Computational Complexity - Code Patterns

Data Structure and Algorithms

Slides credit: Ms. Saba Anwar, CUI Lahore

Major Growth Orders

The big-O notation gives an upper bound on the growth rate of a function

The statement "f(n) is O(g(n))" means that the growth rate of f(n) is no more than the growth rate of g(n). O(n^2) is O(n) but O(n) is not O(n^2).

We can use the big-O notation to rank functions according to their growth rate.

Seven functions are ordered by increasing growth rate in the sequence below, that is, if a function f (n) precedes a function g(n) in the sequence, then f(n) is O(g(n)):

constant	*logarithm*	*linear*	*n-log-n*	*quadratic*	*cubic*	*exponential*
1	$\log n$	n	$n \log n$	n^2	n^3	a^n

Table 4.1: Classes of functions. Here we assume that $a > 1$ is a constant.

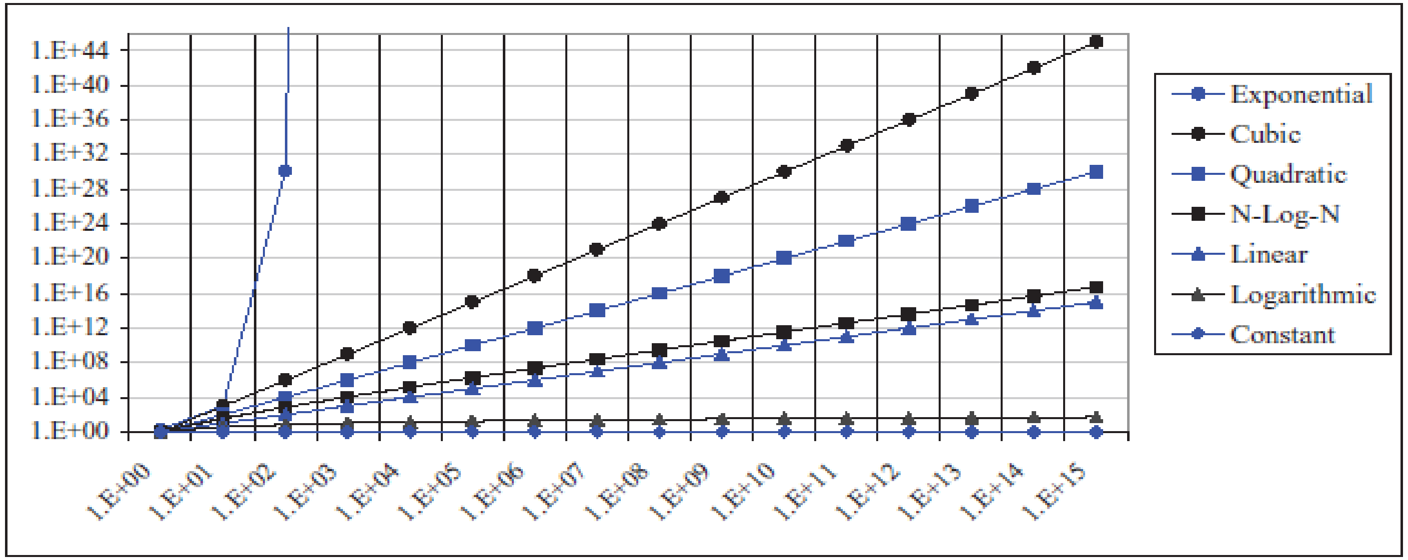
Growth Rate Comparison

Growth rate comparison:

Difference is more visible

at larger values of n

n	$\log n$	n	$n \log n$	n^2	n^3	2^n
8	3	8	24	64	512	256
16	4	16	64	256	4,096	65,536
32	5	32	160	1,024	32,768	4,294,967,296
64	6	64	384	4,096	262,144	1.84×10^{19}
128	7	128	896	16,384	2,097,152	3.40×10^{38}
256	8	256	2,048	65,536	16,777,216	1.15×10^{77}
512	9	512	4,608	262,144	134,217,728	1.34×10^{154}



Constant

If input size does not affect the algorithm time, e.g. find max among first/ last elements of an array ‘arr’ of size N.

1. Let a, b, max be integers
2. a = arr[0]
3. b = arr[N-1]
4. if a>=b
5. max=b;
6. else
7. max=a;
8. print max

Print some message a fixed number of times, even if this number is as large as 10^6.

1. for i=1 to 1000 step 1
2. print “welcome”

Linear

When algorithm time is directly proportional to input size.

This usually happens when the algorithm has Single loops, whose number of iterations depend on the input size.

Example 1:

```
1. sum=0
2. for (i=0;i<n;++i)
3.     sum++;
4. print sum
```

Note that even the following algorithm is linear. WHY?

```
1. for i=0 ; i< n-5; i++)
2.     print i*n
3. for (i=1;i<n/2; i+=1)
4.     print i*n
```

Example 2:

```
1. for i=0 ; i< n-5; i++)
2.     print i*n
```

Example 3:

```
1. for (i=1;i<n/2; i+=1)
2.     print i*n
```

Logrithmic

When algorithm time is proportional to logarithm (usually base 2) of input size.

This usually happens when the problem size is divided by half in each loop iteration.

Example 1: loop counter increased with multiplication factor. Each time the counter moves closer to n at a speed double than previous iteration.

```
1.  for (i=1;i<=n; i*=2)
2.      print i
```

Binary Search (An example of algorithm with Log(n) growth)

Input: Arr, V; We need to search value V.

