BITS F232: FOUNDATIONS OF DATA STRUCTURES & ALGORITHMS (1ST SEMESTER 2023-24) ALGORITHM COMPLEXITY

Chittaranjan Hota, PhD
Sr. Professor of Computer Sc.
BITS-Pilani Hyderabad Campus
hota[AT]hyderabad.bits-pilani.ac.in

RECAP: CONSTANT, LINEAR, LOGN, NLOGN

```
function isEvenOrOdd(n) {
    if (n%2 == 0)
        return even;
    else
        return odd;
}
```

```
list<int> numbers {1, 2, 3, 4};
for(int number : numbers)
{
    cout << number <<", ";
}
```

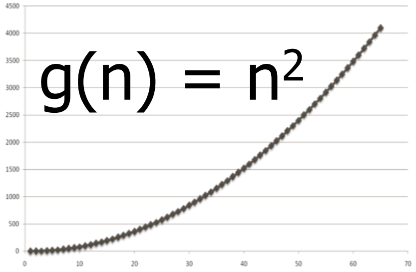
(printing out all the elements)

```
int binarySearch(int array[], int x, int low, int high)
{
  while (low <= high)
  {
      int mid = low + (high - low) / 2;
      if (array[mid] == x) return mid;
      if (array[mid] < x) low = mid +1;
      else
        high = mid - 1;
   }
   return -1;
}
```

```
int partition(int arr[], int low, int high) { int pivot=arr[high];
   int i = (low - 1);
   for (int j = low; j <= high - 1; j++) {
     if (arr[j] < pivot) { i++; swap(&arr[i], &arr[j]); } }
   swap(&arr[i + 1], &arr[high]); return (i + 1);
}
```

QUADRATIC FUNCTIONS

Quadratic: Given an input value ‘n’, the function ‘g’ assigns the product of ‘n’ with itself. Also, called ‘n squared’.

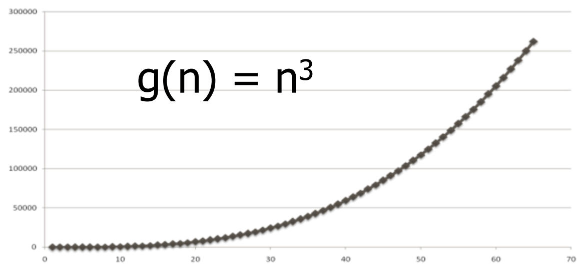


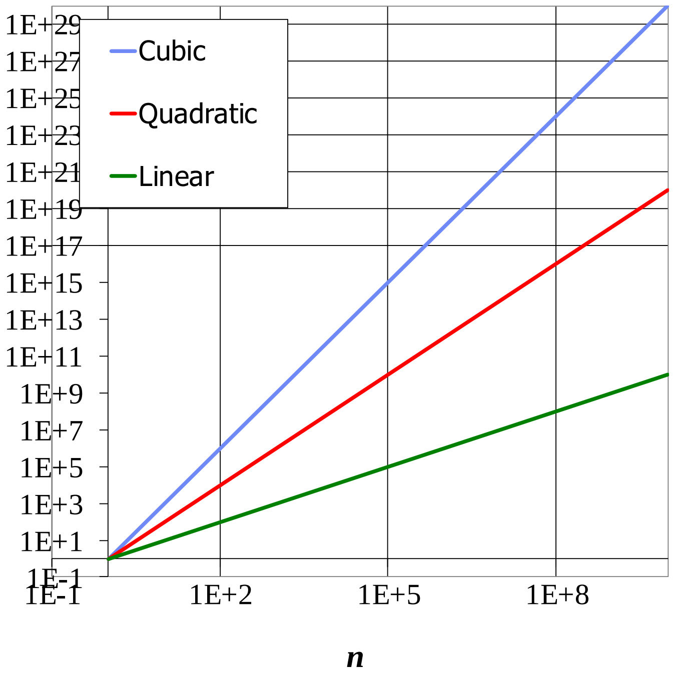
Used in analysing algorithms where nested loops are used.

```
for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
        // Print if their modulo equals to k
        if (i != j && arr[i] % arr[j] == k) {
            cout << "(" << arr[i] << ", "
                 << arr[j] << ")"
                 << " ";
            isPairFound = true;
        }
    }
}
```

