CS1100
Introduction to Programming

Searching in Arrays

Random Q

Fill in the blanks:

```
int Sigma (int n) // Computes 1 + 2 + … + n
{
    if (n == 1)
        return(1);
    return (n ____ Sigma(______));
}
```


Searching

- Consider a lottery, where tickets numbered 1 through 100 are sold.
- Let **five** tickets be selected for a prize.
- You hold a ticket with number (X, say 41).
  - We need to know if your number has won a prize.

- Store the 5 winning numbers in an array
- Compare the array elements one-by-one to X.
  - If X is in the array, report "You won"
  - Else, report "You Lost"

Searching for Elements

- Given an array of numbers, is the value **X** present in the array?
  - WinNumbers[] = {45, 2, 67, 23, 89};
- If **X** (say 23) occurs in the array, return the index of the position where it occurs.
- If the numbers are not in sorted order, we have to scan the entire array to search for an element.

For loop for this ….

```
int SearchForNumber() {
    int WinNumbers[5] = {45, 2, 67, 23, 89};
      int num;
      printf("Enter your ticket number (1-100): ");
      scanf("%d", &num);
      for (int i = 0; i < 5; i++)
         if (WinNumbers[i] = = num)
           {
                  printf("You won a prize!\n"); return i;
           }
      }
      printf("Sorry. You lost!\n");
      return -1;
}
```


Random Q

- There is a sorted array with 1 Billion elements (approx. 2^{30})

1. If Linear Search is used, the worst-case number of elements compared is:

   ____

2. If a Cleverer Search technique is used (yet to be discussed in class), the worst-case number of elements compared is: ______________

Linear Search (While loop)

```
int Linear Search(int value, int array[ ], int n){
// array[0], array[1], …, array[n-1]
   int index = 0;
   while (index < n){
      if (array[index] == value) return index;
      else index++;
   }
   return NOTFOUND; /*calling function must interpret
                            this correctly! */
   } //Worst case: entire list is searched
```

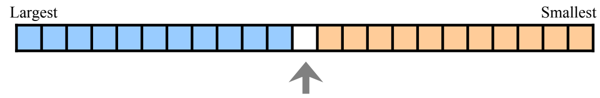

Reducing Search Time

- In LinearSearch, the entire list is searched in the worst case
- What if the list has 1 billion numbers?
- Can we reduce the search time?
- What if the list is always in sorted order (DESCENDING)?
  - int WinNumbers[5] = {89, 67, 45, 23, 2};
  - List can be in ASCENDING order too

Searching in a Sorted Array

- Given an array of marks sorted in ***descending*** order of marks, is there someone who got X marks?
- If X is high (say 92/100), one could start scanning from the left.
- If X is low (say 47/100), one could scan the array right to left.
- But what if we do not know whether X is high or low?

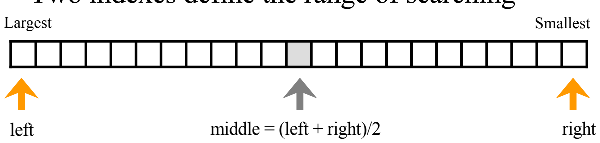
Divide and Conquer



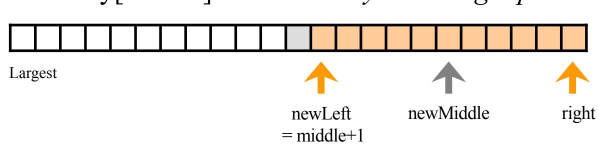
- Look at the middle element
- If array[middle] = = X, done
- If array[middle] > X, look *only in the right(second) part*
- Else look for the number *only in the left (first) part*
- The problem is reduced into a smaller problem
  - new problem is half the size of the original one
- *Recursively apply this strategy*

Divide and Conquer

- Two indexes define the range of searching

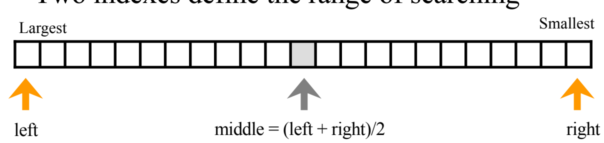


- If array[middle] > X look *only in the right part*

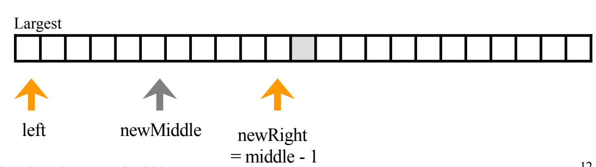


Divide and Conquer

- Two indexes define the range of searching



- If array[middle] < X look *only in the left part*



Comparison outcomes

- if array[middle] < X
  - left does not change
  - right = middle -1
- if array[middle] > X
  - left = middle + 1
  - right does not change
- if array[middle] = X
  - Found the element

Binary Search (also called Binary Chop)

- Starts with the full sorted array
  - left = 0 and right = N-1
- The range of search are the elements between left and right including array[left] and array[right]
- Search terminates if **right < left (i.e. left > right)**
- Otherwise
  - If (array[middle] == X) return middle
  - If (array[middle] > X) left = middle +1
  - Else right = middle -1

