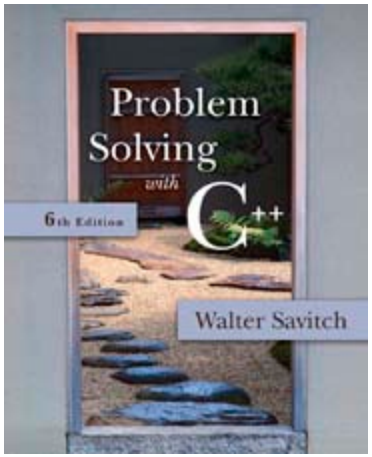
# APS105: Lecture 22

Wael Aboelsaadat

wael@cs.toronto.edu

http://ccnet3.utoronto.ca/20079/aps105h1f/

Acknowledgement: These slides are a modified version of the text book slides as supplied by Addison Wesley



# Binary Search

- A binary search can be used to search a sorted array to determine if it contains a specified value
  - The array indexes will be 0 through final_index
  - Because the array is sorted, we know a[0] <= a[1] <= a[2] <= … <= a[final_index]
  - If the item is in the list, we want to know where it is in the list

# Binary Search: Problem Definition

- The function will use two call-by-reference parameters to return the outcome of the search
  - One, called found, will be type bool. If the value is found, found will be set to true. If the value is found, the parameter, location, will be set to the index of the value
- A call-by-value parameter is used to pass the value to find
  - This parameter is named key

# Binary Search
# Problem Definition (cont.)

- Pre and Postconditions for the function:

```
//precondition:  a[0] through a[final_index] are
//                sorted in increasing order

//postcondition: if key is not in a[0] -
a[final_index]
//                found = = false; otherwise
//                found = = true
```

# Binary Search Algorithm Design

- Our algorithm is basically:
  - Look at the item in the middle
    - If it is the number we are looking for, we are done
    - If it is greater than the number we are looking for, look in the first half of the list, why?
    - If it is less than the number we are looking for, look in the second half of the list, why?

# Binary Search Algorithm Design (cont.)

- Here is an attempt at our algorithm:
  - found = false;
    mid = approx. midpoint between 0 and final_index;
    if (key == a[mid])
    {
    found = true;
    location = mid;
    }
    else if (key < a[mid])
    search a[0] through a[mid -1]
    else if (key > a[mid])
    search a[mid +1] through a[final_index];

**Display 14.5**

**Display 14.7**

# Binary Search: the code

**Display 14.8**

# Bubble sort…?

```
void bubbleSort(int arrNumbers[], int length)
{
  int i, j, temp;

  bool bContinue = true;

  while( bContinue == true )
  {
    bContinue=false;
    for(j = 0; j < (length-1); j++)
    {
      if(arrNumbers[j] > arrNumbers[j+1]) /* compare neighboring elements */
      {
        temp = arrNumbers[j];    /* swap array[j] and array[j+1] */
        arrNumbers[j] = arrNumbers[j+1];
        arrNumbers[j+1] = temp;
        bContinue = true;
      }
    } /*end for j*/

  }

}
```

Sorting Animation:

http://www.iste.uni-stuttgart.de/ps/Ploedereder/sorter/sortanimation2.html

# Selection Sort Algorithm

- One array is sufficient to do our sorting
  - Search for the smallest value in the array
  - Place this value in a[0], and place the value that was in a[0] in the location where the smallest was found
  - Starting at a[1], find the smallest remaining value swap it with the value currently in a[1]
  - Starting at a[2], continue the process until the array is sorted

**Display 7.11** **Display 7.12 (1-3)**

# Display 14.5



**Pseudocode for Binary Search**

```
int a[Some_Size_Value];
```

**Algorithm to search** `a[first]` **through** `a[last]`

```
//Precondition:

//a[first]<= a[first + 1] <= a[first + 2] <= ... <= a[last]
```

