Introduction

- $T(n)$: Time / Steps taken by an algorithm for input of size n
- $S(n)$: Space (memory cells) required by an algorithm for input of size n

Shiv Nadar University

Introduction

- $T(n)$: Time / Steps taken by an algorithm for input of size n
- $S(n)$: Space (memory cells) required by an algorithm for input of size n
- It does not make much sense to express $T(n)$ in standard time units (Why?)

Introduction

- $T(n)$: Time / Steps taken by an algorithm for input of size n
- $S(n)$: Space (memory cells) required by an algorithm for input of size n
- It does not make much sense to express $T(n)$ in standard time units (Why?)
- Does the time taken by an algorithm depend on the input size n alone?

Introduction

- $T(n)$: Time / Steps taken by an algorithm for input of size n
- $S(n)$: Space (memory cells) required by an algorithm for input of size n
- It does not make much sense to express $T(n)$ in standard time units (Why?)
- Does the time taken by an algorithm depend on the input size n alone?
- What would be useful to us is to talk about the "growth rate" of $T(n)$
- And express that growth rate in terms of known simple functions — $T(n) \propto f(n)$

Introduction

- $T(n)$: Time / Steps taken by an algorithm for input of size n
- $S(n)$: Space (memory cells) required by an algorithm for input of size n
- It does not make much sense to express $T(n)$ in standard time units (Why?)
- Does the time taken by an algorithm depend on the input size n alone?
- What would be useful to us is to talk about the "growth rate" of $T(n)$
- And express that growth rate in terms of known simple functions — $T(n) \propto f(n)$
- Normally we consider the time / steps taken in the worst-case

Introduction

- $T(n)$: Time / Steps taken by an algorithm for input of size n
- $S(n)$: Space (memory cells) required by an algorithm for input of size n
- It does not make much sense to express $T(n)$ in standard time units (Why?)
- Does the time taken by an algorithm depend on the input size n alone?
- What would be useful to us is to talk about the "growth rate" of $T(n)$
- And express that growth rate in terms of known simple functions — $T(n) \propto f(n)$
- Normally we consider the time / steps taken in the worst-case
- Other analysis include best-case and average-case

The Notations

- Big-Oh notation provides a tight upper bound for how $T(n)$ grows
  - For example, if $T(n) = 5n^2 + 3n - 2$, then it is $O(n^2)$, which means that $T(n)$ grows like the function n^2, but not faster than that

The Notations

- Big-Oh notation provides a tight upper bound for how $T(n)$ grows
  - For example, if $T(n) = 5n^2 + 3n - 2$, then it is $O(n^2)$, which means that $T(n)$ grows like the function n^2, but not faster than that
- Big-Omega notation provides a tight lower bound for how $T(n)$
  - $T(n)$ is $\Omega(g(n))$, if $T(n)$ grows like $g(n)$, but not slower than that
  - This notation is not widely used

SHIV NADAR UNIVERSITY

The Notations

- Big-Oh notation provides a tight upper bound for how $T(n)$ grows
  - For example, if $T(n) = 5n^2 + 3n - 2$, then it is $O(n^2)$, which means that $T(n)$ grows like the function n^2, but not faster than that
- Big-Omega notation provides a tight lower bound for how $T(n)$
  - $T(n)$ is $\Omega(g(n))$, if $T(n)$ grows like $g(n)$, but not slower than that
  - This notation is not widely used
- Big-Theta notation combines both
  - $T(n)$ is $\Theta(f(n))$, if $f(n)$ is both upper bound and lower bound for $T(n)$ (with different constants)
  - In other words, for any $T(n)$ and $f(n)$, $T(n)$ is $\Theta(f(n))$ if and only if $T(n)$ is $O(f(n))$ and $T(n)$ is $\Omega(f(n))$

SHIV NADAR UNIVERSITY

Big-Theta Theorem

Theorem

Any polynomial $T(n) = \sum_{i=0}^{d} a_i n^i$ *with* $a_d > 0$ *is* $\Theta(n^d)$

Shiv Nadar University

Big-Theta Theorem

Theorem

Any polynomial $T(n) = \sum_{i=0}^{d} a_i n^i$ *with* $a_d > 0$ *is* $\Theta(n^d)$

Theorem

As a special case, when $d = 0$, $T(n)$ *is a constant and can be expressed as* $\Theta(1)$

Shiv Nadar University

Typical Growth Rates

- $O(1)$
- $O(log\ n)$
- $O(log^2\ n)$
- $O(n)$
- $O(n\ log\ n)$
- $O(n^2)$
- $O(n^3)$
- $O(2^n)$

Typical Growth Rates

Constant Time

- $O(1)$
- $O(log\ n)$
- $O(log^2\ n)$
- $O(n)$
- $O(n\ log\ n)$
- $O(n^2)$
- $O(n^3)$
- $O(2^n)$

