Monsoon-2023

An Introduction to Program Design

CS200D Program Design

Lecture 1

Recall 1004D

- Basic C programming
- Control flow statements
- Data types and Functions
- Arrays and Pointers
- Problem solving

Introduction

- Computational Problem - Input - Output
- Algorithm - Step-by-step computational procedure
  - Mapping of input to output.

  - Algorithms in our Daily Life?

- Program - Implementation of an algorithm
  - In a programming language.
- Data Structure - Organization of data.

Computational Problems

- Well defined input and output
  - FIND_MAX(Integer_Array A)
    - Input: Array A, Output: max(A)
  - SEARCHING(Integer_Array A, Element x)
    - Input: Array A, Output: Yes if $x \in A$, No, otherwise
  - SORTING(Integer_Array A)
    - Input: Array A, Output: Elements of A in sorted order

More Computational Problems

- Resource Allocation
- Minimum Spanning Tree
- Shortest Path problem
- Longest common subsequence
- Pattern Matching
- Logic and algorithmic puzzles

Algorithm

- A sequence of computational steps that transform the input into output.
- A finite set of instructions that, if followed, accomplishes a particular task.
- An algorithm is a step-by-step procedure for solving a computational problem in a finite amount of time.

Algorithm Properties

- Input: Well defined input.
- Output: Well defined output.
- Finiteness: Execution terminates after a finite number of steps on all possible inputs.
- Definiteness: Each step must be clear and unambiguous.

Can you write an algorithm to print $\mathbb{N}$?

Algorithm vs. Program

An Example : Add two numbers

Input: Integers x and y, Output: x+y

Algorithm ADD(Integers x,y)	Source Code in 'C'
1. Declare Variables	`int a,b, sum;`
2. Get input	`scanf("%d%d",&a,&b);`
3. Add and assign the value	`sum=a+b;`
4. Display output result	`printf("Sum=%d",sum);`

Pseudocode conventions

- **Indentation** indicates block structure
- Keyword **to** - loop increments its value by 1 in each iteration
- Keyword **downto** - loop decrements its value by 1 in each iteration
- keyword **by** to indicate measure of loop counter update

Pseudocode conventions

- Variables are local to a given procedure
- A[i] indicates the i^{th} element of A
- A[i…j] indicates the elements of subarray of A from i to j
- To access object attributes: object_name.attribute_name Eg: A.length
- Parameters are passed to a procedure by **value.**

Pseudocode conventions

- A **return** statement immediately transfers control back to the point of call in the calling procedure.
- Multiple values to be returned in a single **return** statement
- Arithmetic, assignment, relational, boolean operators have the same meaning.

Array sum - Pseudocode

Array-Sum(A)

1. sum = 0
2. for i = 1 to A.length
    a. sum = sum + A[i]
3. return sum

```
#include <stdio.h>

int arraysum(int *a, int n)
{int sum=0;
for(int i=0;i<n;i++)
{
     sum+=a[i];
}
return sum;
}

int main()
{
int i,n,sum,a[10];
scanf("%d",&n);
for(i=0;i<n;i++)
{
     scanf("%d",(a+i));
}
sum=arraysum(a,n);
printf("Sum=%d",sum);
return 0;
}
```

Count occurrence - Pseudocode

Count-Occurrence(A, key)

1. count = 0
2. for i = 1 to A.length
    a. if A[i] == key
        i. count = count + 1
3. return count

```
int count(int * a, int n,
int key)
{
      int count=0;
      for(int i=0;i<n;i++)
      {
        if(a[i]==key)
        count++;
      }
      return count;
}
```

Algorithm Selection

- ★ Iterative solution
  - ○ Using a looping construct
  - ○ Loop iterator is required
- ★ Recursive solution
  - ○ Using a recursive function call
  - ○ Base case is required

An example: factorial of a number.

Think!!!

You can have an iterative and recursive solution to find factorial of a number.

Does every iterative solution has its equivalent recursive solution? Viceversa?

Some examples

1. Print all fibonacci numbers until a positive integer ‘n’.
2. Find a recursive solution to check whether a number is fibonacci or not?
3. Find a recursive solution to check whether a number is palindrome or not?
4. Enumerate all 4 digit palindrome numbers, and count them.

Checking Fibonacci

Check_Fibonacci(value)

1. current =1, previous =0
2. if (value==0)
   a. print the value is a Fibonacci Number
   b. return
3. while (current <= value)
   a. if (value==current)
      i. print the value is a Fibonacci Number
   b. else
      i. temp=current
      ii. current = current + previous
      iii. previous = temp
4. Print the value is NOT a Fibonacci Number

int Recursive_Check_Fibonacci(value, current, previous)

1. if (value<current)
   a. Return 0
2. else if (value==current or value==previous)
   a. Return 1
3. else
   a. Return Recursive_Check_Fibonacci(value, current+previous, current)

call Recursive_Check_Fibonacci(value, 1, 0)

More Problems

1. Given an Array, insert or delete elements from desired locations.
2. Find maximum frequent element in an array.
3. Set operations using arrays.
4. Combining Arrays, using subarrays of size k, $k \geq 1$.
5. Finding k^{th} minimum element in an array.

