## Maximum Subarray

```
FIND-MAX-CROSSING-SUBARRAY(A, low, mid, high)
left-sum = -∞
sum = 0
for i = mid downto low
    sum = sum + A[i]
    if sum > left-sum
        left-sum = sum
        max-left = i
right-sum = -∞
sum = 0
for j = mid + 1 to high
    sum = sum + A[j]
    if sum > right-sum
        right-sum = sum
        max-right = j
return (max-left, max-right, left-sum + right-sum)
```

```
FIND-MAXIMUM-SUBARRAY(A, low, high)
if high == low
    return (low, high, A[low])          // base case: only one element
else mid = ⌊(low + high)/2⌋
    (left-low, left-high, left-sum) =
        FIND-MAXIMUM-SUBARRAY(A, low, mid)
    (right-low, right-high, right-sum) =
        FIND-MAXIMUM-SUBARRAY(A, mid + 1, high)
    (cross-low, cross-high, cross-sum) =
        FIND-MAX-CROSSING-SUBARRAY(A, low, mid, high)
    if left-sum ≥ right-sum and left-sum ≥ cross-sum
        return (left-low, left-high, left-sum)
    elseif right-sum ≥ left-sum and right-sum ≥ cross-sum
        return (right-low, right-high, right-sum)
    else return (cross-low, cross-high, cross-sum)
```

## Merge Sort

```
MERGE-SORT(A, p, r)
if p < r
    q = ⌊(p + r)/2⌋
    MERGE-SORT(A, p, q)
    MERGE-SORT(A, q + 1, r)
    MERGE(A, p, q, r)
```

```
MERGE(A, p, q, r)
n1 = q − p + 1
n2 = r − q
let L[1 . . n1 + 1] and R[1 . . n2 + 1] be new arrays
for i = 1 to n1
    L[i] = A[p + i − 1]
for j = 1 to n2
    R[j] = A[q + j]
L[n1 + 1] = ∞
R[n2 + 1] = ∞
i = 1
j = 1
for k = p to r
    if L[i] ≤ R[j]
        A[k] = L[i]
        i = i + 1
    else A[k] = R[j]
        j = j + 1
```

## Quick Sort

```
QUICKSORT(A, p, r)
if p < r
    q = PARTITION(A, p, r)
    QUICKSORT(A, p, q − 1)
    QUICKSORT(A, q + 1, r)
```

```
PARTITION(A, p, r)
x = A[r]
i = p − 1
for j = p to r − 1
    if A[j] ≤ x
        i = i + 1
        exchange A[i] with A[j]
exchange A[i + 1] with A[r]
return i + 1
```

## Heap Sort

```
HEAPSORT(A)
BUILD-MAX-HEAP(A)
for i = A.length downto 2
    exchange A[1] with A[i]
    A.heap-size = A.heap-size - 1
    MAX-HEAPIFY(A, 1)
```

```
MAX-HEAPIFY(A, i)
l = LEFT(i)
r = RIGHT(i)
if l ≤ A.heap-size and A[l] > A[i]
    largest = l
else largest = i
if r ≤ A.heap-size and A[r] > A[largest]
    largest = r
if largest ≠ i
    exchange A[i] with A[largest]
    MAX-HEAPIFY(A, largest)
```

```
BUILD-MAX-HEAP(A)
A.heap-size = A.length
for i = ⌊A.length/2⌋ downto 1
    MAX-HEAPIFY(A, i)
```

**********

- A combination of iterative and incremental process models with focus on process adaptability and customer satisfaction by rapid delivery of working software product. - Agile Methods break the product into small incremental builds. These builds are provided in iterations.

- Each iteration typically lasts from about one to three weeks.

- Every iteration involves cross functional teams working simultaneously on various areas like –

  Planning

  Requirements Analysis

  Design

  Coding

  Unit Testing and

  Acceptance Testing.

- At the end of the iteration, a working product is displayed to the customer and important stakeholders.

- Agile model believes that every project needs to be handled differently and the existing methods need to be tailored to best suit the project requirements. In Agile, the tasks are divided to time boxes (small time frames) to deliver specific features for a release.

- Iterative approach is taken and working software build is delivered after each iteration. Each build is incremental in terms of features; the final build holds all the features required by the customer.

- Agile modeling (AM) is a methodology for modeling and documenting software systems based on best practices.

- It is a collection of values and principles, that can be applied on an (agile) software development project.

- This methodology is more flexible than traditional modeling methods, making it a better fit in a fast changing environment.

- It is part of the agile software development tool kit.

- Agile modeling is a supplement to other agile development methodologies such as Scrum, extreme programming (XP), and Rational Unified Process (RUP).

- It is explicitly included as part of the disciplined agile delivery (DAD) framework.

Cons:

- In case of some software deliverables, especially the large ones, it is difficult to assess the effort required at the beginning of the software development life cycle.

- There is lack of emphasis on necessary designing and documentation.

- The project can easily get taken off track if the customer representative is not clear what final outcome that they want.

- Only senior programmers are capable of taking the kind of decisions required during the development process.

- harder for new members to get up to speed

- difficult to measure progress than it is in Waterfall

- Agile demands more time and energy from everyone - interaction is demanding

- Projects can become ever-lasting because there's no clear end

- lacks overall design, both from a UX and architecture point of view

- difficult to implement when clients follow rigid processes or methods.

- the risk that the long-term vision will be lost does exist.

- It required expert persons to take important decisions in the meeting.

- Cost more as compare to waterfall or interactive model.

Pros:

- Customer satisfaction by rapid, continuous delivery of useful software.

- People and interactions are emphasized rather than process and tools. Customers, developers and testers constantly interact with each other.

- Working software is delivered frequently (weeks rather than months).

- Face-to-face conversation is the best form of communication.

- Close, daily cooperation between business people and developers.

- Continuous attention to technical excellence and good design.

- Regular adaptation to changing circumstances.

- Even late changes in requirements are welcomed.

- You can deploy software quicker - fast value

- You waste fewer resources because you always work on up-to-date tasks

- Respond faster to any change, and faster turnaround times

- Detect and fix issues and defects faster

- Less time on bureaucracy and busywork

- There's a big community of Agile practitioners with whom you can share knowledge

- Immediate feedback (which also improves team morale)

- Developers can improve their skills based on QA feedback

- You can experiment and test ideas because its costs are low

- More flexibility;

- More productivity;
- More transparency;
- Faster implementation of solutions;
- Waste reduction thanks to the minimization of the resources;
- A lighter/less complicated structure;
- Excellent project control;
- Greater attention to specific customer needs;
- Increased collaboration frequency and feedback.
- Customers can change or add requirements at any stage.

- A particular kind of mathematically rigorous techniques for the specification, development and verification of software and hardware systems.

- The use of formal methods for software and hardware design is motivated by the expectation that, as in other engineering disciplines, performing appropriate mathematical analysis can contribute to the reliability and robustness of a design.

- Formal methods are best described as the application of a fairly broad variety of theoretical computer science fundamentals, in particular logic calculi, formal languages, automata theory, discrete event dynamic system and program semantics, but also type systems and algebraic data types to problems in software and hardware specification and verification.[3]

- concerned with the application of a mathematical technique to design and implement the software.

- This model lays the foundation for developing a complex system and supporting the program development.

- Formal methods comprise formal specification using mathematics to specify the desired properties of the system.

- Formal specification is expressed in a language whose syntax and semantics are formally defined. This language comprises a syntax that defines specific notation used for specification representation; semantic, which uses objects to describe the system; and a set of relations, which uses rules to indicate the objects for satisfying the specification.

- Formal method comprises two approaches, namely, property based and model-based.

- The property-based specification describes the operations performed on the system. In addition, it describes the relationship that exists among these operations. A property-based specification consists of two parts: signatures, which determine the syntax of operations and an equation, which defines the semantics of the operations through a set of equations known as axioms.

