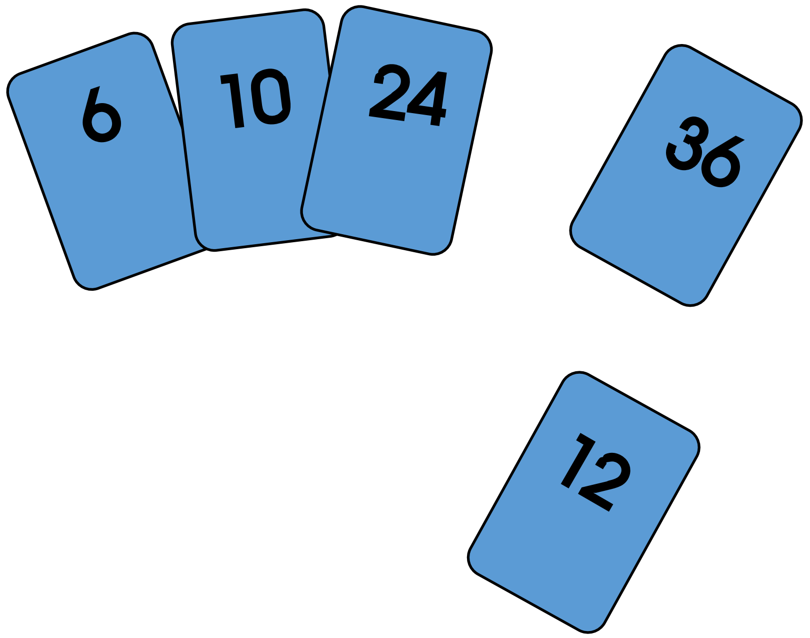
- In an **insertion sort**, you start with one item, take a new item and sort the two items relative to each other, then take a new item and sort the three items relative to each other (swapping the new item with consecutive values until it is no longer lower, and thus inserting it in that position), and so on. It is like sorting a deck of cards with your

Insertion Sort

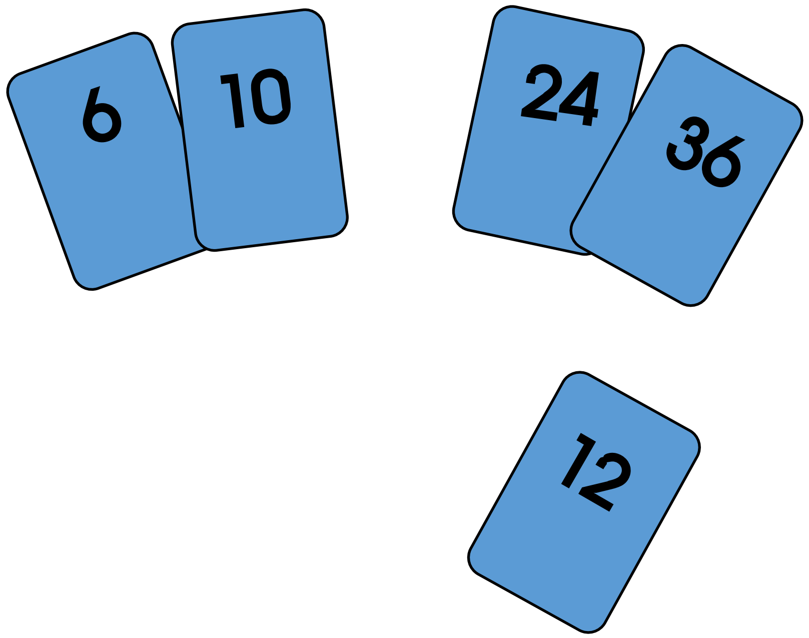


To insert 12, we need to make room for it by moving first 36 and then 24.