SHIV NADAR UNIVERSITY

Typical Growth Rates

- $O(1)$
- $O(log\ n)$
- $O(log^2\ n)$
- $O(n)$
- $O(n\ log\ n)$
- $O(n^2)$
- $O(n^3)$
- $O(2^n)$

Constant Time

Logarithmic Time

Shiv Nadar University

Typical Growth Rates

- $O(1)$ — Constant Time
- $O(log\ n)$ — Logarithmic Time
- $O(log^2\ n)$
- $O(n)$ — Linear Time
- $O(n\ log\ n)$
- $O(n^2)$
- $O(n^3)$
- $O(2^n)$

Typical Growth Rates

- $O(1)$
- $O(log\ n)$
- $O(log^2\ n)$
- $O(n)$
- $O(n\ log\ n)$
- $O(n^2)$
- $O(n^3)$
- $O(2^n)$

Constant Time

Logarithmic Time

Linear Time

Polynomial Time

SHIV NADAR UNIVERSITY

Typical Growth Rates

- $O(1)$ Constant Time
- $O(log\ n)$ Logarithmic Time
- $O(log^2\ n)$
- $O(n)$ Linear Time
- $O(n\ log\ n)$
- $O(n^2)$ Polynomial Time
- $O(n^3)$
- $O(2^n)$ Exponential Time