Binary Search (list is in descending order)

```
int BinarySearch(int value, int array[ ], int n){
    int left = 0, right = n-1;
    while (left <= right){
        middle = (left+right)/2;
        if (array[middle] == value) return middle;
        if (array[middle] > value) left = middle +1;
        else right = middle -1;
    }
    return INVALID; /*e.g. -1, calling function must
      interpret this                correctly! */
}
```


Example

0	1	2	3	4	5	6	7	8
89	78	67	56	45	34	23	12	1

- Array = {89, 78, 67, 56, 45, 34, 23, 12, 1}
- X = 12
  1. left = 0; right = 8; left <= right
     1. middle = 8/2 = 4; A[4] = 45; 45 > 12;
     2. left = 5;
  2. left = 5; right = 8; left <= right
     1. middle = 13/2 = 6; A[6] = 23; 23 > 12;
     2. left = 7;
  3. left = 7; right = 8; left <= right
     1. middle = 15/2 = 7; A[7] = 12; Found X in array!

Example-2

0	1	2	3	4	5	6	7	8
89	78	67	56	45	34	23	12	1

- Array = {89, 78, 67, 56, 45, 34, 23, 12, 1}
- X = 1
  1. left = 0; right = 8; left <= right
     1. middle = 8/2 = 4; A[4] = 45; 45 > 1;
     2. left = 5;
  2. left = 5; right = 8; left <= right
     1. middle = 13/2 = 6; A[6] = 23; 23 > 1;
     2. left = 7;
  3. left = 7; right = 8; left <= right
     1. middle = 15/2 = 7; A[7] = 12; 12 > 1;
     2. Left = 8;
  4. left = 8; right = 8; left <= right
     1. middle = 16/2 = 8; A[8] = 1; X is found.

Example

0	1	2	3	4	5	6	7	8
89	78	67	56	45	34	23	12	1

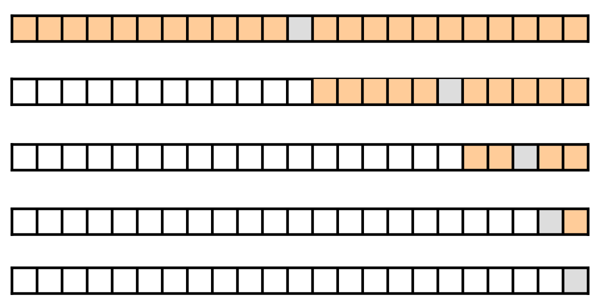
- Array = {89, 78, 67, 56, 45, 34, 23, 12, 1}
- X = 80
  1. left = 0; right = 8; left <= right
     1. middle = 8/2 = 4; A[4] = 45; 45 < 80;
     2. right = 3;
  2. left = 0; right = 3; left <= right
     1. middle = 3/2 = 1; A[1] = 78; 78 < 80;
     2. right = 0;
  3. left = 0; right = 0; left <= right
     1. middle = 0/2 = 0; A[0] = 89; 89 > 80;
     2. left = 1;
  4. left = 1; right = 0; left > right
     1. Terminate and report "X is not found in array"

Random Q

0	1	2	3	4
89	78	67	56	45

- Array = {89, 78, 67, 56, 45}
- X = 85
  1. left = __; right = ___;
     1. middle = ____;
     2. Updated left or right pointer = ?

Complexity of Binary Search



After each inspection the array reduces by half. For an array of size N there are about $\log_2 N$ inspections *in the worst case*.

Things not considered

- What if there are multiple elements in the list with the same value?
  - Which one will be reported by search?
- What if the array contains floating point numbers?
  - Equality is not always possible with such numbers
- What if the value compared is a string?
  - strcmp() can be used

Binary Search (list is in ascending order)

```
int BinarySearch(int value, int array[ ], int n){
    int left = 0, right = n-1;
    while (left <= right){
        middle = (left+right)/2;
        if (array[middle] == value) return middle;
        if (array[middle] < value) left = middle +1;
        else right = middle -1;
    }
    return INVALID; /*calling function must interpret this
                        correctly! */
}
```


Marks and Names

```
typedef struct {
    char *name;
    int mark;
} Student;

Student s1, s2;
Student s3 = { "Ramesh" , 79 };
Student studentarr[100];
```

name could itself be a struct made up of first name, middle name and last name…
Nested structures are allowed

Binary Search (list is in ascending order of names)

```
int BinarySearch(Student value, Student array[ ], int n){
    int left = 0, right = n-1; int compresult;
    while (left <= right){
        middle = (left+right)/2;
        compresult = strcmp(array[middle].name, value.name);
        if (compresult = = 0) return middle;
        if (compresult < 0)    left = middle +1;
        else right = middle -1;
    }
    return INVALID; /*calling function must interpret this
                        correctly! */
}
```

Exercises

- Modify the binary search to search in an array of Student datatypes:
  - Given a number X, return the name of at least one student who has obtained marks X, if such a student exists in the array
  - Given student name Y, return the marks obtained by the student, if the student name is in the array.

About GNU C Manual

- Want to know the syntax of C supported by GCC: https://www.gnu.org/software/gnu-c-manual/gnu-c-manual.pdf