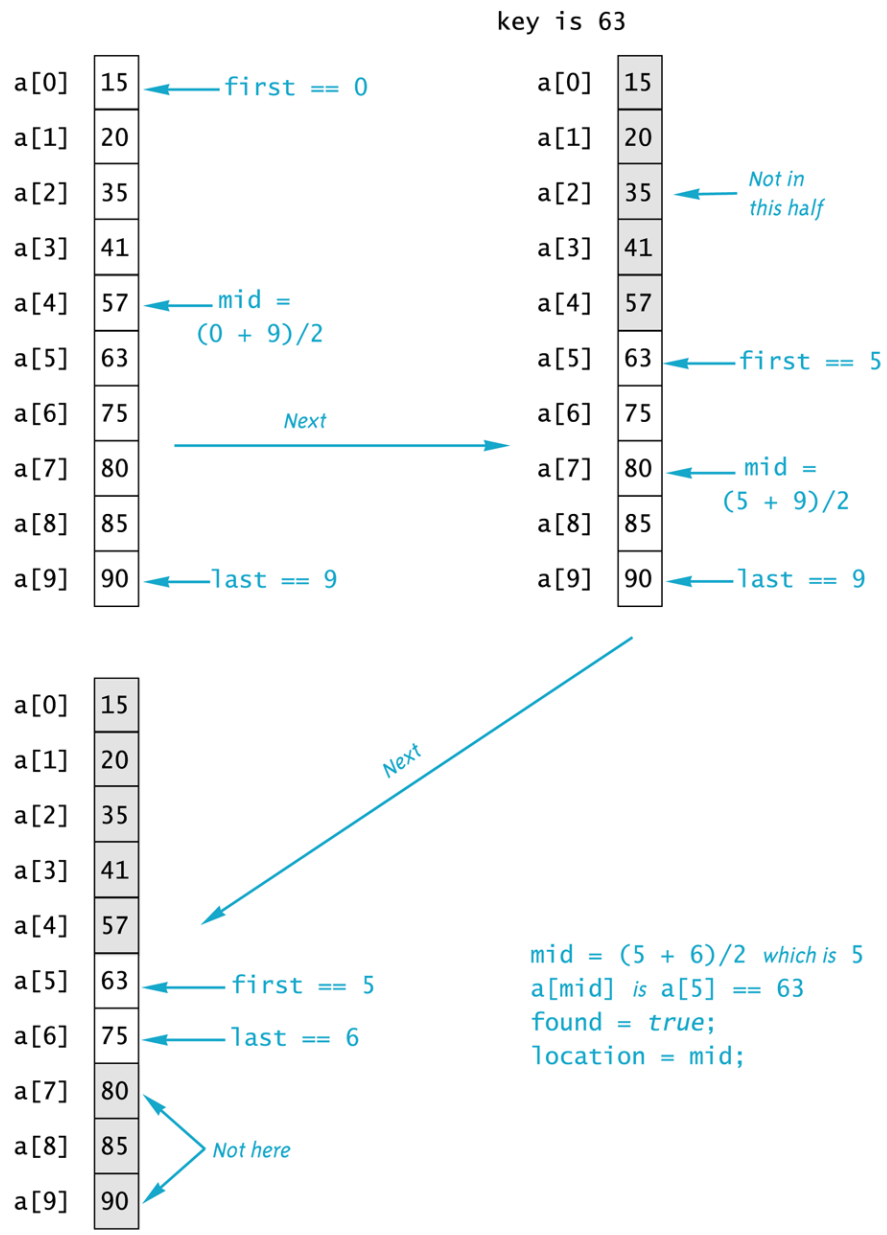
To locate the value key:

```
if (first > last) //A stopping case
    found = false;
else
{
    mid = approximate midpoint between first and last;
    if (key == a[mid]) //A stopping case
    {
        found = true;
        location = mid;
    }
    else if key < a[mid] //A case with recursion
        search a[first] through a[mid - 1];
    else if key > a[mid] //A case with recursion
        search a[mid + 1] through a[last];
 }
```