- The model-based specification utilizes the tools of set theory, function theory, and logic to develop an abstract model of the system. In addition, it specifies the operations performed on the abstract model. The model thus developed is of a high level and idealized. A model-based specification comprises a definition of the set of states of the system and definitions of the legal operations performed on the system to indicate how these legal operations change the current state.

Pros:

- Formal development and formal verification may be used to produce a program in a more formal manner.

- Formal methods may be used to give a description of the system to be developed, at whatever level(s) of detail desired.

- This formal description can be used to guide further development activities (see following sections);

- additionally, it can be used to verify that the requirements for the system being developed have been completely and accurately specified. or formalising system requirements by expressing them in a formal language with a precise and unambiguously defined syntax and semantics.

- Formal methods are applied in different areas of hardware and software, including routers, Ethernet switches, routing protocols, security applications, and operating system microkernels such as seL4.

- Formal verification has been frequently used in hardware by most of the well-known hardware vendors, such as IBM, Intel, and AMD. There are many areas of hardware, where Intel have used FMs to verify the working of the products, such as parameterized verification of cache coherent protocol,[18] Intel Core i7 processor execution engine validation

- Discovers ambiguity, incompleteness, and inconsistency in the software.

- Offers defect-free software.

- Incrementally grows in effective solution after each iteration.

- This model does not involve high complexity rate.

- Formal specification language semantics verify self-consistency.

- Abstractness. If done well, one can verify a variety of problems into formal methods.

- Robustness. It can mathematically prove certain properties of a model. This is significantly better than standard unit-testing and other methods to understand executions of a model.

- provides insights into and an understanding of the software requirements and software design

-

Cons:

- Theorem provers may be used to undertake fully formal machine-checked proofs. This can be very expensive and is only practically worthwhile if the cost of mistakes is extremely high (e.g., in critical parts of microprocessor design).

- Formal methods may be used to give a description of the system to be developed, at whatever level(s) of detail desired. This formal description can be used to guide further development activities (see following sections);

- Formal verification using a software tool to prove properties of a formal specification, or that a formal model of a system implementation satisfies its specification. Once a formal specification has been developed, the specification may be used as the basis for proving properties of the specification (and hopefully by inference the developed system).

- Time consuming and expensive.

- Difficult to use this model as a communication mechanism for non technical personnel.

- Extensive training is required since only few developers have the essential knowledge to implement this model.

- little assistance in actually arriving at that description.

- designers often find it difficult to apply these methods as they give them little assistance in working with anything other than complete and closed descriptions

- tough to apply (traditional) formal methods in real-world giant applications

# GROUP 2 – SURVEY (DR. CHEN).

Q4. Explain the following software process models and point out their pros and cons:

(a) Formal Methods
(b) Agile development model.

(a) Formal Methods – mathematical specification of software systems.

- Has 2 steps:
  (i) Write requirements in formal language.
  (ii) Translate requirements into source code.
- Best suited for mission critical projects. Very high s/w quality (if this is used).

Pros:

- efficient than natural languages. (i.e. not ambiguous).
- can identify errors before they reach code.
- can be submitted to automatic code generators.

Cons:

- produces specifications that cannot be interpreted by customer.
- Requires extensive training to be properly used.
- It is time-consuming.
- usually can be used for mission critical projects.

(b) Agile Development Method – an incremental approach.

- Similar to incremental model, but has short term plans
- Uses small increments which are of about 2-4 weeks.
- Best suited for projects where requirements change rapidly during development process.

Pros:

- Emphasizes on customer collaboration over contract.
- Often interaction b/w customers and users.
- Respond to changes ~~[illegible]~~ faster.
- Stresses on working s/w rather than document.

Cons:

- team may not interact with other teams/members.
- maintaining simplicity needs extra efforts.
- Difficult for process-driven organizatn to change defined working model.

a) Formal Method and
- mathematical techniques are used for designing and implementation of the software.
- help in remove any problems by provide the needed mechanisms.
- The property based model specifies the operations/functions which the system will perform
- the model-based approach ,with the help of mathematical theory like set and function theory and logic creates an abstract model of the system which is to be developed.

- Pros:
  1.formal method can help identify any inconsistency or incompleteness in the system that is being developed.
  2.Software which is developed using formal method is free from defects.
  3.Complexity rate of formal method is generally low.

- Cons:
  1.Formal methods are expensive and take a lot of time.
  2.Using formal method as a communication means is difficult to understand without technical knowledge.
  3.Proper Training is required for applying formal method.

b) Agile development model
- an incremental software development method where the development and testing is carried out throughout the whole development and development and testing is done simultaneously
- The whole development process is divided into smaller individual models on which the development team works.
- It has 4 core values :
  Interaction between the team and client is given more importance.
  working software is more important than documentation.
  client has to provide participation.
  Less importance is given to the software development plan and is more focused on accommodating new changes to the software as and when required.
- Pros
  Agile development method has more flexibility when compared to other development methods.
  Changes to the system can be accommodated easily.
  Development of the system takes less time.
  There is regular communication between the team working on the project which can help the team perform better.
  Agile is a great model to apply for development when the client does not have all the requirements or the requirements might change.
- Cons:
  As it is flexible,the customer might keep on asking for changes ,which makes agile development method hard to predict the end result of the system and Project might go in some other directions then what was first decided.
  Documentation is not given much priority in this method.
  Constant communication between the developers,testers and the client is needed and if anyone of them are lacking participation then it effects the software which is being developed.

1. **CS 5329 Algorithm Design and Analysis**
   { *From Dr. Hwang* }

   For the recurrence formula of Merge Sort and Max-subarray,

$$T(n) = 2T(n/2) + n \quad \text{for } n>1$$

$$T(1) = 1$$

   (a) Derive the total running time T(n) by using Recursion Tree method, and
   (b) Derive the total running time T(n) by using Iteration method.

(a) T(n) by Recursion Tree method.:

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

Recursion Tree →

So, $\frac{n}{2^x} \cong 1 \Rightarrow n \cong 2^x$

i.e. $n = 2^x$ ( Take log on both sides).

$\log n = \log 2^x$

$\log n = x \log 2$

$x = \frac{\log n}{\log 2} \rightarrow x = \log_2 n.$

Total runing time $(T(n)) = Cn(x)$

$= Cn(\lg_2 n)$

$$\boxed{T(n) = O(n \lg n).}$$

(b) $T(n)$ by Iteration method:

$$T(n) = 2T(n/2) + n. \quad - ①$$

→ put

So, $\boxed{T(n/2) = 2T(n/4) + n/2}$ - put in 1.

$$= 2(2T(n/4) + n/2) + n$$

$$= 4T(n/4) + n + n$$

$$= 4T(n/4) + 2n \quad - ⓐ$$

Put $\boxed{T(n/4) = 2T(n/4) + n/4}$ in ⓐ

$$= 4\left(2T\left(\frac{n/2}{4}\right) + \frac{n/2}{4}\right) + 2n$$

$$= 8T(n/8) + (n/8) + 2n$$

$$=$$

$\rightarrow 4T(n/4) + 2n.$

$$4\left(2T\left(\frac{n/2}{4}\right) + \frac{n}{4}\right) + 2n.$$

$$= 4 \cdot 2\, T(n/8) + 4 \cdot \frac{n}{4} + 2n$$

$$= 8T(n/8) + n + 2n$$

$$= 8T(n/8) + 3n.$$

$$= 8T\left(\frac{n}{2^3}\right) + 3n.$$

$$= 8T\left(\frac{n}{2^i}\right) + 3n. \rightarrow$$

$$= \cancel{c\, T(n/2^i) + 3n}$$

Since $\frac{n}{2^i} = 1$, $n = 2^i$

$\therefore i = \log_2 n$

Total time $(T_n)$.

$$= c_n(i)$$

$$= c_n(\log_2 n)$$

$$= c_n \lg n$$

$$\boxed{T(n) = \Theta(n \lg n)}$$

$$T(n) = 2T(n/2) + n \quad \text{— (1)}$$

Put $\boxed{T(n/2) = 2T(n/4) + n/2}$ in (1)

$$T(n) = 2(2T(n/4) + n/2) + n$$

$$= 4T(n/4) + n + n$$

$$= 4T(n/4) + 2n.$$

$\boxed{\text{Put } T(n/4) = (2T(n/8) + n/4)}$

$$= 4\cdot(2T(n/8) + n/4) + 2n$$

$$= 8T(n/8) + n + 2n.$$

$$= 8T(n/8) + 3n.$$

$$\vdots$$

$$= 2^3 T\left(\frac{n}{2^3}\right) + 3n$$

$$\vdots$$

$$= 2^i T\left(\frac{n}{2^i}\right) + 3n$$

So, here, $\frac{n}{2^i} = 1$ [i.e. 1 subproblem to be solved].

so $n = 2^i$

- Taking log on bothsides, we get

$$\log n = \log 2^i$$

$$\log n = i \log 2$$

$$i = \frac{\log n}{\log 2}$$

$$\boxed{i = \log_2 n}$$

So, $T(n) = cn(i)$.