Shiv Nadar University

Typical Growth Rates

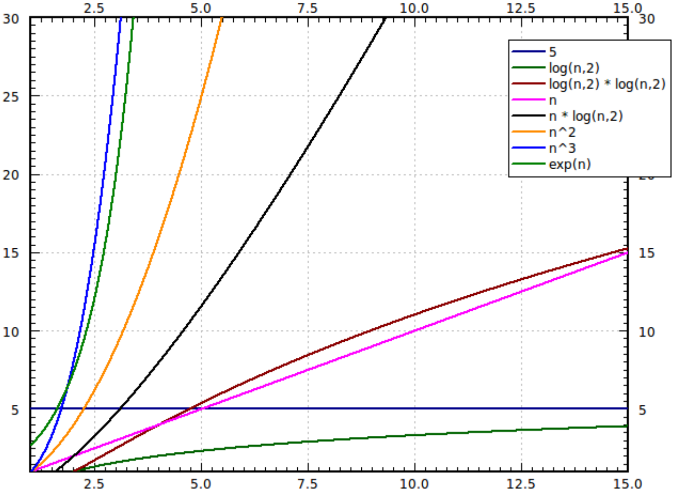

Figure: Typical Growth Rates

Typical Growth Rates

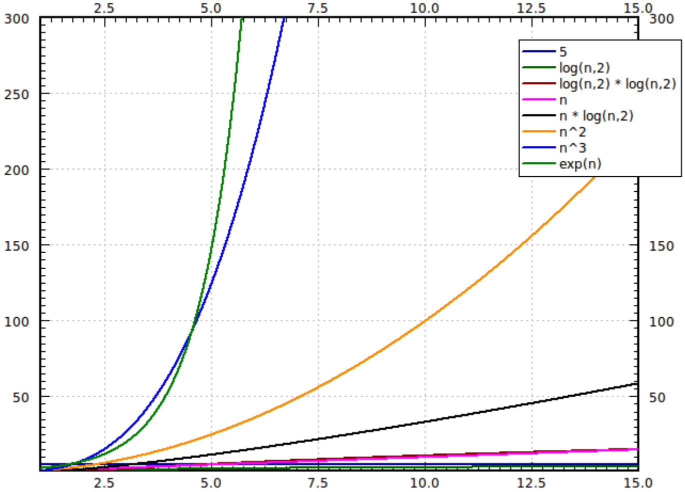

Figure: Typical Growth Rates

Typical Growth Rates

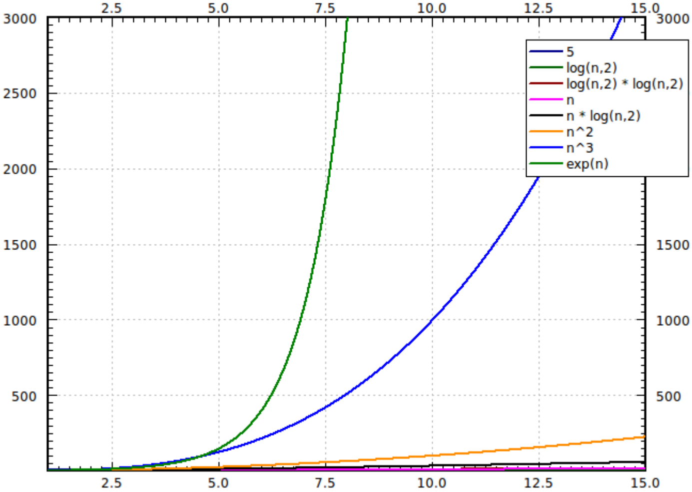

Figure: Typical Growth Rates

Typical Growth Rates

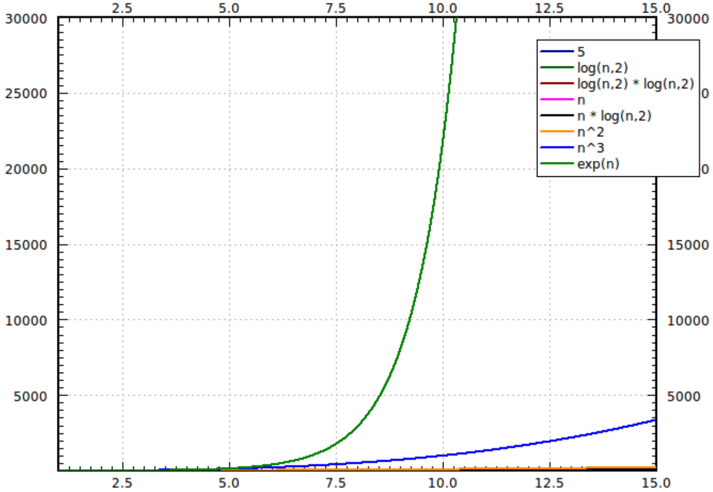

Figure: Typical Growth Rates

Exponential Complexity

NOTE

Algorithms with exponential complexities are practically intractable!

Empirical Verification: Ratio Analysis

- Is it possible to empirically verify if the running time of an algorithm is $O(f(n))$?

Shiv Nadar University

Empirical Verification: Ratio Analysis

- Is it possible to empirically verify if the running time of an algorithm is $O(f(n))$?
- Implement the algorithm and note down the running time $T(n)$ for different values of n.
- Now find the ratio $\frac{T(n)}{f(n)}$, for those different values on n.

Empirical Verification: Ratio Analysis

- Is it possible to empirically verify if the running time of an algorithm is $O(f(n))$?
- Implement the algorithm and note down the running time $T(n)$ for different values of n.
- Now find the ratio $\frac{T(n)}{f(n)}$, for those different values on n.
- $f(n)$ is a tight bound if this ratio converges to a positive constant

Empirical Verification: Ratio Analysis

- Is it possible to empirically verify if the running time of an algorithm is $O(f(n))$?
- Implement the algorithm and note down the running time $T(n)$ for different values of n.
- Now find the ratio $\frac{T(n)}{f(n)}$, for those different values on n.
- $f(n)$ is a tight bound if this ratio converges to a positive constant
- If the ratio converges to 0, then $f(n)$ is an over-estimate

Empirical Verification: Ratio Analysis

- Is it possible to empirically verify if the running time of an algorithm is $O(f(n))$?
- Implement the algorithm and note down the running time $T(n)$ for different values of n.
- Now find the ratio $\frac{T(n)}{f(n)}$, for those different values on n.
- $f(n)$ is a tight bound if this ratio converges to a positive constant
- If the ratio converges to 0, then $f(n)$ is an over-estimate
- $f(n)$ is an under-estimation, if this ratio diverges.

Simple Recurrence

```
long factorial(long n)
{
  if ( n < 2 ) return 1;
  return ( n * factorial(n-1) );
}
```

Shiv Nadar University

Simple Recurrence

```
long factorial(long n)
{
  if ( n < 2 ) return 1;
  return ( n * factorial(n-1) );
}
```

- How do we obtain $T(n)$ in this case?

Shiv Nadar University

Simple Recurrence

```
long factorial(long n)
{
  if ( n < 2 ) return 1;
  return ( n * factorial(n-1) );
}
```

- How do we obtain $T(n)$ in this case?

$$T(n) = \begin{cases} d & n \leq 1 \\ T(n-1) + c & n > 1 \end{cases}$$

- This is a simple recurrence equation.

Shiv Nadar University

Solving recurrences: Expansion Method

- Recurrences may be solved by a simple expansion process

Solving recurrences: Expansion Method

- Recurrences may be solved by a simple expansion process

$$T(n) = T(n-1) + c \qquad n > 1$$

Solving recurrences: Expansion Method

- Recurrences may be solved by a simple expansion process

$$
\begin{aligned}
T(n) &= T(n-1) + c & n > 1 \\
&= T(n-2) + c + c & n > 2 \\
&= T(n-2) + 2c & n > 2
\end{aligned}
$$

Solving recurrences: Expansion Method

- Recurrences may be solved by a simple expansion process

$$
\begin{aligned}
T(n) &= T(n-1) + c & n > 1 \\
&= T(n-2) + c + c & n > 2 \\
&= T(n-2) + 2c & n > 2 \\
&= T(n-3) + 3c & n > 3
\end{aligned}
$$