Calculate mid=(low+high)/2

There can be three situations:

1. V is equal to Arr[mid]

   We have found the element

2. It is less than < Arr[mid]

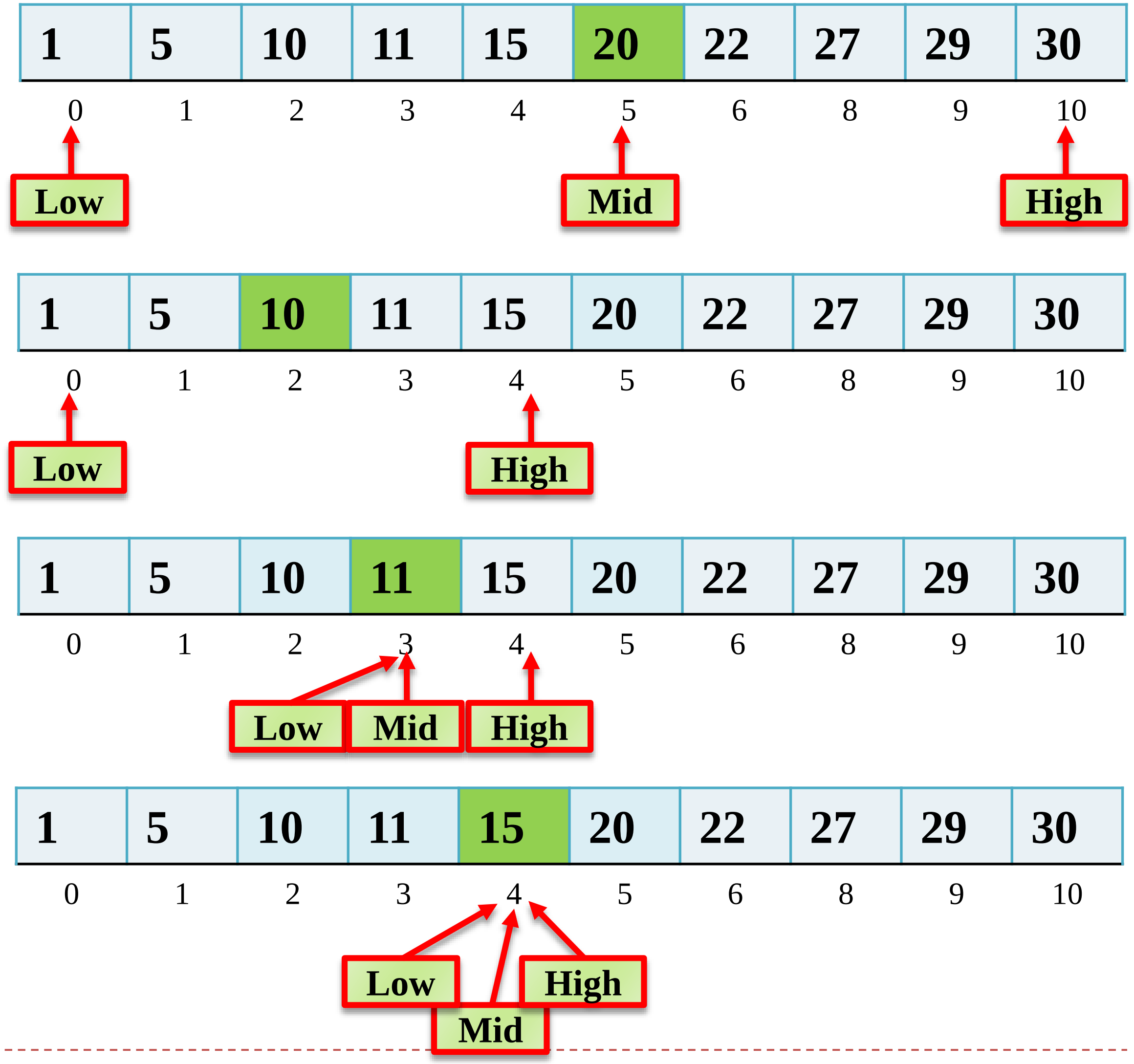
   It cannot be in right half of array

   So search in left half only

3. It is greater than Arr[mid]

   It cannot be in left half of array

   So search in right half only



Note how the problem size is being reduced by half in every loop iteration.

Binary Search

Algorithm:BINARY_SEARCH(A, N, V)

Input: Sorted Array in ascending order , lower and upper index of list, value to be searched

Output: index of value if found

Steps:

```
Start
1.  Set low=0, high=N-1
2.  While (low <= high)
3.      mid = (low +  high) / 2;
4.      If (V < A[mid])             //If value in the left half
5.          high = mid - 1;    //Update high index
6.      Else If (V >  A[mid])   //If  value in  the right half
7.          low = mid + 1;          //Update the low index
8.      Else
9.          return mid;  // value == Array[mid]
10.     END If
11.   End While
12.   return -1;   //index was not found
End
```

The binary search gets its name because the algorithm continually divides the list into two parts.

Binary search algorithm is good for larger arrays.
If array size is very small, there is not huge difference between linear search and binary search algorithm time

Linearithmic/nlogn

Combination of linear and logarithmic.

```
1. for (i=1;i<=n; i++)
2.     for(i=1;i<=n; i*=2)
3.             print i
```

Quadratic

When algorithm time is directly proportional to square of input size.

Nested loops

Example 2

```
1. sum=0
2.  for (i=0;i<n;++i)
3.      for (j=0;j<n;++j)
4.              sum++
5. print sum
```

Example 2

```
1. for (i=1;i<n; i++)
2.     for (j=1;j<i; j++)
3.             print i
```

Cubic

When algorithm time is directly proportional to cube of input size

Triple nested loops

1. sum=0
2. for (i=0;i<n;++i)
3. for (j=0;j<n;++j)
4. for (k=0;k<n;++k)
5. sum++
6. print sum

Exponential

Involves dynamic programming and brute force algorithms

Few recursion problems also run in exponential time.