$\boxed{i = \log_2 n}$ where 'i' is the number of level or steps of recursive tree which taken $\boxed{\frac{n}{2^i} = 1}$ and not that at each level it ~~cost~~ taken time $= n$.

$\therefore$ total running time is $= n \cdot i$

$$\boxed{T(n) = n \cdot \log_2 n}$$

(b) Iteration method.

$$T(n) = 2 \cdot T\left(\frac{n}{2}\right) + n \quad \text{——— (1)}$$

$$= 2 \cdot \left[\underbrace{2 \cdot T\left(\frac{n}{2^2}\right) + \frac{n}{2}}\right] + n \qquad \left\{T\left(\frac{n}{2}\right) = 2 \cdot T\left(\frac{n}{4}\right) + \frac{n}{2}\right\}$$

$$= 2^2 \cdot T\left(\frac{n}{2^2}\right) + n + n$$

$$= 2^2 \cdot \left[\underbrace{2 \cdot T\left(\frac{n}{2^3}\right) + \frac{n}{2^2}}\right] + 2n \qquad \left\{T\left(\frac{n}{4}\right) = 2 \cdot T\left(\frac{n}{8}\right) + \frac{n}{4}\right\}$$

$$= 2^3 \cdot T\left(\frac{n}{2^3}\right) + n + 2n$$

$$= 2^3 \cdot T\left(\frac{n}{2^3}\right) + 3n$$

let iterate it for 'i' times we get

$$T(n) = 2^i \cdot T\left(\frac{n}{2^i}\right) + i \cdot n \quad \text{——} \quad (2)$$

Assume, $T\left(\frac{n}{2^i}\right) = T(1)$

$$\frac{n}{2^i} = 1 \Rightarrow n = 2^i$$

taking 'log' both sides :-

$$\boxed{\log_2 n = i}$$

putting value of 'i' in eq: (2) we get

$$T(n) = 2^{\log_2 n} \cdot T(1) + n \cdot \log_2 n$$

$$T(n) = n * 1 + n \log_2 n$$

$$\therefore \boxed{T(n) = O(n \log_2 n)}$$

[∵ by removing lower order terms]

Hence proved.

## * Recursion Tree Method.

**Example ①** $T(n) = 3T(n/3) + cn$ ← this is $f(n)$.

← in terms of this.

So,

(3 children).

**Note:** $T(n) = cn + T(n/3) + T(n/3) + T(n/3)$.

↓ expand these nodes further

So, $T(n/3) = 3T(n/3/3)$

$= 3T(n/9) + c(\frac{n}{3})$

so now:

$T(n/9) = 3T(n/9) + c(n/9)$

This tree will look like (at back of the page)

leaf node → $c(\frac{n}{3^i})$

So, $c\left(\frac{n}{3^i}\right) = 1$ [Subproblems are only 1 now from n to 1.]

$n = 3^i$

Get log on both sides.

$\log n = \log 3^i$

$\log n = i \log 3$

$i = \frac{\log n}{\log 3}$

$\boxed{i = \log_3 n.}$

So. $T(n) = 3^i \times T(1) + cn + cn + cn \cdots\cdots$ (from Lev. 0 to h-1).

$= 3^h \times T(1) + \sum_{i=0}^{h} cn.$

(lower bound) $= 3^{\log_3 n} T(1) + cn \log_3 n.$

as constant so, $\Theta$ $= n\,T(1) + cn \log_3 n \rightarrow \Theta(n \log n)$

— Height and cost at each level —

$T(n)$ = value of all nodes of the tree at each levels.

cost at level 1 $= cn$

lev. 2 $= 3\left(\frac{cn}{3}\right) = cn.$

$= 3^2\left(\frac{cn}{9}\right) = cn.$

$\vdots$

$3^i\left(\frac{cn}{3^i}\right) = cn.$

— So, level sum will always be cn. (except for h (height))

$T(1)$ will be some constant k.

## ⊕. Recursion Tree Method: (Merge Sort):

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

So, $\frac{n}{2^i} = 1 \quad i = \log_2 n.$

$$T(n) = 2^i \sum_{i=0}^{\log_2 n} n$$

$$= 2^{\log_2 n}(T(1)) + n\log_2 n$$

$$= \underbrace{n(T(1))}_{\text{some constant}} + n\log_2 n$$

$$T(n) = \Theta(n\log_2 n).$$

1. **CS 5329 Algorithm Design and Analysis**
{*from Dr. Hwang*}

(a) **Write** the program algorithm in some detail for the Partition function in the following QuickSort (A, p, r) function, given A an array with the first index p and with the last index r.

```
QuickSort (A, p, r)

If p < r

Then q <- Partition(A, p, r)

QuickSort (A, p, q-1)

QuickSort (A, q+1, r)
```

(b) Assuming the Partition function of QuickSort always splits the sub-array of array A into 20% and 80%, and n is size of array A, **use recursion tree** and **log algebra** to **derive** the computation time T(n) for such special case of QuickSort in terms of big O **or** in terms of Theta. (Both proof and result are required.)

```
PARTITION(A, p, r)
x = A[r]
i = p - 1
for j = p to r - 1
    if A[j] ≤ x
        i = i + 1
        exchange A[i] with A[j]
exchange A[i + 1] with A[r]
return i + 1
```

```
QUICKSORT(A, p, r)
if p < r
    then q ← PARTITION(A, p, r)
         QUICKSORT(A, p, q − 1)
         QUICKSORT(A, q + 1, r)
```

$\Theta(n \lg n)$

$O(n)$ 3rd ed.

Recursion : $\lg n$

```
PARTITION(A, p, r)
x ← A[r]
i ← p − 1
for j ← p to r − 1
    do if A[j] ≤ x
          then i ← i + 1
               exchange A[i] ↔ A[j]
exchange A[i + 1] ↔ A[r]
return i + 1
```

$\Theta(n)$ $O(n)$

$\Theta(n)$ $O(n)$

2nd

Shortest path height:

$$\frac{n}{10^x} = 1 \Rightarrow 10^x = n \Rightarrow x = \log_{10} n$$

$$x \Rightarrow x = \frac{\log_2 n}{\log_2 10} = \left(\frac{1}{\lg 10}\right)(\lg n)$$

$$= c_1 \lg n$$

longest Path height:

$$n\left(\frac{9}{10}\right)^y = 1 \Rightarrow \left(\frac{10}{9}\right)^y = n$$

$$y = \log_{\frac{10}{9}} n$$

- 9:1 split still a partition.
- $T(n) = T\left(\frac{9n}{10}\right) + T\left(\frac{n}{10}\right) + \theta(n) \le \left(\log_{\frac{10}{9}} n\right) \cdot \theta(n)$

$$= \theta(\lg n) \cdot \theta(n) = \theta(n \lg n)$$

$$\log_{\frac{10}{9}} n = \frac{\lg n}{\lg \frac{10}{9}} = c \lg n \ (c > 0)$$

∴ Balanced Partition for Quick Sort $T(n) = \theta(n \lg n)$

- Balanced Partitioning : assuming 9 : 1 proportional split
(average case)

