

SORTING

- ❑ Computer science theory says that most of computer science problems involves sorting.
- ❑ **Sorting:** “Arranging components of a list into order”
- ❑ There exists plenty of algorithms used to sort arrays or lists, important is “how effective” they are.
- ❑ Best known algorithms works with $O(n \log_2 n)$ complexity.
- ❑ Most simple ones are usually $O(n^2)$.
- ❑ Try to write down list random list of numbers, and sort them.
- ❑ Can you figure out algorithm you use? If you can describe your intuitive way, you can write a program that implements it.

SELECTION SORT (1)

```
// Sort a list into ascending order
#include <iostream.h>
#include <fstream.h>
#include <iomanip.h>

// Maximum list size
const int MAXSIZE = 100;

// Sort an array
void selection(int list[ ], int length)
{ // position of min number
  int minLoc;

  // top of sorted part of list
  int top;

  // swap numbers in list
  int temp;

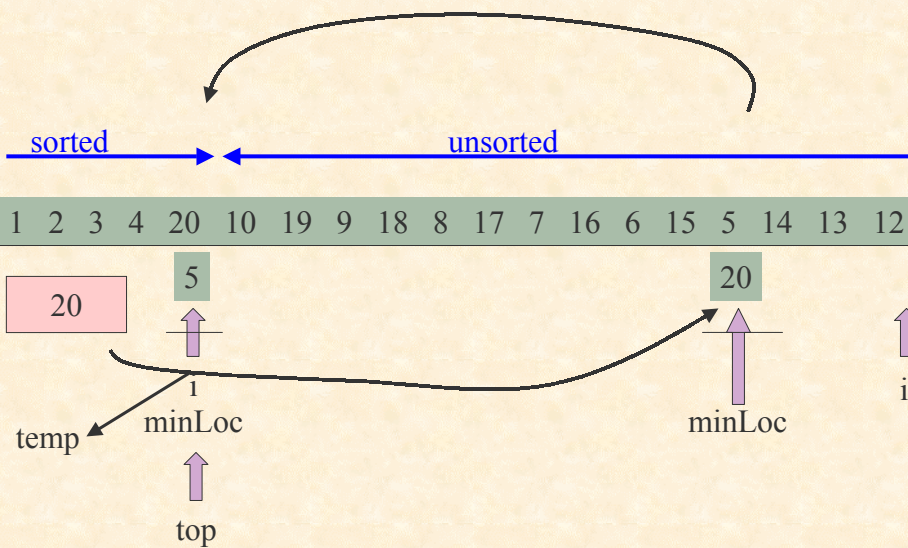
  // loop variable
  int i;
```

- ❑ Given a list of numbers, we want to sort the list into ascending order.
 - there are many application programs that need to sort
- ❑ We will do the sorting in a separate function that can easily be moved to any program that needs to sort a list.
- ❑ We will use the selection sort algorithm.
- ❑ The program sorts any list of numbers up to size MAXSIZE.
 - currently MAXSIZE is 100

SELECTION SORT ALGORITHM

- ❑ At any point during the algorithm, the first part of the list is sorted and the last part is unsorted.
 - initially the first part is empty and the last part is the entire list
- ❑ Search the unsorted part of the list for the smallest number.
- ❑ Swap this smallest number to the end of the sorted part of the list.
- ❑ Repeat this process until the list is sorted.

SELECTION SORT ALGORITHM



SELECTION SORT (2)

```
// loop to sort list
for (top=0; top<=length-2; top=top+1) {
    // find smallest number in unsorted
    // part of the list
    minLoc = top;
    for (i=top+1; i<=length-1; i=i+1) {
        if (list[i] < list[minLoc]) {
            minLoc = i;
        }
    };
    // swap smallest number into its
    // final position in sorted list
    temp = list[top];
    list[top] = list[minLoc];
    list[minLoc] = temp;
};
}
```

- ❑ In a list of length n , once $n-1$ numbers are in their final positions, the list is sorted.
 - by default, the remaining number must be in the correct position (where else?)
- ❑ The program swaps two elements of the array.
 - swapping requires an extra variable (named `temp` in this case) to temporarily hold the value of one of the variables being swapped

Programming 1

Lecture 13 -- 5

SELECTION SORT (3)

```
void main ()
{
    int list[MAXSIZE]; // list to sort
    int size; // size of list
    int i; // loop variable
    ifstream listfile; // file containing list

    // read the numbers into array list
    // and display them on the screen
    listfile.open("list.txt", ios::in);
    size = 0;
    listfile >> list[size];
    cout << "Original List:" << endl;
    while (!listfile.eof()) {
        cout << setw(4) << list[size];
        size = size + 1;
        listfile >> list[size];
    };
};
```

```
// sort the list
selection(list, size);

// display the sorted list
cout << endl << endl
    << "Sorted List:" << endl;
for (i=0; i<size; i=i+1) {
    cout << setw(4) << list[i];
};
cout << endl << endl;
}
```

- ❑ Read the list of numbers from a file named `list.txt`.
- ❑ Display the list before and after sorting.