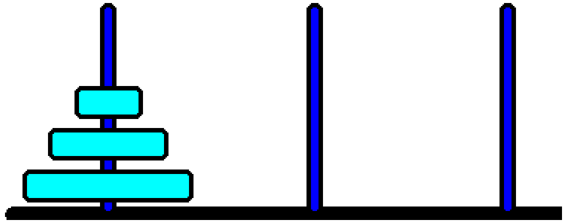
CUBIC, AND EXPONENTIAL FUNCTIONS

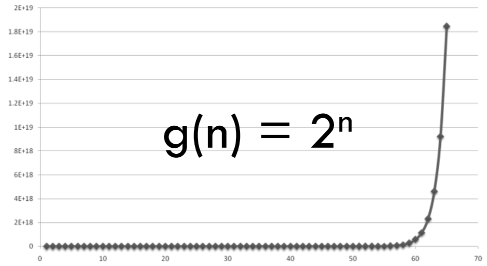
```
procedure NAIVE-MATRIX-MULTIPLY(A, B)
    n = A.rows
    let C be a new n × n matrix
    for i = 1 to n do
        for j = 1 to n do
            c_ij = 0
            for k = 1 to n do
                c_ij = c_ij + a_ik · b_kj
            end for
        end for
    end for
    return C
end procedure
```





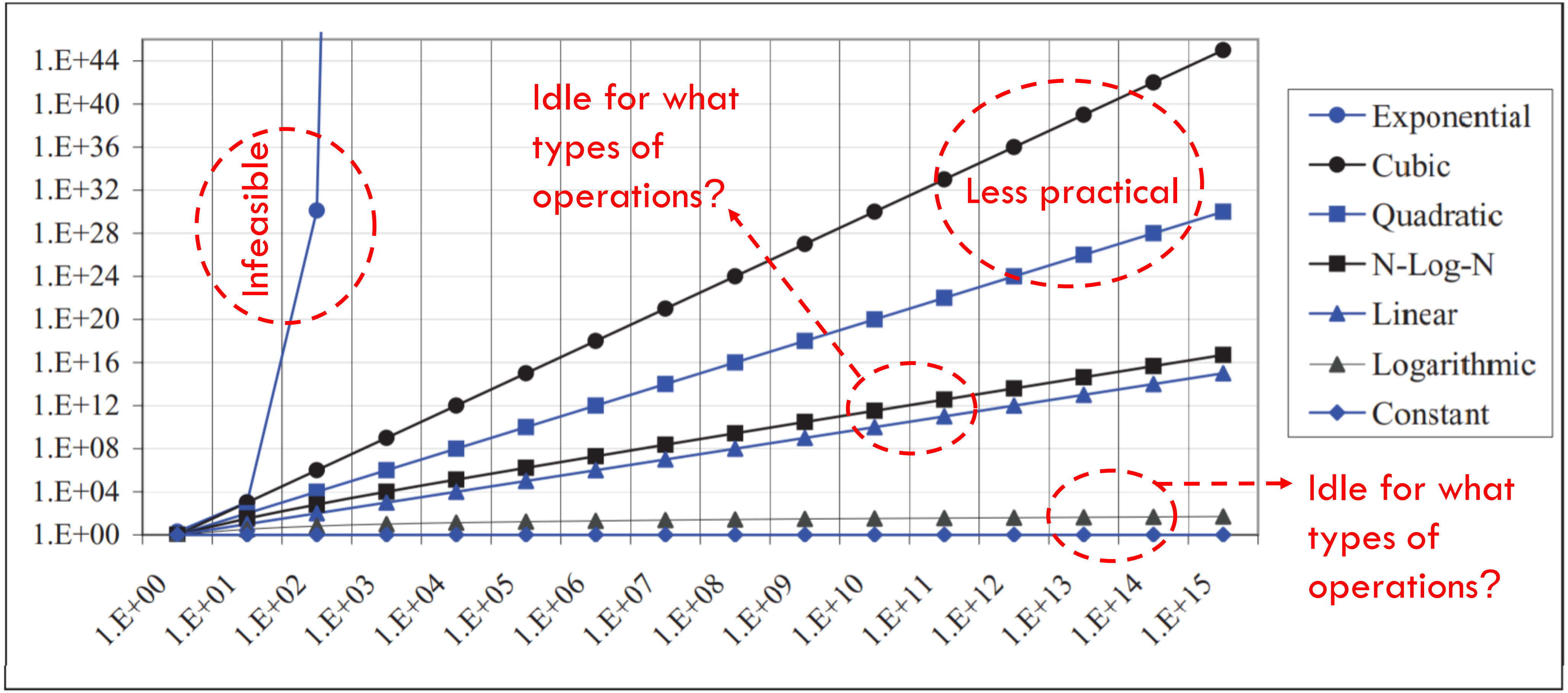
(log-log graph)






```
int Fibonacci (int number) {
   if (number <= 1)
         return number;
   return Fibonacci(number - 2) +
         Fibonacci(number - 1);
}
```