$\Theta(n \lg n)$

$\Theta(n)$

$$\log_{\frac{10}{9}} n = \frac{\lg n}{\lg \frac{10}{9}} = c \lg n \quad (c > 0)$$

Shortest path height:

$$\frac{n}{10^x} = 1 \Rightarrow 10^x = n \Rightarrow x = \log_{10} n$$

$$x \Rightarrow x = \frac{\log_2 n}{\log_2 10} = \left(\frac{1}{\lg 10}\right)(\lg n)$$

$$= \boxed{c_1 (\lg n)}$$

longest Path height:

$$n\left(\frac{9}{10}\right)^y = 1 \Rightarrow \left(\frac{10}{9}\right)^y = n$$

$$y = \log_{\frac{10}{9}} n = \frac{1}{\log_2 \frac{10}{9}}(\log_2 n)$$

$$= \boxed{c_2 (\lg n)}$$

$$T(n) = n \cdot c_2 (\lg n)$$

$$= n \lg n$$

1% & 99%,

then what is $T(n)$? Derive the answer.

$$n\left(\frac{1}{100}\right)^i = 1 \qquad 100^i = n$$

$$i = \log_{100} n = \frac{\lg n}{\lg 100} = 6.7\ ?$$

$$n\left(\frac{99}{100}\right)^i = 1 \qquad \frac{99}{100}\left(\frac{100}{99}\right)^i = n$$

$$i = \log_{\frac{100}{99}} n = \frac{\lg n}{\lg \frac{100}{99}}$$

when iteration steps

$$\frac{n}{2^i} = 1 \qquad n = 2^i$$

$$i = \lg n$$

$$T(n) = 2^{\lg n} T(1) + (\lg n) n$$

Given $T(n) = \ldots \Rightarrow \Theta(n \lg n)$

(II) Iteration

$$T(n) = 2 \cdot T\left(\frac{n}{2}\right) + n$$

$$= 2 \cdot \left(2 \cdot T\left(\frac{n}{4}\right) + \frac{n}{2}\right) + n$$

$$= 2^2 T\left(\frac{n}{2^2}\right) + n + n$$

$$= 2^2 \left(2 \cdot T\left(\frac{n}{3}\right) + \frac{n}{2^2}\right) + 2n$$

$$= 2^3 T\left(\frac{n}{2^3}\right) + \underbrace{n + 2n}_{3n}$$

when iteration steps

$$\frac{n}{2^i} = 1 \qquad n = 2^i$$

$$i = \lg n$$

$$T(n) = 2^{\lg n} T(1) + (\lg n) n$$

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

$$= \Theta(n \lg n)$$

Given ~~Q~~ Exm

$$T(n) = n + 3T\left(\left\lfloor \frac{n}{4} \right\rfloor\right)$$

to prove

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

$$= n + 3\left(\frac{n}{4} + 3T\left(\left\lfloor \frac{\frac{n}{4}}{4} \right\rfloor\right)\right)$$

$$= n + \frac{3}{4}n + 3^2 T\left(\frac{n}{4^2}\right)$$

$$= n + \frac{3}{4}n + 3^2\left(\frac{n}{4^2} + 3 \cdot T\left(\left\lfloor \frac{\frac{n}{4^2}}{4} \right\rfloor\right)\right)$$

$$= n + \frac{3}{4}n + \left(\frac{3}{4}\right)^2 n + 3^3 T\left(\frac{n}{4^3}\right)$$

$$= n + \frac{3}{4}n + \left(\frac{3}{4}\right)^2 n + \left(\frac{3}{4}\right)^3 n + \cdots$$

$$+ \cdots + 3^i T\left(\frac{n}{4^i}\right)$$

$$\frac{n}{4^i} = 1 \Rightarrow n = 4^i \Rightarrow i = \log_4 n$$

II) Iteration Method on recurrence: $T(n) = 3T\left(\left\lfloor \frac{n}{4} \right\rfloor\right) + n$

$$\begin{aligned}
T(n) &= n + 3T(\lfloor n/4 \rfloor) \\
&= n + 3(\lfloor n/4 \rfloor + 3T(\lfloor n/16 \rfloor)) \\
&= n + 3(\lfloor n/4 \rfloor + 3(\lfloor n/16 \rfloor + 3T(\lfloor n/64 \rfloor))) \\
&= n + 3\lfloor n/4 \rfloor + 9\lfloor n/16 \rfloor + 27T(\lfloor n/64 \rfloor).
\end{aligned}$$

$$\begin{aligned}
T(n) &\le n + 3n/4 + 9n/16 + 27n/64 + \cdots + 3^{\log_4 n}\Theta(1) \\
&\le n\sum_{i=0}^{\infty}\left(\frac{3}{4}\right)^i + \Theta(n^{\log_4 3}) \\
&= 4n + o(n) \\
&= O(n).
\end{aligned}$$

Example: Recurrence $T(n) = T(\frac{n}{3}) + T(\frac{2n}{3}) + n$
(~~Theorem~~)

$T(n) \Rightarrow O(n \lg n)$

The boundary condition $T(1) = \Theta(1)$

when $T\left(\left(\frac{2}{3}\right)^k n\right) = T(1) = \Theta(1)$

$\left(\frac{2}{3}\right)^k n = 1 \qquad n = \left(\frac{3}{2}\right)^k$

$k = \log_{\frac{3}{2}} n$ the height of the recursion tree

Example: Recurrence $T(n) = T(\frac{n}{3}) + T(\frac{2n}{3}) + n$

$\Rightarrow T(n) = O(n \lg n)$

$k = \log_{3/2} n$

The boundary condition $T(1) = \Theta(1)$

when $T\left(\left(\frac{2}{3}\right)^k n\right) = T(1) = \Theta(1)$

$\left(\frac{2}{3}\right)^k n = 1 \qquad n = \left(\frac{3}{2}\right)^k$

**Recursion Tree** (Iteration Method): to visualize what happens when a recurrence is iterated.

$$T(n) = 2T\left(\frac{n}{2}\right) + n^2$$

(Iteration)

$$T(n) = 2\cdot\boxed{T\left(\frac{n}{2}\right)} + n^2$$

$T(n) = \Rightarrow \Theta(n^2)$

$$= 2\cdot\boxed{\left(2\cdot T\left(\frac{\frac{n}{2}}{2}\right) + \left(\frac{n}{2}\right)^2\right)} + n^2$$

$$= 2^2\cdot T\left(\frac{n}{2^2}\right) + 2\cdot\frac{n^2}{2^2} + n^2$$

$$= n^2 + \frac{n^2}{2} + 4\boxed{T\left(\frac{n}{4}\right)} \checkmark$$

$$= n^2 + \frac{n^2}{2} + 4\cdot\left(2\cdot T\left(\frac{n}{8}\right) + \frac{n^2}{4^2}\right)$$

$$+ \frac{n^2}{2} + 4\boxed{T\left(\frac{n}{4}\right)} \checkmark$$

$$= n^2 + \frac{n^2}{2} + 4\cdot\left(2\cdot T\left(\frac{n}{8}\right) + \frac{n^2}{4^2}\right)$$

$$= n^2 + \frac{1}{2}n^2 + \left(8\cdot T\left(\frac{n}{8}\right) + \frac{n^2}{4}\right)$$

$$= n^2 + \frac{1}{2}n^2 + \frac{1}{4}n^2 + \underset{2^3}{8}\cdot T\left(\frac{n}{8}\right) \checkmark$$

$$\leq n^2 + \frac{1}{2}n^2 + \frac{1}{4}n^2 + \frac{1}{8}n^2 + \dots$$

$$\leq n^2\left(\frac{1}{1-\frac{1}{2}}\right) = 2n^2$$

$n \rightarrow$

$$+ 2^i\, T\left(\frac{n}{2^i}\right)$$

(Iteration)

$$T(n) = 2 \cdot \boxed{T\left(\frac{n}{2}\right)} + n^2$$

$$T(n) = \Rightarrow \Theta(n^2)$$

$$2^i \, T\left(\frac{n}{2^i}\right)$$

$$= 2^{\lg n} \, T(1)$$

$$= n^{\lg 2} = n$$

$$\frac{n}{2^i} = 1$$

$$2^i = n$$

$$i = \lg n$$

$$\cancel{T(n) \leq 2^n + n}$$

$$T(n) \leq 2n^2 + n = \Theta(n^2)$$

f) $20 = \frac{20}{100} = \frac{2}{10} = \frac{1}{5}$

$80 = \frac{80}{100} = \frac{8}{10} = \frac{4}{5}$

The recursion tree :

### 7.2.1 Architecture and Services

In this section, we will provide an overview of how email systems are organized and what they can do. The architecture of the email system is shown in Fig. 7-7. It consists of two kinds of subsystems: the **user agents**, which allow people to read and send email, and the **message transfer agents**, which move the messages from the source to the destination. We will also refer to message transfer agents informally as **mail servers**.