Solving recurrences: Expansion Method

- Recurrences may be solved by a simple expansion process

$$
\begin{aligned}
T(n) &= T(n-1) + c & n > 1 \\
&= T(n-2) + c + c & n > 2 \\
&= T(n-2) + 2c & n > 2 \\
&= T(n-3) + 3c & n > 3 \\
&\vdots
\end{aligned}
$$

Shiv Nadar University

Solving recurrences: Expansion Method

- Recurrences may be solved by a simple expansion process

$$
\begin{aligned}
T(n) &= T(n-1) + c && n > 1 \\
&= T(n-2) + c + c && n > 2 \\
&= T(n-2) + 2c && n > 2 \\
&= T(n-3) + 3c && n > 3 \\
&\vdots \\
&= T(n-i) + ic && n > i
\end{aligned}
$$

Solving recurrences: Expansion Method

- Recurrences may be solved by a simple expansion process

$$
\begin{aligned}
T(n) &= T(n-1) + c && n > 1 \\
&= T(n-2) + c + c && n > 2 \\
&= T(n-2) + 2c && n > 2 \\
&= T(n-3) + 3c && n > 3 \\
&\ \ \vdots \\
&= T(n-i) + ic && n > i
\end{aligned}
$$

- The objective is to represent $T(n)$ directly in terms of the base case $T(1)$. This is achieved when $i = n - 1$

$$T(n) = T(n - (n-1)) + (n-1)c$$

Solving recurrences: Expansion Method

- Recurrences may be solved by a simple expansion process

$$
\begin{aligned}
T(n) &= T(n-1) + c && n > 1 \\
&= T(n-2) + c + c && n > 2 \\
&= T(n-2) + 2c && n > 2 \\
&= T(n-3) + 3c && n > 3 \\
&\;\;\vdots \\
&= T(n-i) + ic && n > i
\end{aligned}
$$

- The objective is to represent $T(n)$ directly in terms of the base case $T(1)$. This is achieved when $i = n - 1$

$$
\begin{aligned}
T(n) &= T(n-(n-1)) + (n-1)c \\
&= T(1) + (n-1)c
\end{aligned}
$$

Shiv Nadar University

Solving recurrences: Expansion Method

- Recurrences may be solved by a simple expansion process

$$
\begin{aligned}
T(n) &= T(n-1) + c && n > 1 \\
&= T(n-2) + c + c && n > 2 \\
&= T(n-2) + 2c && n > 2 \\
&= T(n-3) + 3c && n > 3 \\
&\vdots \\
&= T(n-i) + ic && n > i
\end{aligned}
$$

- The objective is to represent $T(n)$ directly in terms of the base case $T(1)$. This is achieved when $i = n - 1$

$$
\begin{aligned}
T(n) &= T(n-(n-1)) + (n-1)c \\
&= T(1) + (n-1)c \\
&= d + (n-1)c
\end{aligned}
$$

Solving recurrences: Expansion Method

- Recurrences may be solved by a simple expansion process

$$
\begin{aligned}
T(n) &= T(n-1) + c & n > 1 \\
&= T(n-2) + c + c & n > 2 \\
&= T(n-2) + 2c & n > 2 \\
&= T(n-3) + 3c & n > 3 \\
&\ \ \vdots & \\
&= T(n-i) + ic & n > i
\end{aligned}
$$

- The objective is to represent $T(n)$ directly in terms of the base case $T(1)$. This is achieved when $i = n - 1$

$$
\begin{aligned}
T(n) &= T(n-(n-1)) + (n-1)c \\
&= T(1) + (n-1)c \\
&= d + (n-1)c
\end{aligned}
$$

- It is obvious that this $T(n)$ is $\Theta(n)$

SHIV NADAR UNIVERSITY

Solving Recurrences: Telescopic Sum

$$T(n) = T(n-1) + c$$

Shiv Nadar University

Solving Recurrences: Telescopic Sum

$$
\begin{aligned}
T(n) &= T(n-1) + c \\
T(n-1) &= T(n-2) + c
\end{aligned}
$$

Shiv Nadar University

Solving Recurrences: Telescopic Sum

$$
\begin{aligned}
T(n) &= T(n-1) + c \\
T(n-1) &= T(n-2) + c \\
&\vdots \\
T(2) &= T(1) + c
\end{aligned}
$$

Solving Recurrences: Telescopic Sum

$$
\begin{aligned}
T(n) &= T(n-1) + c \\
T(n-1) &= T(n-2) + c \\
&\vdots \\
T(2) &= T(1) + c \\
\hline
\end{aligned}
$$

