Lecture 2.1

- Dr. Anurag Sharma

Algorithms: Efficiency & Analysis

Algorithms

- An algorithm is a step-by-step procedure used to solve a problem. But making sure the developer is using the most efficient algorithm is very crucial no matter how fast computers become or how cheap memory gets.
- In order to determine how efficiently an algorithm solves a problem, we need to analyze the algorithm.
- Order helps group algorithms according to their eventual behavior.

Algorithms (cont.)

- A computer program is composed of individual modules, understandable by a computer, that solve specific tasks (such as sorting).
- The concentration here is not on the design of entire programs, but rather on the design of the individual modules that accomplish the specific tasks.
- These specific tasks are called **problems**. A problem is a question to which we seek an answer.

Algorithms (cont.)

- A problem may contain **variables** that are not assigned specific values in the statement of the problem. These variables are called parameters to the problem.
- Because a problem contains **parameters**, it represents a class of problems, one for each assignment of values to the parameters. Each specific assignment of values to the parameters is called an **instance** of the problem.

Algorithms (cont.)

- To produce a computer program that can solve all instances of a problem, we must specify a general step-by-step procedure for producing the solution to each instance. This step-by-step procedure is called an **algorithm**. We say that the algorithm solves the problem.

The Importance of Developing Efficient Algorithms

- Efficiency is an important consideration when working with algorithms. A solution is said to be efficient if it solves the problem within the required resource constraints.
- Since one problem can be solved by many different algorithms, it may be important to determine which one is the most efficient. Eg. Searching with sequential search and binary search.

Sequential Search vs Binary Search

- The sequential search algorithm begins at the first position in the array and looks at each value in turn until the item is found.
- The binary search algorithm first compares x with the middle item of the array. If they are equal, the algorithm is done. If x is smaller than the middle item, then x must be in the first half of the array (if it is present at all),and the algorithm repeats the searching procedure on the first half of the array

Binary Search (cont.)

- (That is, x is compared with the middle item of the first half of the array. If they are equal, the algorithm is done, etc.) If x is larger than the middle item of the array, the search is repeated on the second half of the array. This procedure is repeated until x is found or it is determined that x is not in the array.

Binary Search (cont.)

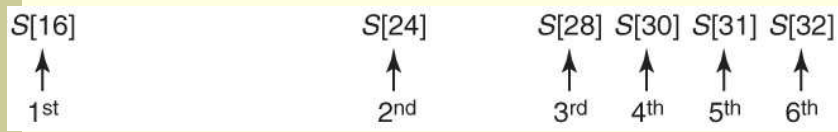

Figure 1.1: The array items that Binary Search compares with x when x is large than all the items in an array of size 32. The items are numbered according to the order in which they are compared.

Sequential Search

- Write the algorithm of sequential search for array of integers.
- Solution will be discussed in the lecture

Sequential Search in Java

```
static int sequentialSearch(final Comparable key,
final ArrayList<? extends Comparable> in){
    int loc = 0;
    while(loc<in.size() && in.get(loc).compareTo(key) != 0){
        loc++;
    }
    if(loc>=in.size())
        loc = -1;
    return loc;
}
```


Binary Search

- Write the algorithm of binary search for array of integers.
- Solution will be discussed in the lecture

Binary Search in Java

```java
static int binarySearch(final Comparable key,
final ArrayList<? extends Comparable> in){
    int low, mid, high; //indices
    int loc = -1;

    low = 0; high = in.size()-1;

    while(low<=high && loc == -1){
        mid = (int)Math.floor((low+high)/2.0);
        if(in.get(mid).compareTo(key) == 0)
            loc = mid;
        else if(in.get(mid).compareTo(key) > 0)
            high = mid-1;
        else
            low = mid+1;
    }

    return loc;
}
```


Efficiency comparison

- To compare Sequential Search to Binary Search, Sequential Search does *n* comparisons to determine that *x* is not in the array of size *n*. If *x* is in the array, the number of comparisons is no greater than *n*.
- A Binary Search is an algorithm for locating the position of an element in a sorted list. The method reduces the number of elements that need to be examined by two each time.
- Binary Search appears to be much more efficient than Sequential Search

Efficiency comparison (cont.)

- Compare the number of comparisons for the worst case:

Array Size	Sequential Search	Binary Search
32	32	6
64	64	7
128	128	8
n	n	$\log_2 n + 1$
1,024	1,024	11
1,048,576	1,048,576	21

- Therefore Binary Search appears to be much more efficient than Sequential Search, based on the number of comparisons done.

Recursive vs iterative Fibonacci

- The Fibonacci sequence is given as:
- $Fib(n) = Fib(n-1) + Fib(n-2)$ for $n >= 2$; $Fib(0) = 0$, $Fib(1) = 1$ Example: 0, 1, 1, 2, 3, 5, 8, …
- Recursive Algorithm (n^{th} Fibonacci term):

```
int fib(int n)
{
if (n <= 1) return n;
else
return fib(n-1) + fib(n-2);
}
```

Iterative (nth Fibonacci term)

```
int fib2(int n)
{
  index i;
  int f[0..n]; //array f[0] = 0;
  if (n > 0) {
    f[1] = 1;

 for (i = 2; i <= n; i++)
 f[i] = f[i-1] + f[i-2];
   }
   return f[n];
}
```


Recursive Fibonacci compuations

14 CHAPTER 1 • ALGORITHMS: EFFICIENCY, ANALYSIS, AND ORDER

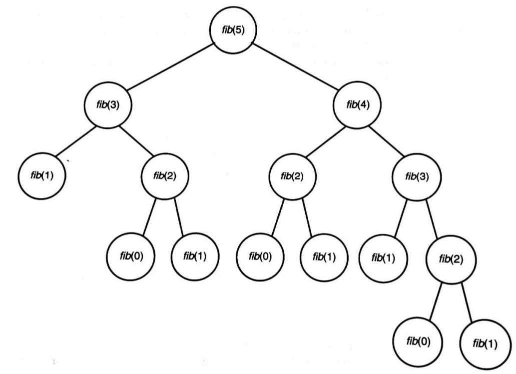

Figure 1.2 • The recursion tree corresponding to Algorithm 1.6 when computing the fifth Fibonacci term.

Efficiency comparison

- Recursive vs iterative Fibonacci.

n	Recursive	Iterative
0	1	1
1	1	2
2	3	3
3	5	4
4	9	5
n	$> 2^{n/2}$	*n* + 1
40	> 1,048,576	41