GROWTH RATES OF SEVEN FUNCTIONS



[log-log plot with growth rates (running times)as slopes]

ANALYSIS OF ALGORITHMS: EXPERIMENTAL STUDIES

```
// example1.cpp

#include <cstdlib>
#include <stdio.h>
using namespace std;
//declaration of functions
int func1();
int func2();

int func1(void) {
    int i=0,g=0;
    while(i++<100000) {
        g+=i;
    }
    return g;
}

int func2(void) {
    int i=0,g=0;
    while(i++<400000) {
        g+=i;
    }
    return g;
}

int main(int argc, char** argv) {
    int iterations = 10000;
    printf(`Number of iterations = %d\n`, iterations);
    while(iterations--) {
        func1();
        func2();
    }
}
```

g
p
r
o
f

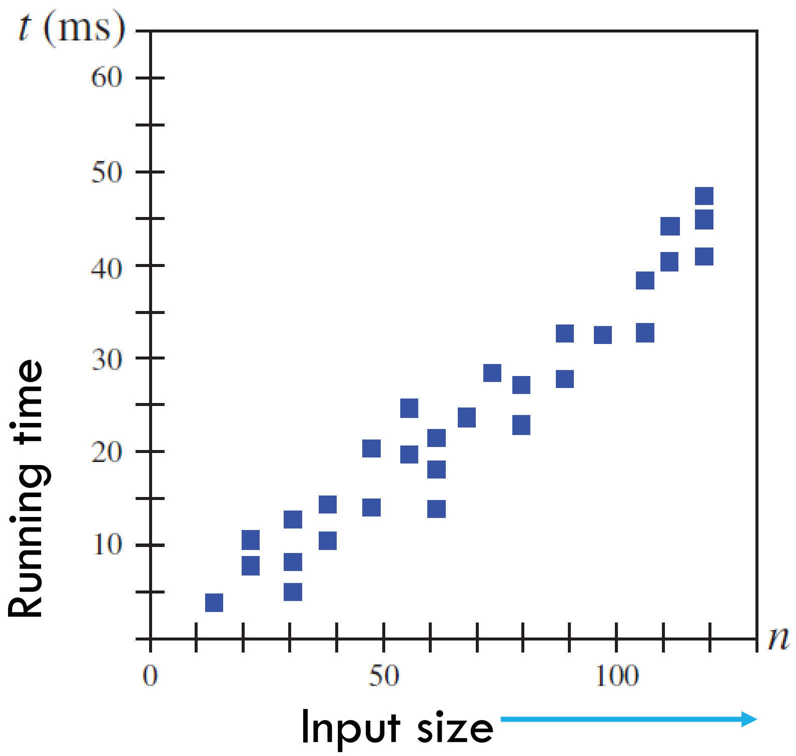
NEXT LAB

```
Flat profile:
Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  us/call  us/call  name
 80.80      9.59     9.59    10000   959.15   959.15  func2()
 20.33     12.00     2.41    10000   241.31   241.31  func1()
```

and the call graph:

```
Call graph (explanation follows)
granularity: each sample hit covers 2 byte(s) for 0.08% of 12.00 seconds
index % time    self  children    called     name
                                                 <spontaneous>
[1]    100.0    0.00   12.00                 main [1]
                9.59    0.00   10000/10000       func2() [2]
                2.41    0.00   10000/10000       func1() [3]
-----------------------------------------------
                9.59    0.00   10000/10000       main [1]
[2]     79.9    9.59    0.00   10000         func2() [2]
-----------------------------------------------
                2.41    0.00   10000/10000       main [1]
[3]     20.1    2.41    0.00   10000         func1() [3]
-----------------------------------------------
```

http://web.cecs.pdx.edu/~karavan/perf/book_gprof.html



Limitations:

- It is necessary to implement the complete algorithm, which may be difficult.
- Results may not be indicative of the running time on other inputs not included in the experiment.
- In order to compare two algorithms, the same hardware and software environments must be used

EXAMPLE RUN TIME…

```cpp
#include <chrono>
class Timer
{
private:
    std::chrono::time_point<std::chrono::high_resolution_clock> startTimePoint;
    std::chrono::time_point<std::chrono::high_resolution_clock> endTimePoint;
    double getTimeDifference();

public:
    Timer();
    void start();
    void stop();
    double getDurationInSeconds();
    double getDurationInMilliSeconds();
    double getDurationInMicroSeconds();
};
Timer::Timer() {}
void Timer::start()
{
    startTimePoint = std::chrono::high_resolution_clock::now();
}
void Timer::stop()
{
    endTimePoint = std::chrono::high_resolution_clock::now();
}
double Timer::getTimeDifference()
{
    auto start = std::chrono::time_point_cast<std::chrono::microseconds>(
    auto end = std::chrono::time_point_cast<std::chrono::microseconds>(en
    return end - start;
}
double Timer::getDurationInSeconds()
{
    return getDurationInMilliSeconds() * 0.001; // in seconds
}
double Timer::getDurationInMilliSeconds()
{
    return getTimeDifference() * 0.001; // in milli seconds
}
double Timer::getDurationInMicroSeconds()
{
    return getTimeDifference(); // in micro-seconds
}
```

NEXT LAB

```
Linear Search took: 37.647 ms.
Binary Search took: 0.002 ms.
Enter the size of the array: 7
Enter a sorted list of 7 elements:
10 20 30 40 50 60 70
Enter the target item to search for: 40
40 FOUND at index 3
Binary Search took: 0 ms.


...Program finished with exit code 0
Press ENTER to exit console.
```

recursive

Inside main()

```cpp
Timer timer; // initialize timer class object.

timer.start(); // start timer.

linearSearch(arr, n, n); // call to linear search

timer.stop(); // stop timer.

// function to get time in milli seconds
double milliSecs = timer.getDurationInMilliSeconds();

cout << "Linear Search took: " << milliSecs << " ms." << endl;

timer.start(); // start timer.

binarySearch(arr, n, n); // call to binary search

timer.stop(); // stop timer.

// function to get time in milli seconds
milliSecs = timer.getDurationInMilliSeconds();

cout << "Binary Search took: " << milliSecs << " ms." << endl;
```

CONTINUED...

```cpp
// finds and returns the n'th node from the end of the list.
template <typename DT>
SinglyLinkedNode<DT> *SinglyLinkedList<DT>::nthNodeFromEnd(int n)
{
    // code here
    counter = 0;
    tmp = NULL;
    nthNodeFromEndRecursive(head, n);
    return tmp; // return the n'th node from the end
}
```

```cpp
// recursive solution to find out the n'th node from the end.
template <typename DT>
void SinglyLinkedList<DT>::nthNodeFromEndRecursive(SinglyLinkedNode<DT>*head,int n)
{
    if (head == NULL)
        return;

    nthNodeFromEndRecursive(head->next, n);
    counter++;
    if (counter == n)
    {
        tmp = head;
    }
}
```

```cpp
        case '6':
            cout << "Enter N: ";
            cin >> a;
            timer.start();
            node = list.nthNodeFromEnd(a);
            timer.stop();
            if (node == NULL)
                cout << "Such a node does not exist." << endl;
            else
                cout << "N'th node from the end: " << node->dataItem << endl;
            cout << "Time spent: " << timer.getDurationInMilliSeconds() << " ms." << endl;
            break;
```