**Figure 7-7.** Architecture of the email system.

The user agent is a program that provides a graphical interface, or sometimes a text- and command-based interface that lets users interact with the email system. It includes a means to compose messages and replies to messages, display incoming messages, and organize messages by filing, searching, and discarding them. The act of sending new messages into the mail system for delivery is called **mail submission**.

Some of the user agent processing may be done automatically, anticipating what the user wants. For example, incoming mail may be filtered to extract or

deprioritize messages that are likely spam. Some user agents include advanced features, such as arranging for automatic email responses ("I'm having a wonderful vacation and it will be a while before I get back to you"). A user agent runs on the same computer on which a user reads her mail. It is just another program and may be run only some of the time.

The message transfer agents are typically system processes. They run in the background on mail server machines and are intended to be always available. Their job is to automatically move email through the system from the originator to the recipient with **SMTP** (**Simple Mail Transfer Protocol**). This is the message transfer step.

SMTP was originally specified as RFC 821 and revised to become the current RFC 5321. It sends mail over connections and reports back the delivery status and any errors. Numerous applications exist in which confirmation of delivery is important and may even have legal significance ("Well, Your Honor, my email system is just not very reliable, so I guess the electronic subpoena just got lost somewhere").

Message transfer agents also implement **mailing lists**, in which an identical copy of a message is delivered to everyone on a list of email addresses. Other advanced features are carbon copies, blind carbon copies, high-priority email, secret (i.e., encrypted) email, alternative recipients if the primary one is not currently available, and the ability for assistants to read and answer their bosses' email.

Linking user agents and message transfer agents are the concepts of mailboxes and a standard format for email messages. **Mailboxes** store the email that is received for a user. They are maintained by mail servers. User agents simply present users with a view of the contents of their mailboxes. To do this, the user agents send the mail servers commands to manipulate the mailboxes, inspecting their contents, deleting messages, and so on. The retrieval of mail is the final delivery (step 3) in Fig. 7-7. With this architecture, one user may use different user agents on multiple computers to access one mailbox.

Mail is sent between message transfer agents in a standard format. The original format, RFC 822, has been revised to the current RFC 5322 and extended with support for multimedia content and international text. This scheme is called MIME and will be discussed later. People still refer to Internet email as RFC 822, though.

A key idea in the message format is the distinction between the **envelope** and its contents. The envelope encapsulates the message. It contains all the information needed for transporting the message, such as the destination address, priority, and security level, all of which are distinct from the message itself. The message transport agents use the envelope for routing, just as the post office does.

The message inside the envelope consists of two separate parts: the **header** and the **body**. The header contains control information for the user agents. The body is entirely for the human recipient. None of the agents care much about it. Envelopes and messages are illustrated in Fig. 7-8.

## ⊛ CROW FOOT NOTATIONS.

- one-to-one

- many-to-many

- one-to-many

Follow this!!

# University / Enrollment ER

- Functional Dependencies

1. Student (StudentID, FName, LName, Address, Email)
2. Instructor (InstructorID, i_FName, i_LName, i-address, i_email)
3. Course (course_number, description, credit-hour, prerequisite)
4. classroom (roomNumber, size, num_seats, equipment)
5. student_class (studentID (FK), CRN (FK), Grade)
6. Class_Seminar (CRN, sectionNo, semester, class_time, instructorID (FK), course_number (FK), room_number (FK))

5. **CS 5391 Survey of Software Engineering**
{ *from Dr. Palacios* }

The Capability Maturity Model (CMM) was developed as a Software Process Improvement framework.

(a) What is this model predicated on?

(b) Give examples of the Key Process Areas that the model assesses, and

(c) Describe the six Capability Levels defined by the model.

- The term "maturity" relates to the degree of formality and optimization of processes,
- The model's aim is to improve existing software development processes, but it can also be applied to other processes.
- A maturity model can be viewed as a set of structured levels that describe how well the behaviors, practices and processes of an organization can reliably and sustainably produce required outcomes.

## Structure[edit]

The model involves five aspects:

- *Maturity Levels:* a 5-level process maturity continuum - where the uppermost (5th) level is a notional ideal state where processes would be systematically managed by a combination of process optimization and continuous process improvement.
- *Key Process Areas:* a Key Process Area identifies a cluster of related activities that, when performed together, achieve a set of goals considered important.
- *Goals:* the goals of a key process area summarize the states that must exist for that key process area to have been implemented in an effective and lasting way. The extent to which the goals have been accomplished is an indicator of how much capability the organization has established at that maturity level. The goals signify the scope, boundaries, and intent of each key process area.
- *Common Features:* common features include practices that implement and institutionalize a key process area. There are five types of common features: commitment to perform, ability to perform, activities performed, measurement and analysis, and verifying implementation.
- *Key Practices:* The key practices describe the elements of infrastructure and practice that contribute most effectively to the implementation and institutionalization of the area.

The software process framework documented is intended to guide those wishing to assess an organization's or project's consistency with the Key Process Areas. For each maturity level there are five checklist types:

| Type | Description |
|---|---|
| Policy | Describes the policy contents and KPA goals recommended by the Key Process Areas. |
| Standard | Describes the recommended content of select work products described in the Key Process Areas. |
| Process | Describes the process information content recommended by the Key Process Areas. These are refined into checklists for:<br>• Roles, entry criteria, inputs, activities, outputs, exit criteria, reviews and audits, work products managed and controlled, measurements, documented procedures, training, and tools |

| | |
|---|---|
| Procedure | Describes the recommended content of documented procedures described in the Key Process Areas. |
| Level overview | Provides an overview of an entire maturity level. These are further refined into checklists for:<br>• Key Process Areas purposes, goals, policies, and standards; process descriptions; procedures; training; tools; reviews and audits; work products; measurements |

## Levels[edit]

There are five levels defined along the continuum of the model and, according to the SEI: "Predictability, effectiveness, and control of an organization's software processes are believed to improve as the organization moves up these five levels. While not rigorous, the empirical evidence to date supports this belief".[17]

1. *Initial* (chaotic, ad hoc, individual heroics) - the starting point for use of a new or undocumented repeat process.
2. *Repeatable* - the process is at least documented sufficiently such that repeating the same steps may be attempted.
3. *Defined* - the process is defined/confirmed as a standard business process
4. *Capable* - the process is quantitatively managed in accordance with agreed-upon metrics.
5. *Efficient* - process management includes deliberate process optimization/improvement.

Within each of these maturity levels are Key Process Areas which characterise that level, and for each such area there are five factors: goals, commitment, ability, measurement, and verification. These are not necessarily unique to CMM, representing — as they do — the stages that organizations must go through on the way to becoming mature.

The model provides a theoretical continuum along which process maturity can be developed incrementally from one level to the next. Skipping levels is not allowed/feasible.

