INTRODUCTION

Data Structures (CS2001) Fall 2022

Abeeda Akram

Program, Data and Algorithm

1. All programs manipulate data.
2. Data manipulation requires algorithms.

Data should be organized in such a way that its handling become efficient and easy.

Vs

A Data Structure

is a systematic way to organize and access data.

1. Data Structures are mostly user defined types (ADT) and form basic building blocks of a program

A good representation of data can enable us

1. to process the data more efficiently.
2. to produce good quality software.

Choice of an algorithm depends upon the underlying data structures

Course Objectives

To analyze the efficiency of any data structure

To decide the right data structure for a given problem

Problem

Search an integer value from a collection of ten 10 numbers.

Solution No. 1 (Multiple Variables)

```
int    a1 = 1, a2 = 2, a3 = 3, a4 = 4, a5 = 5, a6 = 6,
    a7 = 7, a8 = 8, a9 = 9, a10 = 10;

int key = 0;
cin >> key;
bool found = false;

if (key == a1)      found = true;
else if (key == a2)  found = true;
else if (key == a3)  found = true;
else if (key == a4)  found = true;
else if (key == a5)  found = true;
else if (key == a6)  found = true;
else if (key == a7)  found = true;
else if (key == a8)  found = true;
else if (key == a9)  found = true;
else if (key == a10) found = true;
```

Solution No. 2 (Linear Search)

```
const int size = 10;
int arr[size] = { 1, 5, 8, 4, 6, 3, 9, 7, 10, 2};

int key = 0;
cin >> key;
bool found = false;

for (int i = 0; i < size; i++) {
    if (arr[i] == key)
    found = true;
}
```

Solution No. 3 (Binary Search)

```
const int size = 10;
int arr[size] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
int key = 0;
cin >> key;
bool found = false;

int low = 0;
int high = size - 1;
while ((!found) && (low <= high))
{
    int mid = (low + high) / 2;
    if (arr[mid] == key)
        found = true;
    else if (arr[mid] > key)
        high = mid - 1;
    else
        low = mid + 1;
}
```

Which one is correct?

- All solutions are correct.
- All solve the same problem but use different coding styles and data structure.

Properties of Good Program

1. Correct
2. Efficient
3. Readable and easy to understand
4. Simple and easy to debug
5. Simple and easy to modify
6. Scalable
7. Reusable

Which is the Best solution?

1. **Multiple variables**
2. **Linear Search**
3. **Binary Search**

- Correct
- Efficient
- Readable and easy to understand
- Simple and easy to debug
- Simple and easy to modify
- Scalable
- Reusable

Algorithm Analysis

Efficiency of an algorithm can be measured in terms of:

1. Execution time (**time complexity**)
2. The amount of memory required (**space complexity**)

Which measure is more important?

Answers often depends on the limitations of the technology available at time of analysis.

Execution Time

Is the amount of time required to execute a program.

Factors that affect execution time:

1. The programming language
2. Quality of the compiler
3. Speed of the computer on which the program is going to be executed (processor, memory)
4. Operating System
5. Architecture 32-bit or 64-bit
6. Data Sets

Problem

Write a function which takes an integer array as input and returns the sum of its contents.

```
int  sum( int arr[], int size)
{
   int sum = 0;
   for (int i=0; i<size ; i++)
   {
      sum += arr[i];
   }
   return sum;
}
```

Driver

```
int main()
{
    int a[]={1,2,3,4};
    cout<< sum (a, sizeof(a)/sizeof(int) );
    return 0;
}
```

How much time does Sum function takes to execute?

Method 1 : Measure

```
int main()
{
    int a[]={1,2,3,4};

    time_t start, end;
    time(&start);

    cout<< sum (a, sizeof(a)/sizeof(int) );

    time(&end);
    cout << end-start << endl;

    return 0;
}
```

Method 2 : Operation Count

Estimate the performance of an algorithm through

- **The number of operations required to process an input**

Requires a function expressing relation between **n & t** called time complexity function **T(n)**

For calculating **T(n)** we need to compute the total number of program steps …

(can be the number of executable statements or meaningful program segment)

Method 2 : Operation Count

```
int count = 0;
int sum( int in[], int size)
{
    int sum = 0;
    count++; // for assignment
    for (int i=0; i<size ; i++ )
    {
        count++; // for loop

        sum += in[i];
        count++; // for addition
    }
    count++; // for last time of loop
    count++; // for return
    return sum;
}
```

$\boxed{T(n)=2n+3}$

Comparison

Array sum $T(n) = 2n + 3$

Matrix_sum for m = n $T(n) = 2n^2 + 2n + 3$

n	T(n)	T(n)
2	7	15
10	23	223
100	203	20203
200	403	80403
500	1003	501003

Which function is growing faster ?

Which term is growing faster ?

Time Complexity

10^9 instructions/second

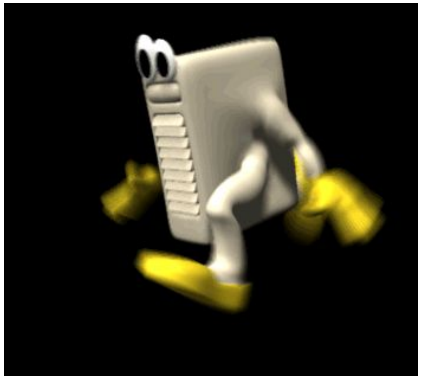
n	*n*	*nlogn*	n^2	n^3
1000	1mic	10mic	1milli	1sec
10000	10mic	130mic	100milli	17min
10^6	1milli	20milli	17min	32years

Time Complexity

10^9 instructions/second

n	n^4	n^{10}	2^n
1000	17min	3.2×10^{13} years	3.2×10^{283} years
10000	116 days	???	???
10^6	3×10^7 years	??????	??????

Faster Computer Vs Better Algorithm



Algorithmic improvement is more useful than just hardware improvement.

2^n to n^3

n^3 to n

Problems with T(n)

- **T(n) is difficult to calculate**
- **T(n) is usually very complicated so we need an approximation of T(n)….close to T(n).**
- **This measure of efficiency or approximation of T(n) is called**

ASYMPTOTIC COMPLEXITY or

ASYMPTOTIC ALGORITHM ANALYSIS