Shiv Nadar University

Solving Recurrences: Telescopic Sum

$$
\begin{aligned}
T(n) &= T(n-1) + c \\
T(n-1) &= T(n-2) + c \\
&\vdots \\
T(2) &= T(1) + c
\end{aligned}
$$

$$
\begin{aligned}
T(n) + T(n-1) + T(n-2) + \cdots + T(2) \; = \\
T(n-1) + T(n-2) + \cdots + T(2) + T(1) + (n-1)c
\end{aligned}
$$

Shiv Nadar University

Solving Recurrences: Telescopic Sum

$$\begin{aligned} T(n) &= T(n-1) + c \\ T(n-1) &= T(n-2) + c \\ &\vdots \\ T(2) &= T(1) + c \end{aligned}$$

$$T(n) + T(n-1) + T(n-2) + \cdots + T(2) = \\ T(n-1) + T(n-2) + \cdots + T(2) + T(1) + (n-1)c$$

$$T(n) + \cancel{T(n-1)} + \cancel{T(n-2)} + \cdots + \cancel{T(2)} = \\ \cancel{T(n-1)} + \cancel{T(n-2)} + \cdots + \cancel{T(2)} + T(1) + (n-1)c$$

Shiv Nadar University

Solving Recurrences: Telescopic Sum

$$
\begin{aligned}
T(n) &= T(n-1) + c \\
T(n-1) &= T(n-2) + c \\
&\vdots \\
T(2) &= T(1) + c
\end{aligned}
$$

$$
T(n) + T(n-1) + T(n-2) + \cdots + T(2) = T(n-1) + T(n-2) + \cdots + T(2) + T(1) + (n-1)c
$$

$$
T(n) + \cancel{T(n-1)} + \cancel{T(n-2)} + \cdots + \cancel{T(2)} = \cancel{T(n-1)} + \cancel{T(n-2)} + \cdots + \cancel{T(2)} + T(1) + (n-1)c
$$

$$
T(n) = T(1) + (n-1)c
$$

Shiv Nadar University

Solving Recurrences: Telescopic Sum

$$
\begin{aligned}
T(n) &= T(n-1) + c \\
T(n-1) &= T(n-2) + c \\
&\vdots \\
T(2) &= T(1) + c
\end{aligned}
$$

$$
T(n) + T(n-1) + T(n-2) + \cdots + T(2) = T(n-1) + T(n-2) + \cdots + T(2) + T(1) + (n-1)c
$$

$$
T(n) + \cancel{T(n-1)} + \cancel{T(n-2)} + \cdots + \cancel{T(2)} = \cancel{T(n-1)} + \cancel{T(n-2)} + \cdots + \cancel{T(2)} + T(1) + (n-1)c
$$

$$
\begin{aligned}
T(n) &= T(1) + (n-1)c \\
&= d + (n-1)c
\end{aligned}
$$

Solving Recurrences: Telescopic Sum

$$
\begin{aligned}
T(n) &= T(n-1) + c \\
T(n-1) &= T(n-2) + c \\
&\vdots \\
T(2) &= T(1) + c
\end{aligned}
$$

$$
\begin{aligned}
T(n) + T(n-1) + T(n-2) + \cdots + T(2) \; = \\
T(n-1) + T(n-2) + \cdots + T(2) + T(1) + (n-1)c
\end{aligned}
$$

$$
\begin{aligned}
T(n) + \cancel{T(n-1)} + \cancel{T(n-2)} + \cdots + \cancel{T(2)} \; = \\
\cancel{T(n-1)} + \cancel{T(n-2)} + \cdots + \cancel{T(2)} + T(1) + (n-1)c
\end{aligned}
$$

$$
\begin{aligned}
T(n) &= T(1) + (n-1)c \\
&= d + (n-1)c
\end{aligned}
$$

- It is obvious that this $T(n)$ is $\Theta(n)$

Shiv Nadar University

Another Example: Insertion Sort

```
void IsortList(List l)
{
  Position p = Begin(l);

  if ( p == End(l) ) return;

  ElementType head = Retrieve(p, l);

  Delete(p, l);

  IsortList(l);

  InsertOrder(head, l);

  return;
}
```

Shiv Nadar University

Another Example: Insertion Sort

```
void InsertOrder(ElementType ele, List l)
{
  Position p = Begin(l);

  while ( p != End(l) ) {
    if ( ele <= Retrieve(p, l) ) break;
    else p = Advance(p);
  }

  Insert(ele, p, l);

  return;
}
```

Insertion Sort

- The recurrence equation may be obtained as:

$$T(n) = \begin{cases} d & n \leq 1 \\ T(n-1) + cn & n > 1 \end{cases}$$

SHIV NADAR UNIVERSITY

Insertion Sort

- The recurrence equation may be obtained as:

$$T(n) = \begin{cases} d & n \leq 1 \\ T(n-1) + cn & n > 1 \end{cases}$$

- Let us solve this by telescopic sum method:

$$T(n) = T(n-1) + cn$$

Insertion Sort

- The recurrence equation may be obtained as:

$$T(n) = \begin{cases} d & n \leq 1 \\ T(n-1) + cn & n > 1 \end{cases}$$

- Let us solve this by telescopic sum method:

$$\begin{aligned} T(n) &= T(n-1) + cn \\ T(n-1) &= T(n-2) + c(n-1) \end{aligned}$$