**Level 1 - *Initial***

It is characteristic of processes at this level that they are (typically) undocumented and in a state of dynamic change, tending to be driven in an *ad hoc*, uncontrolled and reactive manner by users or events. This provides a chaotic or unstable environment for the processes. (Example - a surgeon performing a new operation a small number of times - the levels of negative outcome are not known).

**Level 2 - *Repeatable***

It is characteristic of this level of maturity that some processes are repeatable, possibly with consistent results. Process discipline is unlikely to be rigorous, but where it exists it may help to ensure that existing processes are maintained during times of stress.

**Level 3 - *Defined***

It is characteristic of processes at this level that there are sets of defined and documented standard processes established and subject to some degree of improvement over time. These standard processes are in place. The processes may not have been systematically or repeatedly used - sufficient for the users to become competent or the process to be validated in a range of situations. This could be considered a developmental stage - with use in a wider range of conditions and user competence development the process can develop to next level of maturity.

**Level 4 - *Managed (Capable)***

It is characteristic of processes at this level that, using process metrics, effective achievement of the process objectives can be evidenced across a range of operational conditions. The suitability of the process in multiple environments has been tested and the process refined and adapted. Process users have experienced the process in multiple and varied conditions, and are able to demonstrate competence. The process maturity enables adaptions to particular projects without measurable losses of quality or deviations

from specifications. Process Capability is established from this level. (Example - surgeon performing an operation hundreds of times with levels of negative outcome approaching zero).

**Level 5 - *Optimizing (Efficient)***

It is a characteristic of processes at this level that the focus is on continually improving process performance through both incremental and innovative technological changes/improvements. At maturity level 5, processes are concerned with addressing statistical *common causes* of process variation and changing the process (for example, to shift the mean of the process performance) to improve process performance. This would be done at the same time as maintaining the likelihood of achieving the established quantitative process-improvement objectives.

- one or many

- one and only one

- zero or one

9. **CS 5332 Data Base Theory and Design**
{ *from Dr. Ngu* }

Consider the following typed relational schema describing books, authors, borrowings and borrowers in a library:

Book(isbn:integer, title:string, publisher:string, year:integer)
Author(isbn:integer, name:string, rank:integer)
Borrower(bid:integer, name:string, address:string)
Borrowings(isbn:integer, borrower:integer, whenTaken:date, whenReturned:date)

Assume the following:

- the fields that are underlined are the primary keys of the relations
- there is a referential integrity constraint between "borrower" in relation Borrowings and "bid" in relation Borrower
- there are also referential integrity constraints between "isbn" in relation Book and "isbn" in relations Author and Borrowings

- dates are represented as a count of the number of days since Jan 1 2019
- there is a system function `today` that returns today's `date`
- the relational operators < and > allow you test *before* and *after* on `dates`
- all loans are for a period of 14 days
- the `whenReturned` field is `null` until the book is returned
- the `rank` field specifies whether the author is the first, second, ... author

Implement solutions to the following:

(a) Write an SQL query to show all the books (`title` only) written by a particular author called "Ullman".

(b) Write an SQL query to show which books (`title` only) have multiple authors.

(c) Write an SQL query to show which books (`title` only) have never been borrowed.

(d) Write an SQL query to show which books (`title` only) are currently out on loan.

(e) Write an SQL query to list overdue books (`title`) and the borrower's name

```
-- (a)
select b.title from Book b
left join Author a on a.isbn = b.isbn
where a.name = "Ullman";
select 'a';

-- (b)
select b.title from Book b
left join Author a on a.isbn = b.isbn
group by b.title
having COUNT(b.title)>1;

select 'b';

-- (c)
select title from Book
where isbn not in (select isbn from Borrowing);

select 'c';

-- (d)
select b.title from Book b
left join Borrowing w on b.isbn = w.isbn
where w.whenReturned is null;

select 'd';

-- (e)
select b.title, ber.name from Book b
left join Borrowing w ON b.isbn = w.isbn
left join Borrower ber ON w.borrower = ber.bid
```

where w.whenReturned - w.whenTaken > 14 or (w.whenReturned is null && DATEDIFF(CURDATE(),'2019-01-01') - w.whenTaken > 14);

select 'e';

11. **CS 5338 Formal Languages**
    {*from Dr. Gao*}

    (a) Draw a PDA $M$ for language $L(M) = \{a^{3n}b^n : n \geq 0\}$.

    (b) In plain English, describe a turning machine $M$ that decides $a^n b^n c^n$.

12. **CS 5338 Formal Languages**
    {*from Dr. Singh*}

    What are different ways to check for membership?

# Membership question

- Decision Problem: Given L, is w in L?
- Possible answers: Yes or No
- Approach:

1. Build a DFA for L
2. Input w to the DFA
3. If the DFA ends in an accepting state, then yes; otherwise no.

# Example 3: Pumping Lemma

**Claim: L = { 0i | i is a perfect square} is not regular**

- Proof:
  - By contradiction, let L be regular.
  - P/L should apply
  - Let *N* = P/L constant
  - Choose w=0N2
  - By pumping lemma, w=xyz satisfying all three rules
  - By rules (1) & (2), y has between 1 and N 0s
  - By rule (3), any string of the form xykz is also in L for all k≥0
  - Case k=0:
    - #zeros (xy0z) = #zeros (xyz) - #zeros (y)
    - N2 – N ≤ #zeros (xy0z) ≤ N2 - 1
    - **(N-1)2** < N2 - N ≤ #zeros (xy0z) ≤ N2 - 1 < **N2**
    - xy0z ∉ L
    - But the above will complete the proof ONLY IF N>1.
    - … (proof contd.. Next slide)



# Way to check for Membership

- Is a string *w* accepted by a TM?
- Initial condition:
  - The (whole) input string *w* is present in TM, preceded and followed by infinite blank symbols
- Final acceptance:
  - Accept *w* if TM enters final state and halts
  - If TM halts and not final state, then reject

# String membership

How to say if a string belong to the language defined by a CFG?

1. Derivation
   - Head to body
2. Recursive inference
   - Body to head

Both are equivalent forms

<u>Example:</u>

- w = 01110
- Is w a palindrome?

G:
A => 0A0 | 1A1 | 0 | 1 | ▯

A => 0A0
=> 01A10
=> 01110

# The Pumping Lemma for Regular Languages

**What it is?**
The Pumping Lemma is a property of all regular languages.

**How is it used?**
A technique that is used to show that a given language is not regular

Let L be a regular language

Then *there exists* some constant ***N*** such that *for every* string $w \in L$ s.t. $|w| \geq N$, *there exists* a way to break $w$ into three parts, $w=xyz$, such that:

1. $y \neq \varepsilon$
2. $|xy| \leq N$
3. For all $k \geq 0$, all strings of the form $xy^kz \in L$

This property should hold for all regular languages.

**Definition:** $N$ is called the "Pumping Lemma Constant"



# Pumping Lemma: Proof

- L is regular => it should have a DFA.
  - Set $N$ := number of states in the DFA
- Any string $w \in L$, s.t. $|w| \geq N$, should have t form: $w=a_1a_2\ldots a_m$, where $m \geq N$
- Let the states traversed after reading the N symbols be: $\{p_0, p_1, \ldots p_N\}$
  - ==> There are N+1 p-states, while there are c N DFA states
  - ==> at least one state has to repeat i.e, $p_i = p_J$ where $0 \leq i < j \leq N$ (by PHP)

# The Purpose of the Pumping Lemma for RL

- To prove that some languages *cannot be* regular.

# How to use the pumping lemma?

Think of playing a 2 person game

- Role 1: ***We*** claim that the language cannot be regular
- Role 2: An ***adversary*** who claims the language is regular
- We show that the adversary's statement will lead to a contradiction that implies pumping lemma *cannot* hold for the language.
- We win!!



# How to use the pumping lemma? (The Steps)

1. (we) L is not regular.
2. (adv.) Claims that L is regular and gives you a value for N as its P/L constant
3. (we) Using N, choose a string $w \in L$ s.t.,
   1. $|w| \geq N$,
   2. Using w as the template, construct other words $w_k$ of the form $xy^kz$ and show that at least one such $w_k \notin L$

      => this implies we have successfully broken the pumping lemma for the language, and hence that the adversary is wrong.

