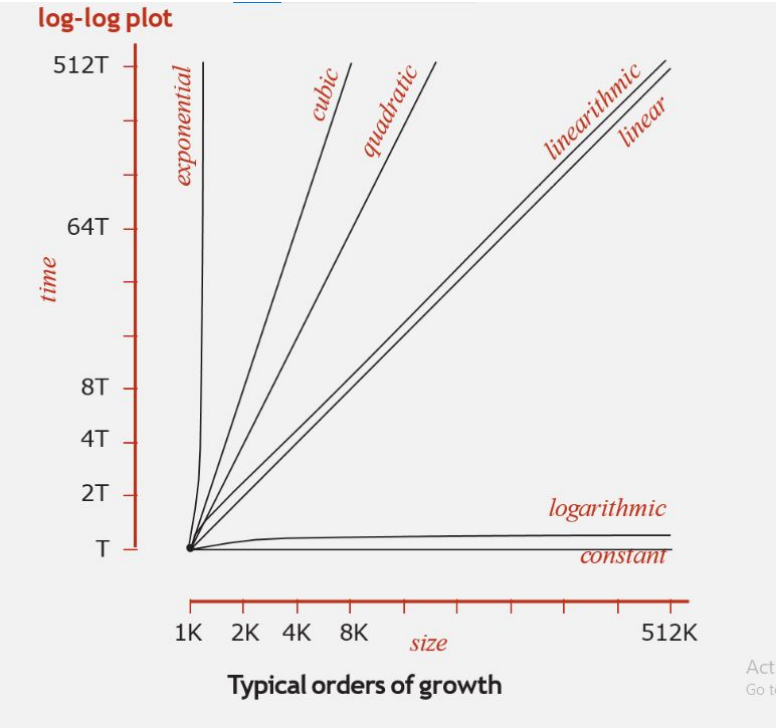
Runtime Analysis
C++ STL

IET CP SMP 2021-22

Common order-of-growth classifications

The small set of functions 1, log N, N, N log N, N^2, N^3, and 2^N suffices to describe order-of-growth of typical algorithms.



Common order-of-growth classifications

order of growth	name	typical code framework	description	example	T(2N) / T(N)
1	constant	`a = b + c;`	statement	add two numbers	1
log N	logarithmic	`while (N > 1) { N = N / 2; ... }`	divide in half	binary search	~ 1
N	linear	`for (int i = 0; i < N; i++) { ... }`	loop	find the maximum	2
N log N	linearithmic	[see mergesort lecture]	divide and conquer	mergesort	~ 2
N^2	quadratic	`for (int i = 0; i < N; i++) for (int j = 0; j < N; j++) { ... }`	double loop	check all pairs	4
N^3	cubic	`for (int i = 0; i < N; i++) for (int j = 0; j < N; j++) for (int k = 0; k < N; k++) { ... }`	triple loop	check all triples	8
2^N	exponential	[see combinatorial search lecture]	exhaustive search	check all subsets	T(N)

Practical Implications of order-of-growth

growth rate	problem size solvable in minutes			
	1970s	1980s	1990s	2000s
1	any	any	any	any
log N	any	any	any	any
N	millions	tens of millions	hundreds of millions	billions
N log N	hundreds of thousands	millions	millions	hundreds of millions
N^2	hundreds	thousand	thousands	tens of thousands
N^3	hundred	hundreds	thousand	thousands
2^N	20	20s	20s	30

Asymptotic Analysis

- Asymptotic analysis of an algorithm refers to defining the mathematical bound/framing of its run-time performance.
- Asymptotic analysis is input bound i.e., if there's no input to the algorithm, it is concluded to work in a constant time. Other than the "input" all other factors are considered constant.
- Usually, the time required by an algorithm falls under three types :

1) Best Case : Minimum time required for program execution (Determined by easiest input)
2) Average Case : Average time required for program execution

   (Determined by random input)
3) Worst Case : Maximum time required for program execution

   (Determined by most difficult input)

Asymptotic Notations

Following are the commonly used asymptotic notations to calculate the running time complexity of an algorithm.

- Big Oh Notation, O

  The notation O(n) is the formal way to express the upper bound of an algorithm's running time. It measures the worst case time complexity or the longest amount of time an algorithm can possibly take to complete.