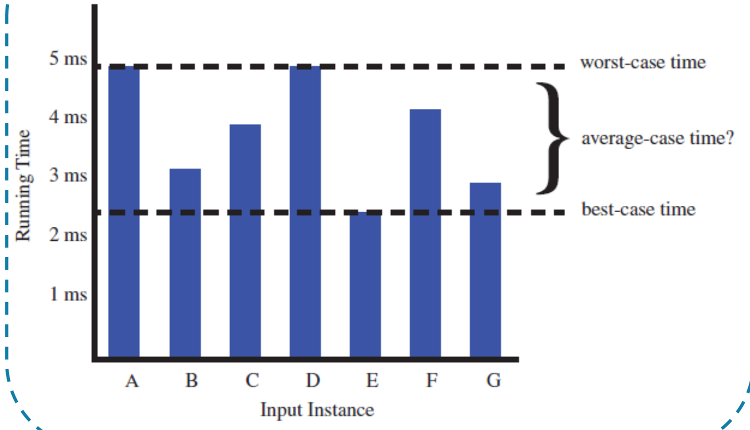
```
Please enter one of the following choices:
1 : Insert at end
2 : Delete from end
3 : Print Forward
4 : Print Backward
5 : Reverse List
6 : Get N'th node from the end
7 : Exit
3
10 20 30 40
Time spent: 0.019 ms.
+-----------------------------------------+
Please enter one of the following choices:
1 : Insert at end
2 : Delete from end
3 : Print Forward
4 : Print Backward
5 : Reverse List
6 : Get N'th node from the end
7 : Exit
6
Enter N: 2
N'th node from the end: 30
Time spent: 0.001 ms.
+-----------------------------------------+
Please enter one of the following choices:
1 : Insert at end
2 : Delete from end
3 : Print Forward
4 : Print Backward
5 : Reverse List
6 : Get N'th node from the end
7 : Exit
```

NEXT LAB

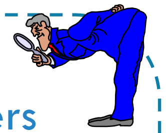
THEORETICAL ANALYSIS

- Uses a high-level description of the algorithm instead of an implementation.
- Characterizes running time as a function of the input size, *n*.
- Takes into account all possible inputs
- Allows us to evaluate the speed of an algorithm independent of the hardware/software environment

Natural measure of "goodness"



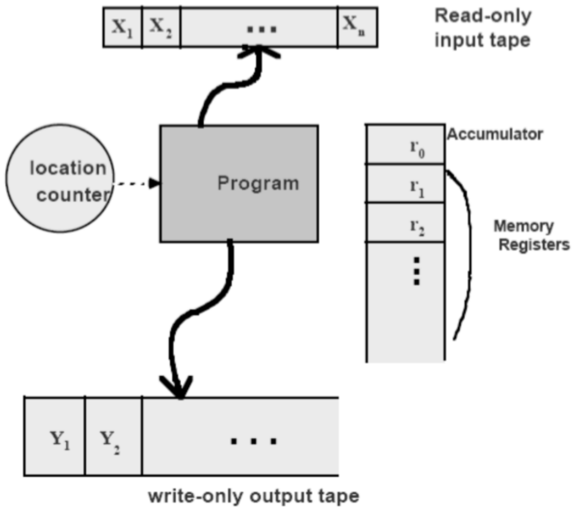
Pseudo



```
Algorithm arrayMax(A, n)
    Input: array A of n integers
    Output: maximum element of A

    max ← A[0]
    for i ← 1 to n − 1 do
        if A[i] > max then
            max ← A[i]
    return max
```

(The RAM Model)



Algorithm *arrayMax*(A, n)	# operations
currentMax ← $A[0]$	2
for $i \leftarrow 1$ **to** $n-1$ **do**	$2n$
if $A[i] >$ *currentMax* **then**	$2(n-1)$
currentMax ← $A[i]$	$2(n-1)$
{ increment counter i }	$2(n-1)$
return *currentMax*	1
	$8n-3$

The algorithm arrayMax executes about $8n$-3 primitive operations in the worst case.