(Note: In this process, we may have to try many values of k, starting with k=0, and then 2, 3, .. so on, until $w_k \notin L$ )

Note: We don't have any control over N, except that it is positive.
We also don't have any control over how to split w=xyz,
but xyz should respect the P/L conditions (1) and (2).

# Using the Pumping Lemma

- What WE do?

- What the Adversary does?
  1. Claims L is regular
  2. Provides *N*

3. Using *N*, we construct our template string *w*
4. Demonstrate to the adversary, either through pumping up or down on *w*, that some string $w_k \notin L$
   (this should happen regardless of w=xyz)



Note: This N can be anything (need not necessarily be the #states in the DFA.
It's the adversary's choice.)

# Example of using the Pumping Lemma to prove that a language is not regular

**Let $L_{eq}$ = {w | w is a binary string with equal number of 1s and 0s}**

- Your Claim: $L_{eq}$ is not regular
- Proof:
  - By contradiction, let $L_{eq}$ be regular — adv.
  - P/L constant should exist — adv.
    - Let *N* = that P/L constant
  - Consider input $w = 0^N 1^N$ — you
    *(your choice for the template string)*
  - By pumping lemma, we should be able to break w=xyz, such that: — you
    1) $y \neq \varepsilon$
    2) $|xy| \leq N$
    3) For all $k \geq 0$, the string $xy^kz$ is also in L

Template string $w = 0^N1^N = \underbrace{00\ldots0}_{N}\underbrace{011\ldots1}_{N}$

# Proof…

- Because $|xy| \le N$, $xy$ should contain only 0s
  - (This and because $y \neq \varepsilon$, implies $y=0^+$)
- Therefore $x$ can contain *at most* N-1 0s
- Also, all the N 1s must be inside $z$
- By (3), any string of the form $xy^kz \in L_{eq}$ for all $k \ge 0$
- Case k=0: $xz$ has at most N-1 0s but has N 1s
- Therefore, $xy^0z \notin L_{eq}$
- This violates the P/L (a contradiction) ↯

you

Setting k=0 is referred to as **"pumping down"**

Setting k>1 is referred to as **"pumping up"**

Another way of proving this will be to show that if the #0s is arbitrarily pumped up (e.g., k=2), then the #0s will become exceed the #1s



# Example 3: Pumping Lemma

**Claim: $L = \{ 0^i \mid i \text{ is a perfect square}\}$ is not regular**

- Proof:
  - By contradiction, let L be regular.
  - P/L should apply
  - Let $N$ = P/L constant
  - Choose $w=0^{N2}$
  - By pumping lemma, $w=xyz$ satisfying all three rules
  - By rules (1) & (2), $y$ has between 1 and N 0s
  - By rule (3), any string of the form $xy^kz$ is also in L for all $k \ge 0$
  - Case k=0:
    - #zeros $(xy^0z)$ = #zeros $(xyz)$ - #zeros $(y)$
    - $N^2 - N \le$ #zeros $(xy^0z) \le N^2 - 1$
    - $\mathbf{(N-1)^2} < N^2 - N \le$ #zeros $(xy^0z) \le N^2 - 1 < \mathbf{N^2}$
    - $xy^0z \notin L$
    - But the above will complete the proof ONLY IF N>1.
    - … (proof contd.. Next slide)

# Example 3: Pumping Lemma

- (proof contd…)
  - If the adversary pick N=1, then $(N-1)^2 \leq N^2 - N$, and therefore the #zeros($xy^0z$) could end up being a perfect square!
  - This means that pumping down (i.e., setting k=0) is not giving us the proof!
  - So lets try pumping up next…
- Case k=2:
  - #zeros ($xy^2z$) = #zeros ($xyz$) + #zeros ($y$)
  - $N^2 + 1 \leq$ #zeros ($xy^2z$) $\leq N^2 + N$
  - $N^2 < N^2 + 1 \leq$ #zeros ($xy^2z$) $\leq N^2 + N < (N+1)^2$
  - $xy^2z \notin L$ ↯
  - (Notice that the above should hold for all possible N values of N>0. Therefore, this completes the proof.)

$T(1) = 4 \qquad T(\boxed{n}) = 2T(\boxed{n}/2) + 4\boxed{n}$

Build Solution

$$T(n) = 2T(n/2) + 4n$$

$$= 2\left[2T\left(\frac{n}{2^2}\right) + 4\frac{n}{2}\right] + 4n$$

$$= 2^2 T\left(\frac{n}{2^2}\right) + 4n + 4n$$

$$= 2^2\left[2T\left(\frac{n}{2^3}\right) + 4\frac{n}{2^2}\right] + 4n + 4n$$

$$= 2^3 T\left(\frac{n}{2^3}\right) + 4n + 4n + 4n$$

$$= 2^3\left[2T\left(\frac{n}{2^4}\right) + 4\frac{n}{2^3}\right] + 4n + 4n + 4n$$

$$= 2^4 T\left(\frac{n}{2^4}\right) + \underbrace{4n + 4n + 4n + 4n}_{4}$$

$$= 2^i T\left(\frac{n}{2^i}\right) + i(4n)$$

$$\frac{n}{2^i} = 1 \Rightarrow n = 2^i \Rightarrow \log_2 n = i$$

Expand Scratch

$$T(\boxed{n/2}) = 2T\left(\frac{n}{2^2}\right) + 4\left(\frac{n}{2}\right)$$

$$T\left(\frac{n}{2^2}\right) = 2T\left(\frac{n}{2^3}\right) + 4\left(\frac{n}{2^2}\right)$$

$$T\left(\frac{n}{2^3}\right) = 2T\left(\frac{n}{2^4}\right) + 4\frac{n}{2^3}$$

$$2^{\log_2 n} = n^{\log_2 2} = n$$

$$2^{\log_2 n} T(1) + (\log_2 n)(4n)$$

$$= n(4) + 4n\log_2 n$$

$$= 4n + 4n\log_2 n = O(n \log n)$$

递归树生成实例
W(n) = 2W(n/2) + n-1
W(n)
初始
第一步迭代
n-1
W(n/2)
W(n/2)

递归树生成实例
n-1
W(n/2)
W(n/2)
第二步迭代
n-1
n/2-1
n/2-1
W(n/4)
W(n/4)
W(n/4)
W(n/4)
7

# 对递归树上的量求和

$$\boxed{\begin{aligned} &W(n)=2W(n/2)+n-1,\ n=2^k, \\ &W(1)=0 \end{aligned}}$$

$$\begin{aligned} W(n) &= n-1+n-2+\ldots+n-2^{k-1} \\ &= kn-(2^k-1) \\ &= n\log n-n+1 \end{aligned}$$

递归树应用实例
求解方程：$T(n)=T(n/3)+T(2n/3)+n$
$n$
$\frac{n}{3}$
$\frac{2n}{3}$
$\frac{n}{9}$
$\frac{2n}{9}$
$\frac{2n}{9}$
$\frac{4n}{9}$
$n$
$n$
$n$
1
$O(n)$
10

递归树应用实例
求解方程：$T(n)=T(n/3)+T(2n/3)+n$
$n$
$\frac{n}{3}$
$\frac{2n}{3}$
$\frac{n}{9}$
$\frac{2n}{9}$
$\frac{2n}{9}$
$\frac{4n}{9}$
$n$
$n$
$n$
1
$O(n)$

# 递归树应用实例

求解方程：$T(n)=T(n/3)+T(2n/3)+n$

# 求和

方程：$T(n)=T(n/3)+T(2n/3)+n$

递归树层数 $k$，每层$O(n)$

$$n(2/3)^k = 1$$

$$\Rightarrow (3/2)^k = n$$

$$\Rightarrow k = O(\log_{3/2} n)$$

# 主方法和递归树

这真的是 2 个很牛叉的算法分析方法，你可以用主方法瞬间估算出算法的复杂度

Master Method

T(n) = aT(n/b)+h(n)

a >=1 ; b >1 ; h(n) : 不参与递归的复杂度函数

判断 $n^{\log_b(a)}$与 h(n)的大小关系
= Θ(h(n)) ：该方法的复杂度为 Θ(h(n)*lg(n))