- Omega Notation, Ω

  The notation Ω(n) is the formal way to express the lower bound of an algorithm's running time. It measures the best case time complexity or the best amount of time an algorithm can possibly take to complete.

- Theta Notation, θ

  The notation θ(n) is the formal way to express both the lower bound and the upper bound of an algorithm's running time.

Examples of time complexity

1. O(1) - Constant time

```
#include <stdio.h>
int main()
{
  int a = 4;
  int b = 6;
  int c;
  c = a + b;
  printf(%d, c);
}
```

2. O(n) - Linear time

```
int count(int arr[], int n)
{
  int sum = 0, i;
  for(i = 0; i < n; i++)  //Control statement
  {
     sum = sum + arr[i];
  }
  return sum;
}
```

Examples of time complexity

3) $O(n^2)$ Quadratic Time

```
int i,j, n = 8;
for (i = 1; i <= n; i++)
{
for (j = 1; j <= n; j++)
{
printf("FACE Prep");
} } }
```

4) O(log n) logarithmic time

```
while(low<=high)
{
mid=(low+high)/2;
if(n<arr[mid])
high=mid-1;
elseif(n>arr[mid])
low=mid+1;
elsebreak;
}
```

C++ Language

- Widely used and very popular
- Closely related to C Language
- Versions- C++11, C++14, C++17

A basic C++ program

```
1   #include <iostream>
2
3   using namespace std;
4
5   int main()
6   {
7       // single line comment
8       /*
9       multi-line
10      comment
11      */
12      cout << "Hello world!";
13      return 0;
14  }
15
```

- Preprocessor Directives
- Namespaces
- Main function
- Block
- Comments
- Standard Input/Output

C++ STL - Standard Template Library

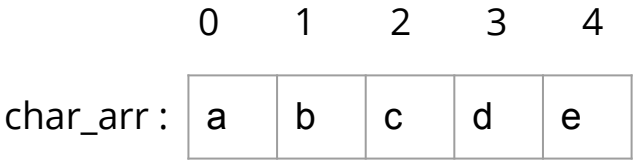
- A set of C++ template classes to provide general-purpose classes and functions with templates that implement many popular and commonly used algorithms and data structures like vectors, lists, queues, and stacks
- Components
  - Algorithms
  - Containers
  - Functions
  - Iterators

Containers

- Vectors
- Lists
- Queues
- Stacks
- Priority queues
- Unordered Sets
- Sets
- Maps
- And many more...

Arrays

- An array is a collection of items stored at contiguous memory locations
- To store multiple items of the same datatype together.



To access the element of an array we use, ‘ char_arr[i] ’, where ‘i’ is the index of the element.

Vectors

- These are dynamic arrays with the ability to resize itself automatically when an element is inserted or deleted
- Functions associated with Vectors:
  - begin()
  - end()
  - size()
  - resize(n)
  - empty()
  - front()
  - back()
  - push_back()
  - pop_back()
  - And many more...

Arrays vs Vectors

- Size of arrays are fixed whereas the vectors are resizable i.e they can grow and shrink
- Arrays cannot be copied or assigned directly whereas Vectors can be copied or assigned directly

Implementing Vectors

```cpp
#include <iostream>
#include <vector>
using namespace std;

int main()
{
    int n,i;
    vector<int> v;

    cin>>n;
    for(i=0;i<n;i++)
    {
        int vi;
        cin>>vi;
        v.push_back(vi);
    }

    for(i=0;i<v.size();i++)
    {
        cout<<v[i]<<" ";
    }
}
```

Program to take a vector of size ‘n’ as input and output the same vector.

Program Optimizations (Fast I/O)

- It is necessary to read input as fast as possible to reduce execution time.
- Optimizations:
  - Use scanf/printf instead of cin/cout
  - To achieve the speed as scanf/printf we use the following lines,

```
ios_base::sync_with_stdio(false);

cin.tie(NULL);

cout.tie(NULL);
```

  - Including Standard Template Library

```
#include <bits/stdc++.h>
```

Basic Template for CP

```
#include <bits/stdc++.h>

using namespace std;

int main()

{

    ios_base::sync_with_stdio(false);

    cin.tie(NULL);

    cout.tie(NULL);

    return 0;

}
```