# Display 14.7

Back Next

**Execution of the Function search**



## Iterative Version of Binary Search

# Display 14.8


**Function Declaration**

```
void search(const int a[], int low_end, int high_end,
                           int key, bool& found, int& location);
//Precondition: a[low_end] through a[high_end] are sorted in increasing
//order.
//Postcondition: If key is not one of the values a[low_end] through
//a[high_end], then found == false; otherwise, a[location] == key and
//found == true.
```

**Function Definition**

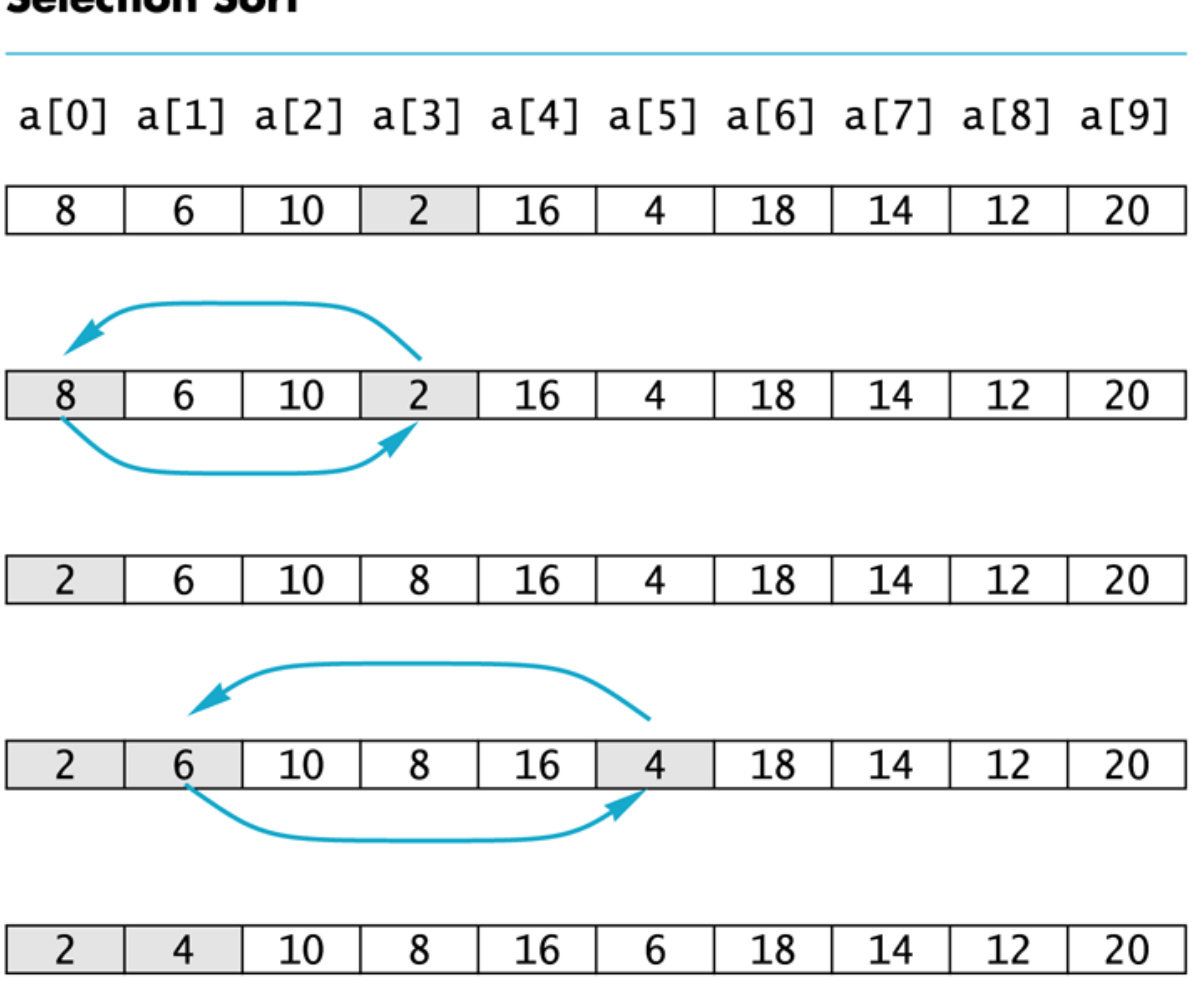
```
void search(const int a[], int low_end, int high_end,
                           int key, bool& found, int& location)
{
    int first = low_end;
    int last = high_end;
    int mid;

    found = false;//so far
    while ( (first <= last) && !(found) )
    {
        mid = (first + last)/2;
        if (key == a[mid])
        {
            found = true;
            location = mid;
        }
        else if (key < a[mid])
        {
            last = mid - 1;
        }
        else if (key > a[mid])
        {
            first = mid + 1;
        }
    }
}
```