> Θ(h(n)) ：该方法的复杂度为 Θ(n^(log a/log b))

< Θ(h(n)) ：该方法复杂度为 Θ(h(n))

这样可以帮助你快速的分析出你得算法的复杂度是否符合要求。

Recursion Tree Method

主要是把递归式转换成树的形式，则利用树的数学概念和特性可以知道树高和叶子结点个数，从而可以求解出算法的复杂度。当然这种方法更合适你去验证自己的结论，因为是严格的证明过程。

**********

**********

另外一篇：递归树

## 求递归算法时间复杂度：递归树

(2011-09-22 11:53:34)

递归算法时间复杂度的计算方程式一个递归方程：

$$T(n) = \begin{cases} O(1) & n = 1 \\ kT(n/m) + f(n) & n > 1 \end{cases}$$

在引入递归树之前可以考虑一个例子：

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

迭代 2 次可以得：

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

还可以继续迭代，将其完全展开可得：

$T(n) = n^2 + 2((n/2)^2 + 2((n/2^2)^2 + 2((n/2^3)^2 + 2((n/2^4)^2 + \cdots + 2((n/2^i)^2 + 2T(n/2^{i+1})))\cdots)))) \quad \cdots\cdots(1)$

而当 $n/2^{i+1} == 1$ 时，迭代结束。

将(1)式小括号展开，可得：

$T(n) = n^2 + 2(n/2)^2 + 2^2(n/2^2)^2 + \cdots + 2^i(n/2^i)^2 + 2^{i+1}T(n/2^{i+1})$

这恰好是一个树形结构，由此可引出递归树法。

图中的(a)(b)(c)(d)分别是递归树生成的第 1,2,3,n 步。每一节点中都将当前的自由项 $n^2$ 留在其中，而将两个递归项 T(n/2) + T(n/2)分别摊给了他的两个子节点，如此循环。

图中所有节点之和为:

$[1 + 1/2 + (1/2)^2 + (1/2)^3 + \cdots + (1/2)^i]\ n^2 = 2n^2$

可知其时间复杂度为 $O(n^2)$

可以得到递归树的规则为：

(1) 每层的节点为 T(n) = kT(n / m) + f(n)中的 f(n)在当前的 n/m 下的值；

(2) 每个节点的分支数为 k；

(3)每层的右侧标出当前层中所有节点的和。

再举个例子：

T(n) = T(n/3) + T(2n/3) + n

其递归树如下图所示：

可见每层的值都为n，从根到叶节点的最长路径是：

$$n \to \frac{2}{3}n \to (\frac{2}{3})^2 n \to \cdots \to 1$$

因为最后递归的停止是在$(2/3)^k n == 1$.则

$$k = \log_{3/2} n$$

于是

$$T(n) \le \sum_{i=0}^{k} n = (k+1)n = n(\log_{3/2} n + 1)$$

即T(n) = O(nlogn)

总结，利用此方法解递归算法复杂度：

f(n) = af(n/b) + d(n)

1.当d(n)为常数时：

# 主定理

**定理**：设$a\geq 1$, $b>1$为常数, $f(n)$为函数, $T(n)$为非负整数，且$T(n)=aT(n/b)+f(n)$， 则

1. 若 $f(n)=O(n^{\log_b a-\varepsilon})$, $\varepsilon>0$, 那么

$$T(n)=\Theta(n^{\log_b a})$$

存在ε

2. 若 $f(n)=\Theta(n^{\log_b a})$, 那么

$$T(n)=\Theta(n^{\log_b a}\log n)$$

存在ε

3. 若 $f(n)=\Omega(n^{\log_b a+\varepsilon})$, $\varepsilon>0$,且对于某个常数 $c<1$和充分大的 $n$ 有$af(n/b)\leq cf(n)$, 那么

$$T(n)=\Theta(f(n))$$

存在c

# 迭代

$$\boxed{T(n)=aT(n/b)+f(n)}$$

设 $n=b^k$

$$T(n)=aT(\frac{n}{b})+f(n)$$

$$=a[aT(\frac{n}{b^2})+f(\frac{n}{b})]+f(n)$$

$$=a^2T(\frac{n}{b^2})+af(\frac{n}{b})+f(n)$$

$$=\ldots$$

# 迭代结果

$$a^{\log_b n} = n^{\log_b a}$$

$$= a^k T(\frac{n}{b^k}) + a^{k-1} f(\frac{n}{b^{k-1}}) + \ldots + af(\frac{n}{b}) + f(n)$$

$$= a^k T(1) + \sum_{j=0}^{k-1} a^j f(\frac{n}{b^j})$$

$$= c_1 n^{\log_b a} + \sum_{j=0}^{k-1} a^j f(\frac{n}{b^j}) \qquad T(1) = c_1$$

# 迭代结果

$$a^{\log_b n} = n^{\log_b a}$$

$$= a^k T(\frac{n}{b^k}) + a^{k-1} f(\frac{n}{b^{k-1}}) + \ldots + af(\frac{n}{b}) + f(n)$$


$$= \underline{a^k T(1)} + \sum_{j=0}^{k-1} a^j f(\frac{n}{b^j})$$

$$= \underline{c_1 n^{\log_b a}} + \sum_{j=0}^{k-1} a^j f(\frac{n}{b^j}) \qquad T(1) = c_1$$

# 迭代结果

$$a^{\log_b n} = n^{\log_b a}$$

$$= a^k T(\frac{n}{b^k}) + a^{k-1} f(\frac{n}{b^{k-1}}) + \ldots + af(\frac{n}{b}) + f(n)$$

$$= \underline{a^k T(1)} + \sum_{j=0}^{k-1} a^j f(\frac{n}{b^j})$$

$$= \underline{c_1 n^{\log_b a}} + \sum_{j=0}^{k-1} a^j f(\frac{n}{b^j}) \qquad T(1) = c_1$$

- 第一项为所有最小子问题的计算工作量
- 第二项为迭代过程归约到子问题及综合解的工作量

# 求解递推方程：例1

**例1** 求解递推方程

$$T(n) = 9T(n/3) + n$$

解 上述递推方程中的

$a = 9$， $b = 3$，$f(n) = n$

$n^{\log_3 9} = n^2$， $f(n) = O(n^{\log_3 9 - 1})$

相当于主定理的case1，其中$\varepsilon$=1.

根据定理得到 $T(n) = \Theta(n^2)$



# 求解递推方程：例2

**例2** 求解递推方程

$$T(n) = T(2n/3) + 1$$

解 上述递推方程中的

$a = 1, b = 3/2, f(n) = 1$，

$n^{\log_{3/2} 1} = n^0 = 1$

相当于主定理的Case2 .

根据定理得到 $T(n) = \Theta(\log n)$

# 条件验证

要使 $a f(n/b) \le c f(n)$ 成立，

代入 $f(n) = n \log n$，得到

$$3\,(n/4)\log(n/4) \le cn\log n$$

只要 $c \ge 3/4$，上述不等式可以对所有充分大的 $n$ 成立. 相当于主定理的 Case3.

因此有 $T(n)=\Theta(f(n))=\Theta(n\log n)$



# 递归算法分析

二分检索：$W(n)=W(n/2)+1,\ W(1)=1$

$a=1,\ b=2,\ n^{\log_2 1}=1,\ f(n)=1,$

属于Case2，

$W(n)=\Theta(\log n)$

二分归并排序：

$W(n)=2W(n/2)+n-1,\ W(1)=0$

$a=2,\ b=2,\ n^{\log_2 2}=n,\ f(n)=n-1$

属于Case2，

$W(n)=\Theta(n\log n)$