Shiv Nadar University

Insertion Sort

- The recurrence equation may be obtained as:

$$T(n) = \begin{cases} d & n \leq 1 \\ T(n-1) + cn & n > 1 \end{cases}$$

- Let us solve this by telescopic sum method:

$$\begin{aligned} T(n) &= T(n-1) + cn \\ T(n-1) &= T(n-2) + c(n-1) \\ &\vdots \\ T(2) &= T(1) + 2c \end{aligned}$$

Shiv Nadar University

Insertion Sort

- The recurrence equation may be obtained as:

$$T(n) = \begin{cases} d & n \leq 1 \\ T(n-1) + cn & n > 1 \end{cases}$$

- Let us solve this by telescopic sum method:

$$\begin{aligned} T(n) &= T(n-1) + cn \\ T(n-1) &= T(n-2) + c(n-1) \\ &\vdots \\ T(2) &= T(1) + 2c \end{aligned}$$

Shiv Nadar University

Insertion Sort

- The recurrence equation may be obtained as:

$$T(n) = \begin{cases} d & n \leq 1 \\ T(n-1) + cn & n > 1 \end{cases}$$

- Let us solve this by telescopic sum method:

$$\begin{aligned} T(n) &= T(n-1) + cn \\ T(n-1) &= T(n-2) + c(n-1) \\ &\vdots \\ T(2) &= T(1) + 2c \end{aligned}$$

$$T(n) = T(1) + c(2 + 3 + \cdots + n)$$

Shiv Nadar University

Insertion Sort

- The recurrence equation may be obtained as:

$$T(n) = \begin{cases} d & n \leq 1 \\ T(n-1) + cn & n > 1 \end{cases}$$

- Let us solve this by telescopic sum method:

$$\begin{aligned} T(n) &= T(n-1) + cn \\ T(n-1) &= T(n-2) + c(n-1) \\ &\vdots \\ T(2) &= T(1) + 2c \end{aligned}$$

$$\begin{aligned} T(n) &= T(1) + c(2 + 3 + \cdots + n) \\ &= T(1) + c(\tfrac{n(n+1)}{2} - 1) \end{aligned}$$

Shiv Nadar University

- The recurrence equation may be obtained as:

$$T(n) = \begin{cases} d & n \leq 1 \\ T(n-1) + cn & n > 1 \end{cases}$$

- Let us solve this by telescopic sum method:

$$\begin{aligned} T(n) &= T(n-1) + cn \\ T(n-1) &= T(n-2) + c(n-1) \\ &\vdots \\ T(2) &= T(1) + 2c \end{aligned}$$

$$\begin{aligned} T(n) &= T(1) + c(2 + 3 + \cdots + n) \\ &= T(1) + c(\tfrac{n(n+1)}{2} - 1) \end{aligned}$$

- It is obvious that this $T(n)$ is $\Theta(n^2)$

Shiv Nadar University

Yet Another Example

- Let us consider the following recurrence equation:

$$T(n) = \begin{cases} d & n \leq 1 \\ 2T(n/2) + n & n > 1 \end{cases}$$

SHIV NADAR UNIVERSITY

Yet Another Example

- Let us consider the following recurrence equation:

$$T(n) = \begin{cases} d & n \leq 1 \\ 2T(n/2) + n & n > 1 \end{cases}$$

- This arises, for example, in the divide and conquer algorithms such as mergesort
- The input is divided into two halves and both the halves are solved independently
- Then both the solutions are merged in linear time to get the overall solution

Yet Another Example — Expansion

- Let's first try the expansion method

$$T(n) = 2T(n/2) + n \qquad\qquad n > 1$$

Yet Another Example — Expansion

- Let's first try the expansion method

$$
\begin{aligned}
T(n) &= 2T(n/2) + n & n > 1 \\
&= 2[2T(n/4) + n/2] + n & n > 2 \\
&= 2^2 T(n/2^2) + 2n & n > 2
\end{aligned}
$$

SHIV NADAR UNIVERSITY

Yet Another Example — Expansion

- Let's first try the expansion method

$$
\begin{aligned}
T(n) &= 2T(n/2) + n & n > 1 \\
&= 2[2T(n/4) + n/2] + n & n > 2 \\
&= 2^2 T(n/2^2) + 2n & n > 2 \\
&= 2^2[2T(n/2^3) + n/2^2] + 2n & n > 2^2 \\
&= 2^3 T(n/2^3) + 3n & n > 2^2
\end{aligned}
$$