Linear Search

Linear-Search(A, key)

1. for i = 1 to A.length
    a. if A[i] = key
        i. return i
2. return -1

Linear Search

1. Loop invariant:
   a. At the start of ith iteration of the for loop, the subarray A[1 … i−1] does not contain the value key
2. Minimum and maximum number of comparisons
3. Average number of comparisons

Binary Search

Binary-Search(A, key)

1. low=0
2. High = A.length - 1
3. While low<= high
   a. Find mid = (low + high) / 2
   b. if(A[mid]==key)
      i. Print the search is successful
      ii. Print element found at position mid
   c. else if(A[mid]<key)
      i. low = mid +1
   d. else
      i. high = mid - 1
4. Return -1

Binary Search

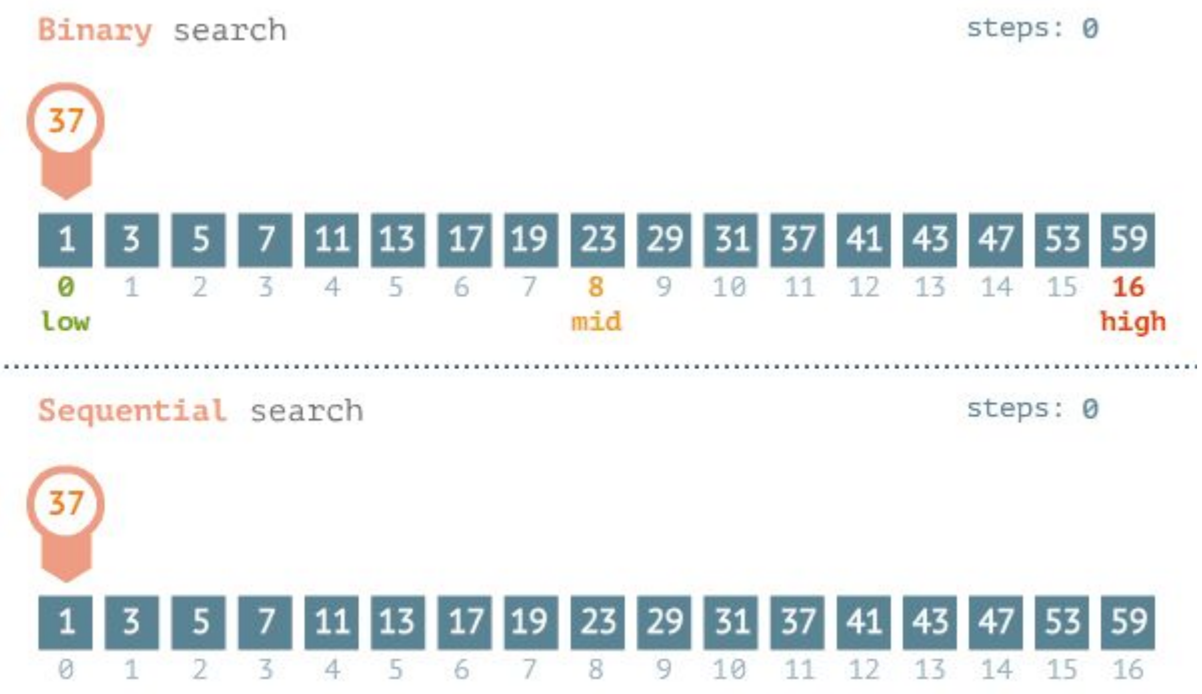
1. Loop invariant:
   a. At the start of ith iteration of the for loop, either the subarray A[low … high] contains the value key or it is not present in A.
2. Minimum and maximum number of comparisons
3. Average number of comparisons

Binary Search

Binary-Search(A, key, low, high)

1. if (high>low)
   a. Return -1
2. else
   a. Find mid = (low + high) / 2
   b. if(A[mid]==key)
      i. Print the search is successful and position is mid
   c. else if(A[mid]<key)
      i. Binary-Search(A, key, mid+1, high)
   d. else
      i. Binary-Search(A, key, low, mid-1)

Searching - Visualization



Binary Search

1. Proof of correctness
2. Minimum and maximum number of comparisons

A

index	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
value	1	2	3	4	9	15	23	25	26	29	31	33	35	40	44

Binary Search (A,14,1,15)

Binary Search

1. Proof of correctness
2. Minimum and maximum number of comparisons

A

index	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
value	1	2	3	4	9	15	23	**25**	26	29	31	33	35	40	44

Binary Search (A,14,1,15)

Binary Search

1. Proof of correctness
2. Minimum and maximum number of comparisons

A

index	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
value	1	2	3	4	9	15	23	25	26	29	31	33	35	40	44

Binary Search (A,14,1,15)

Binary Search

1. Proof of correctness
2. Minimum and maximum number of comparisons

A

index	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
value	1	2	3	4	9	**15**	23	25	26	29	31	33	35	40	44

Binary Search (A,14,1,15)

Binary Search

1. Proof of correctness
2. Minimum and maximum number of comparisons

A

index	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
value	1	2	3	4	**9**	15	23	25	26	29	31	33	35	40	44

Binary Search (A,14,1,15): unsuccessful after 4 comparisons