Programming 1

Lecture 13 -- 6

PROGRAM RESULTS

```
"C:\WINNT\Profiles\All Users\Desktop\CS 1\Lectures\Lecture 25\sort\Debug\sort.exe"
Original List:
20 10 1 19 9 2 18 8 3 17 7 4 16 6 5 15 11 14 12 13

Sorted List:
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

Press any key to continue
```

PERFORMANCE OF SELECTION SORT

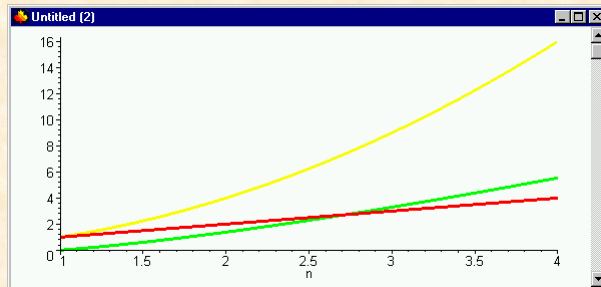
- ❑ The selection sort function has nested for loops as follows:

```
for (top=0; top<=length-2; top=top+1) {
    ...
    for (i=top+1; i<=length-1; i=i+1) {
        comparison with list element
    };
};
```

- ❑ The outer loop is executed $n-1$ times, where n is the length of the list.
- ❑ The inner loop executes: $n-1, n-2, n-3, \dots, 2, 1$ times on successive iterations of the outer loop, respectively.
 - this is an average of about $n/2$ times for large n
- ❑ Thus, the comparison inside the inner loop is made about $(n-1) \times (n/2) = 1/2(n^2 - n)$ times.
- ❑ Therefore, selection sort is an $O(n^2)$ algorithm.

WHAT DOES THIS MEAN?

- ❑ The most efficient sorting algorithms can be shown to be $O(n \log_2 n)$, where n is the size of the list to sort.



- ❑ As can be seen in the graph $O(n \log_2 n)$ is much better than $O(n^2)$ when n gets large.
- ❑ The more efficient sorting algorithms are more complex to implement, however (at least most of them).
 - some of them will be studied next semester in the Programming II course.

BUBBLE SORT

- ❑ Another one from “slow” sorting algorithms. However, one of the most simple to implement and understand.
- ❑ Basic idea:
 - Compare two successive elements of array.
 - If they are “in order” proceed, if not, swap them and proceed. When you reach end of array, start again from beginning, until all array is sorted.
- ❑ This algorithm uses two nested loops same way as previous selection sort.
- ❑ The name comes from effect algorithm has. Smallest values moves to one end of array like bubbles of air in water:-)

BUBBLE SORT PROGRAM

```
#include <iostream.h>
#include <fstream.h>
#include <iomanip.h>

const int MAXSIZE = 100;

void bubble_sort(int list[], int length)
{
    int temp; // Temporary variable
    int i,j;   // Will be used as counters in for loops, names i and j are
              // traditional
    for ( i = 0 ; i < length ; i++)
        for ( j = 0; j < length-1;j++){
            if ( list[j] > list[j+1]){
                temp = list[j];
                list[j] = list[j+1];
                list[j+1] = temp;
            }
        }
}
```

Programming I

Lecture 13 -- 11

BUBBLE SORT PROGRAM (CONT.)

```
void main ()
{
    int list[MAXSIZE]; // list to sort
    int size; // size of list
    int i; // loop variable
    fstream listfile; // file containing list

    listfile.open("list.txt", ios::in);

    if ( !listfile) return -1;

    size = 0;
    listfile >> list[size];
    cout << "Original List:" << endl;
    while (!listfile.eof()) {
        cout << setw(4) << list[size];
        size = size + 1;
        listfile >> list[size];
    }
};
```

Programming I

Lecture 13 -- 12

BUBBLE SORT PROGRAM (CONT.)

```
// sort the list
bubble_sort(list, size);

// display the sorted list
cout << endl << endl
    << "Sorted List:" << endl;
for (i=0; i<size; i=i+1) {
    cout << setw(4) << list[i];
};
cout << endl << endl;
}
```

- Can we make this program faster? Sure yes.
- One idea can be, DO NOT SORT SORTED LIST!!!
- How do we know that list is already sorted?
- It is sorted when there are no more members exchanging places

BUBBLE SORT PROGRAM – FASTER VERSION

```
void bubble_sort(int list[], int length)
{
    int temp,i,j;
    int counter = 0;        // Variable to count how many times does variable
                          // swaps
    for ( i = 0 ; i < length ; i++)
    {
        for ( j = 0; j < length-1;j++)
            if ( list[j] > list[j+1]){
                temp = list[j];
                list[j] = list[j+1];
                list[j+1] = temp;
                counter ++;
            }
        if (counter == 0 )
            break;
        counter = 0;
    }
}
```