Shiv Nadar University

Yet Another Example — Expansion

- Let's first try the expansion method

$$
\begin{aligned}
T(n) &= 2T(n/2) + n && n > 1 \\
&= 2[2T(n/4) + n/2] + n && n > 2 \\
&= 2^2 T(n/2^2) + 2n && n > 2 \\
&= 2^2[2T(n/2^3) + n/2^2] + 2n && n > 2^2 \\
&= 2^3 T(n/2^3) + 3n && n > 2^2 \\
&\vdots
\end{aligned}
$$

Shiv Nadar University

Yet Another Example — Expansion

- Let's first try the expansion method

$$
\begin{aligned}
T(n) &= 2T(n/2) + n && n > 1 \\
&= 2[2T(n/4) + n/2] + n && n > 2 \\
&= 2^2 T(n/2^2) + 2n && n > 2 \\
&= 2^2[2T(n/2^3) + n/2^2] + 2n && n > 2^2 \\
&= 2^3 T(n/2^3) + 3n && n > 2^2 \\
&\;\;\vdots \\
&= 2^i T(n/2^i) + in && n > 2^{i-1}
\end{aligned}
$$

Shiv Nadar University

Yet Another Example — Expansion

- Let's first try the expansion method

$$
\begin{aligned}
T(n) &= 2T(n/2) + n && n > 1 \\
&= 2[2T(n/4) + n/2] + n && n > 2 \\
&= 2^2 T(n/2^2) + 2n && n > 2 \\
&= 2^2[2T(n/2^3) + n/2^2] + 2n && n > 2^2 \\
&= 2^3 T(n/2^3) + 3n && n > 2^2 \\
&\quad \vdots \\
&= 2^i T(n/2^i) + in && n > 2^{i-1}
\end{aligned}
$$

- When $i = \log n$

$$T(n) = nT(1) + n \log n$$

Yet Another Example — Expansion

- Let's first try the expansion method

$$
\begin{aligned}
T(n) &= 2T(n/2) + n && n > 1 \\
&= 2[2T(n/4) + n/2] + n && n > 2 \\
&= 2^2 T(n/2^2) + 2n && n > 2 \\
&= 2^2[2T(n/2^3) + n/2^2] + 2n && n > 2^2 \\
&= 2^3 T(n/2^3) + 3n && n > 2^2 \\
&\vdots \\
&= 2^i T(n/2^i) + in && n > 2^{i-1}
\end{aligned}
$$

- When $i = \log n$

$$T(n) = nT(1) + n \log n$$

- $nO(1) + O(n \log n)$
- $O(n) + O(n \log n)$
- $O(n \log n)$

Shiv Nadar University

Yet Another Example — Telescoping

$$T(n) = 2T(n/2) + n$$

Shiv Nadar University

Yet Another Example — Telescoping

$$T(n) = 2T(n/2) + n$$

$$\frac{T(n)}{n} = \frac{2T(n/2)}{n} + 1$$

Shiv Nadar University

Yet Another Example — Telescoping

$$T(n) = 2T(n/2) + n$$

$$\frac{T(n)}{n} = \frac{2T(n/2)}{n} + 1$$

$$\frac{T(n)}{n} = \frac{T(n/2)}{n/2} + 1$$

Shiv Nadar University

Yet Another Example — Telescoping

$$T(n) = 2T(n/2) + n$$

$$\frac{T(n)}{n} = \frac{2T(n/2)}{n} + 1$$

$$\frac{T(n)}{n} = \frac{T(n/2)}{n/2} + 1$$

$$\frac{T(n/2)}{n/2} = \frac{T(n/4)}{n/4} + 1$$
$$\frac{T(n/4)}{n/4} = \frac{T(n/8)}{n/8} + 1$$
$$\vdots$$
$$\frac{T(2)}{2} = \frac{T(1)}{1} + 1$$

SHIV NADAR UNIVERSITY

Yet Another Example — Telescoping

$$T(n) = 2T(n/2) + n$$

$$\frac{T(n)}{n} = \frac{2T(n/2)}{n} + 1$$

$$\frac{T(n)}{n} = \frac{T(n/2)}{n/2} + 1$$

$$\frac{T(n/2)}{n/2} = \frac{T(n/4)}{n/4} + 1$$
$$\frac{T(n/4)}{n/4} = \frac{T(n/8)}{n/8} + 1$$
$$\vdots$$
$$\frac{T(2)}{2} = \frac{T(1)}{1} + 1$$