# Display 7.11



**Selection Sort**



# Display 7.12 (1/2)

Back Next

**DISPLAY 7.12 Sorting an Array** *(part 1 of 2)*

```
1  //Tests the procedure sort.
2  #include <iostream>
3  void fill_array(int a[], int size, int& number_used);
4  //Precondition: size is the declared size of the array a.
5  //Postcondition: number_used is the number of values stored in a.
6  //a[0] through a[number_used - 1] have been filled with
7  //nonnegative integers read from the keyboard.
8  void sort(int a[], int number_used);
9  //Precondition: number_used <= declared size of the array a.
10 //The array elements a[0] through a[number_used - 1] have values.
11 //Postcondition: The values of a[0] through a[number_used - 1] have
12 //been rearranged so that a[0] <= a[1] <= ... <= a[number_used - 1].
13 void swap_values(int& v1, int& v2);
14 //Interchanges the values of v1 and v2.
15 int index_of_smallest(const int a[], int start_index, int number_used);
16 //Precondition: 0 <= start_index < number_used. Referenced array elements have
17 //values.
18 //Returns the index i such that a[i] is the smallest of the values
19 //a[start_index], a[start_index + 1], ..., a[number_used - 1].
20 int main( )
21 {
22     using namespace std;
23     cout << "This program sorts numbers from lowest to highest.\n";
24     int sample_array[10], number_used;
25     fill_array(sample_array, 10, number_used);
26     sort(sample_array, number_used);
27     cout << "In sorted order the numbers are:\n";
28     for (int index = 0; index < number_used; index++)
29         cout << sample_array[index] << " ";
30     cout << endl;
31     return 0;
32 }
33 //Uses iostream:
34 void fill_array(int a[], int size, int& number_used)
35 void sort(int a[], int number_used)
36 {
37     int index_of_next_smallest;
```

<The rest of the definition of `fill_array` is given in Display 7.9.>

*(continued)*

# Display 7.12 (2/2)


**DISPLAY 7.12** **Sorting an Array** *(part 2 of 2)*

```
38      for (int index = 0; index < number_used - 1; index++)
39      {//Place the correct value in a[index]:
40           index_of_next_smallest =
41                          index_of_smallest(a, index, number_used);
42           swap_values(a[index], a[index_of_next_smallest]);
43          //a[0] <= a[1] <=...<= a[index] are the smallest of the original array
44          //elements. The rest of the elements are in the remaining positions.
45      }
46  }
47
48  void swap_values(int& v1, int& v2)
49  {
50      int temp;
51      temp = v1;
52      v1 = v2;
53      v2 = temp;
54  }
55
56  int index_of_smallest(const int a[], int start_index, int number_used)
57  {
58      int min = a[start_index],
59          index_of_min = start_index;
60      for (int index = start_index + 1; index < number_used; index++)
61          if (a[index] < min)
62          {
63              min = a[index];
64              index_of_min = index;
65              //min is the smallest of a[start_index] through a[index]
66          }
67
68      return index_of_min;
69  }
```

***Sample Dialogue***

```
This program sorts numbers from lowest to highest.
Enter up to 10 nonnegative whole numbers.
Mark the end of the list with a negative number.
80 30 50 70 60 90 20 30 40 -1
In sorted order the numbers are:
20 30 30 40 50 60 70 80 90
```