COMPLEXITY ANALYSIS

DESIRED PROPERTIES OF A GOOD ALGORITHM

- Any good algorithm should satisfy 2 obvious conditions:
  - compute correct (desired) output (for the given problem)
  - be effective (fast)

1. correctness of algorithm
2. complexity of algorithm

DESIRED PROPERTIES OF A GOOD ALGORITHM

- Complexity of algorithm measures how fast is the algorithm (time complexity) and what amount of memory it uses (space complexity)

- 2 basic resources in computations
  - Space complexity
  - Time Complexity

TIME COMPLEXITY: ANALYSIS

TIME COMPLEXITY

- Worst-case
  - An upper bound on the running time for any input of given size
- Average-case
  - Assume all inputs of a given size are equally likely
- Best-case
  - The lower bound on the running time

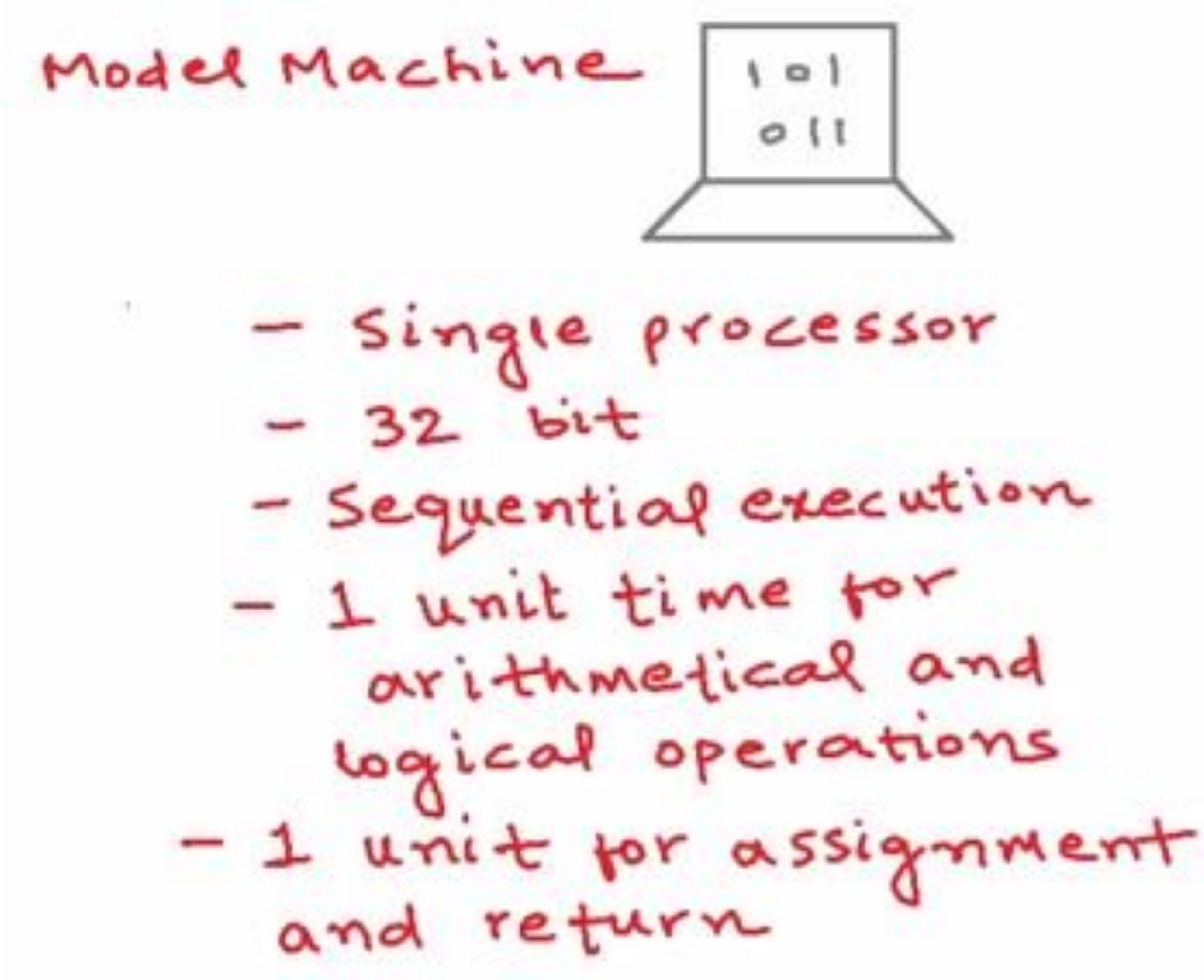
HOW TO ANALYZE TIME COMPLEXITY

Running time depends upon:

- ✗ 1) Single vs multi processor
- ✗ 2) Read/write speed to memory
- ✗ 3) 32 bit vs 64-bit
- ✓ 4) Input

  ↳ rate of growth of time

MODEL MACHINE



HOW TO ANALYZE TIME COMPLEXITY

```
Sum(a, b)
{
 return a+b
}
```

$T_{sum} = 2$

↓

Constant time

$O(1)$

HOW TO ANALYZE TIME COMPLEXITY

```
SumOfList(A, n)
{
1.  total = 0
2.  for i = 0 to n-1
3.      total = total + A_i
4.  return total
}
```

Cost	no. of times
1	1
2	n + 1
2	n
1	1

$$T_{sumoflist} = 1 + 2(n+1) + 2n + 1$$
$$= 4n + 4$$

$O(n)$

EXERCISE

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

int main ()
{
   // for loop execution
   for(int i = 0; i < 10; i = i + 1 )
   {
       cout << "value of i : " << i << endl;
   }
   return 0;
}
```

TIME COMPLEXITY: GENERAL RULES

SOME GENERAL RULES

We analyze time complexity for:

a) Very large input-size

b) Worst case scenario

Rule: a) drop lower order terms

b) drop constant multiplier

$T(n) = n^3 + 3n^2 + 4n + 2$

$O(n^3)$

$T(n) = 17n^4 + 3n^3 + 4n + 8$

$= O(n^4)$

$T(n) = 16n + \lg n$

$= O(n)$

SOME GENERAL RULES

Rule: Running Time = $\sum$ Running time of all fragments

Simple statements — Fragment 1

```
int a;
a = 5
a++;
```

$O(1)$

Single loop — Fragment 2

```
for(i=0;i<n;i++)
{
    // simple statements
}
```

$O(n)$

Nested loop — Fragment 3

```
for(i=0;i<n;i++)
{
    for(j=0;j<n;j++)
    {
        // simple statements
    }
}
```

$O(n^2)$

```
Function ( )
{ int a;
  a = 5
  a++;                                  } O(1)
  for(i=0;i<n;i++)
  {
     // simple statements               } O(n)
  }
  for(i=0;i<n;i++)
  {
     for(j=0;j<n;j++)
     {
        // simple statements            } O(n^2)
     }
  }
}
```

$$T(n) = O(1) + O(n) + O(n^2)$$

$$= O(n^2)$$

```
Function ( )
{
  if (some Condition)
  {
     for(i=0; i<n; i++)
     {  //simple statements           } O(n)
     }
  }
  else
  {  for(i=0; i<n; i++)
     {  for(j=0; j<n; j++)
        {  // simple statements       } O(n^2)
        }
     }
  }
}
```

$T(n) = O(n^2)$

Rule : Conditional Statements

Pick complexity of condition which is worst case

EXERCISE

```
#include<iostream.h>
#include<conio.h>
void  main()
{
int  a,no,sum=0;
clrscr();
cout<<"Enter  any  num  :  ";
cin>>no;
while(no>0)
{
a=no%10;
no=no/10;
sum=sum+a;
}
cout<<"\nSum  of  digits:  "<<sum;
getch();
}
```

EXERCISE

```
char[] arr = { 'a', 'b', 'b', 'd', 'e' };
char invalidChar = 'b';
int ptr = 0, N = arr.Length;
for (int i = 0; i < n; i++)
{
    if (arr[i] != invalidChar)
    {
        arr[ptr] = arr[i];
        ptr++;
    }
}

for (int i = 0; i < ptr; i++)
{
    Console.Write(arr[i]);
    Console.Write(' ');
}
Console.ReadLine();
```

EXERCISE

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

int main ()
{
   int i, j;

   for(i=0; i<=5; i++) {

      for(j=0; j <= 5; j++) {
        cout << i << j <<" \t";
      }

   cout <<"\n";
   }

   return 0;
}
```