Shiv Nadar University

Yet Another Example — Telescoping

$$T(n) = 2T(n/2) + n$$

$$\frac{T(n)}{n} = \frac{2T(n/2)}{n} + 1$$

$$\frac{T(n)}{n} = \frac{T(n/2)}{n/2} + 1$$

$$\frac{T(n/2)}{n/2} = \frac{T(n/4)}{n/4} + 1$$
$$\frac{T(n/4)}{n/4} = \frac{T(n/8)}{n/8} + 1$$
$$\vdots$$
$$\frac{T(2)}{2} = \frac{T(1)}{1} + 1$$

$$\frac{T(n)}{n} = T(1) + \log n$$
$$T(n) = cn + n \log n$$

Shiv Nadar University

Yet Another Example — Telescoping

$$T(n) = 2T(n/2) + n$$

$$\frac{T(n)}{n} = \frac{2T(n/2)}{n} + 1$$

$$\frac{T(n)}{n} = \frac{T(n/2)}{n/2} + 1$$

$$\begin{aligned}
\frac{T(n/2)}{n/2} &= \frac{T(n/4)}{n/4} + 1 \\
\frac{T(n/4)}{n/4} &= \frac{T(n/8)}{n/8} + 1 \\
&\vdots \\
\frac{T(2)}{2} &= \frac{T(1)}{1} + 1 \\
\hline
\frac{T(n)}{n} &= T(1) + \log n \\
T(n) &= cn + n \log n
\end{aligned}$$

$O(n \log n)$

SHIV NADAR UNIVERSITY

- Let us consider the following recurrence equation (where $T(1)$ is a constant):

$$T(n) = \frac{2}{n}\left(\sum_{j=0}^{n-1} T(j)\right) + cn$$

Slightly Complicated Example

Slightly Complicated example:

$$T(n) = \frac{2}{n}\left(\sum_{j=0}^{n-1} T(j)\right) + cn$$

$$n\,T(n) = 2\left(\sum_{j=0}^{n-1} T(j)\right) + cn^2$$

$$(n-1)\,T(n-1) = 2\left(\sum_{j=0}^{n-2} T(j)\right) + c(n-1)^2$$

Subtract

$$n\,T(n) - (n-1)\,T(n-1) = 2\,T(n-1) + 2cn - c$$

$$n\,T(n) = (n+1)\,T(n-1) + 2cn$$

Now, to telescope divide by $n(n+1)$

$$\frac{T(n)}{n+1} = \frac{T(n-1)}{n} + \frac{2c}{n+1}$$

Slightly Complicated Example

$$\frac{T(n)}{n+1} = \frac{T(n-1)}{n} + \frac{2c}{n+1}$$

$$\frac{T(n-1)}{n} = \frac{T(n-2)}{n-1} + \frac{2c}{n}$$

$$\frac{T(n-2)}{n-1} = \frac{T(n-3)}{n-2} + \frac{2c}{n-1}$$

$$\vdots$$

$$\frac{T(2)}{3} = \frac{T(1)}{2} + \frac{2c}{3}$$

$$\frac{T(n)}{n+1} = \frac{T(1)}{2} + 2c \underbrace{\sum_{i=3}^{n+1} 1/i}_{O(\log n)}$$

$$\frac{T(n)}{n+1} = O(\log n)$$

$$T(n) = O(n \log n)$$

Master Theorem

- Let $T(n) = aT(n/b) + f(n)$, where $a \geq 1$ and $b > 1$

Shiv Nadar University

Master Theorem

- Let $T(n) = aT(n/b) + f(n)$, where $a \geq 1$ and $b > 1$
- If $f(n)$ is sub-linear time, then $T(n)$ is $\Theta(n^{\log_b a})$

Shiv Nadar University

Master Theorem

- Let $T(n) = aT(n/b) + f(n)$, where $a \geq 1$ and $b > 1$
- If $f(n)$ is sub-linear time, then $T(n)$ is $\Theta(n^{\log_b a})$
- If $f(n)$ is linear time, then $T(n)$ is $\Theta(n^{\log_b a} \log n)$

Shiv Nadar University

- Let $T(n) = aT(n/b) + f(n)$, where $a \geq 1$ and $b > 1$
- If $f(n)$ is sub-linear time, then $T(n)$ is $\Theta(n^{\log_b a})$
- If $f(n)$ is linear time, then $T(n)$ is $\Theta(n^{\log_b a} \log n)$
- If $f(n)$ is more than linear time, then $T(n)$ is $\Theta(f(n))$

A algorithm whose execution time, f(n), grows slower than the size of the problem, n, but only gives an approximate or probably correct answer.

Summary

- Time / Space complexity of an algorithm are expressed in notations such as Big-Oh and Big-Theta
- These notations bring out the growth rate of time / space wrt the size of the input
- These notations enable us to avoid exact calculations of number of "basic steps" or memory space required — overall growth rate can be estimated based on growth rates of components
- Complexity of recursive algorithms can be analyzed through the corresponding recurrence equations
- Recurrences may be solved by expansion method, telescopic sum method, or by solving corresponding characteristic equations
- It is also possible to perform empirical ratio analysis to determine / verify time complexity of an algorithm