Insertion Sort



Insertion Sort



Insertion sort in C

- C program for insertion sort to sort numbers. This code implements insertion sort algorithm to arrange numbers of an array in ascending order. With a little modification, it will arrange numbers in descending order. Best case complexity of insertion sort is O(n), average and the worst case complexity is O(n2).

```c
#include <stdio.h>
int main(){
 int n, array[1000], c, d, t;
  printf("Enter number of elements\n");
 scanf("%d", &n);
  printf("Enter %d integers\n", n);
  for (c = 0; c < n; c++)
   scanf("%d", &array[c]);
  for (c = 1 ; c <= n - 1; c++) {
    d = c;
     while ( d > 0 && array[d-1] > array[d]) {
      t          = array[d];
      array[d]   = array[d-1];
      array[d-1] = t;
       d--;
    }
  }
   printf("Sorted list in ascending order:\n");
   for (c = 0; c <= n - 1; c++) {
    printf("%d\n", array[c]);
  }
   return 0;
}
```

Linear search

The following code implements linear search (Searching algorithm) which is used to find whether a given number is present in an array and if it is present then at what location it occurs. It is also known as sequential search.

It is straightforward and works as follows: We keep on comparing each element with the element to search until it is found or the list ends. Linear search in C language for multiple occurrences and using function

```c
#include <stdio.h>
int main(){
  int array[100], search, c, n;
  printf("Enter number of elements in array\n");
  scanf("%d", &n);
  printf("Enter %d integer(s)\n", n);
   for (c = 0; c < n; c++)
    scanf("%d", &array[c]);

   printf("Enter a number to search\n");
   scanf("%d", &search);
   for (c = 0; c < n; c++)  {
     if (array[c] == search)    /* If required element is found */
     {
       printf("%d is present at location %d.\n", search, c+1);
       break;
     }
   }
   if (c == n)
     printf("%d isn't present in the array.\n", search);
    return 0;
}
```

binary search

This code implements binary search in C language. It can only be used for sorted arrays, but it's fast as compared to linear search.

If you wish to use binary search on an array which isn't sorted, then you must sort it using some sorting technique say merge sort and then use the binary search algorithm to find the desired element in the list.

If the element to be searched is found then its position is printed. The code below assumes that the input numbers are in *ascending* order.

```
#include <stdio.h>
int main(){
   int c, first, last, middle, n, search, array[100];
    printf("Enter number of elements\n");
   scanf("%d",&n);
   printf("Enter %d integers\n", n);
    for (c = 0; c < n; c++)
      scanf("%d",&array[c]);
    printf("Enter value to find\n");
   scanf("%d", &search);
    first = 0;
   last = n - 1;
 middle = (first+last)/2;
    while (first <= last) {
      if (array[middle] < search)
         first = middle + 1;
      else if (array[middle] == search) {
printf("%d found at location %d.\n", search, middle+1);
         break;       }
      else
         last = middle - 1;
      middle = (first + last)/2;    }
   if (first > last)
printf("Not found! %d isn't present in the list.\n", search);
    return 0;
}
```

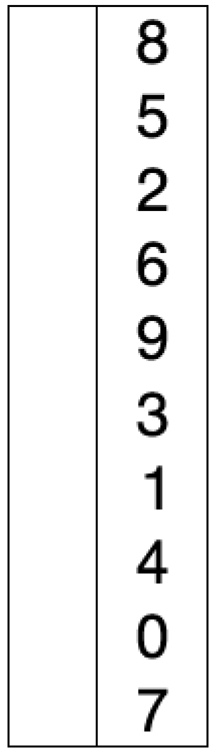
江西理工大学
JIANGXI UNIVERSITY OF SCIENCE AND TECHNOLOGY

Small comparsion

Bubble Sort

•Bubble Sort- A sorting algorithm which compares one element to its next element and if requires it swaps like a bubble.

Selection Sort



•Selection Sort - A sorting algorithm which selects a position in the elements and compares it to the rest of the positions one by one.

Insertion sort

•Insertion sort
•One element from the array is selected and is compared to the one side of the array and inserted to the proper position while shifting the rest of the elements accordingly.

Other Sort algo

- **Selection sort**
- Bubble sort
- **Insertion sort**
- Shell sort
- **Merge sort**
- Heapsort
- **Quicksort**

•Quick Sort - A sorting algorithm which divides the elements into two subsets and again sorts recursively.
- Heap Sort - A sorting algorithm which is a comparison based sorting technique based on Binary Heap data structure.
- Merge sort - A sorting algorithm which divides the elements to subgroups and then merges back to make a sorted.
- Radix Sort - A sorting algorithm used for numbers. It sorts the elements by rank of the individual digits.

Reference