UCSC Silicon Valley Extension

Advanced C Programming

Analysis and Design of Algorithms

Instructor: Radhika Grover

Overview

- Algorithm design techniques
- Empirical measurement of running time
- Asymptotic notation
- Common functions
- Determining recurrence relations for algorithms
- Methods to solve recurrences

Algorithm design techniques

- Exhaustive search algorithm
- Backtracking algorithm
- Branch and bound algorithm
- Divide and conquer algorithm
- Transform and conquer algorithm

Algorithm design techniques

- Greedy algorithm
- Dynamic programming algorithm
- Parallel programming algorithm
- Approximation algorithm
- Genetic algorithm

Exhaustive search

- Try all possible solutions till a solution is found.
- Simple to implement.
- Cost is proportional to number of solutions.

Exhaustive search example

Magic square puzzle

Fill the 4x4 table with 16 distinct integers from 1 to 16 so the sum of numbers in each row and column and diagonal is the same.

5x5 square: $(5^2)! = 10^{25}$ combinations possible, computation time on processor?

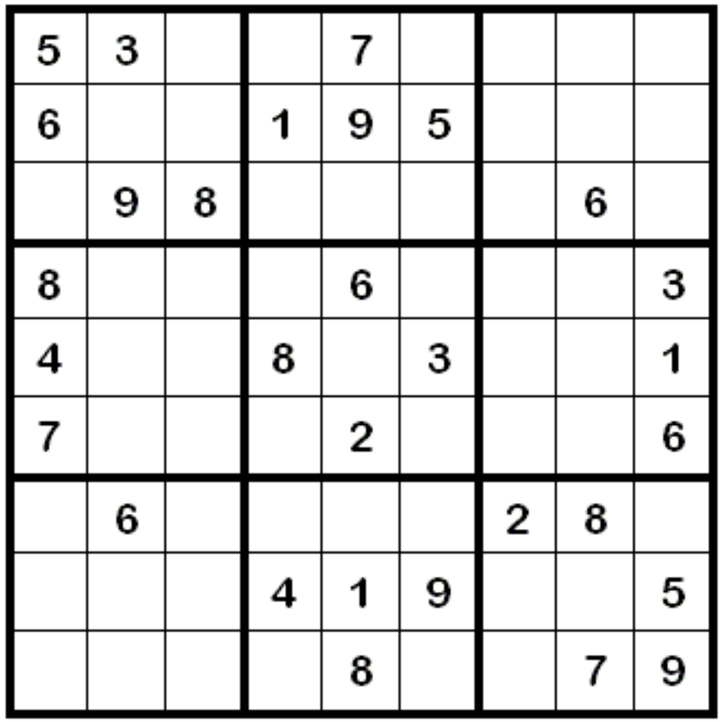
15	10	3	6
4	5	16	9
14	11	2	7
1	8	13	12

Backtracking

- Develop partial solutions further without violating constraints.
- If partial solution is incorrect, don't consider remaining alternatives.
- Replace the partial solution with the next option. (backtrack).

Backtracking example

Sudoku



5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Divide and conquer

- Partition a problem to divide a problem into similar subproblems.
- Solve the subproblem to find the solution.

Divide and conquer example

Given a balance scale with no weights and *n* items where *n* is greater than 1, design an algorithm to find the lightest and heaviest items with the smallest number of weighings.

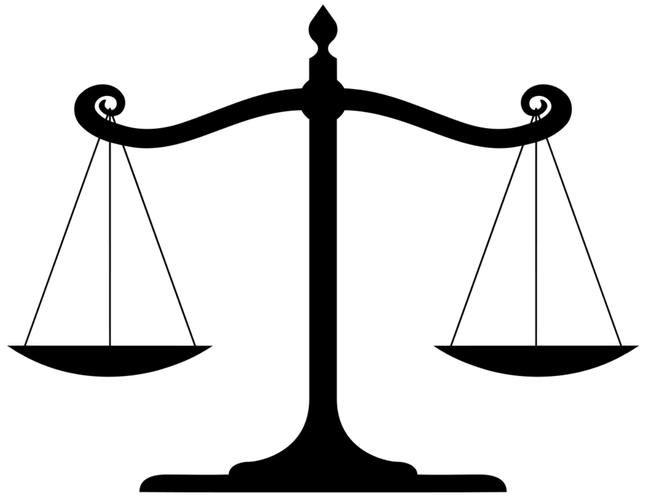
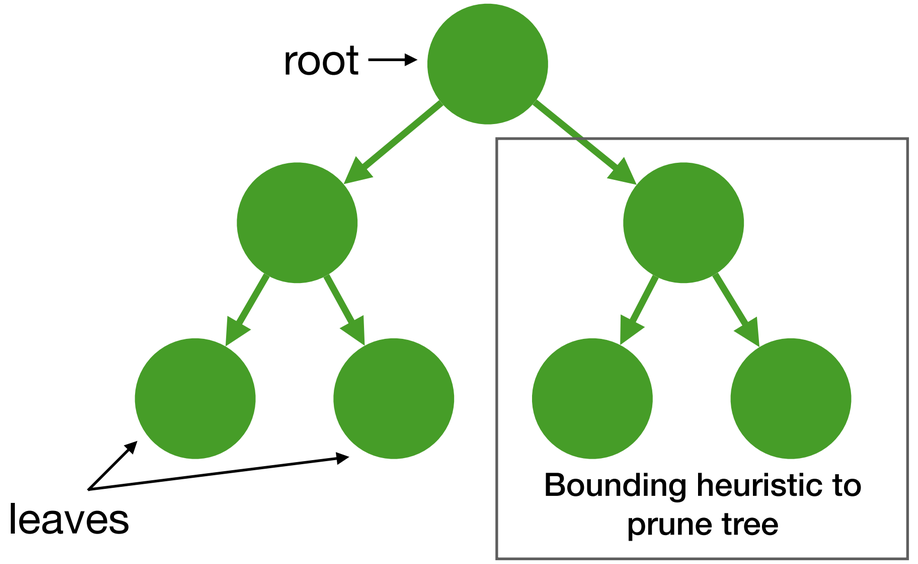


Image Source : Balanced_scale,[1]

Branch and bound

- Uses a heuristic to bound the search space.
- Constructs a tree.
- Leaf nodes have complete solution.



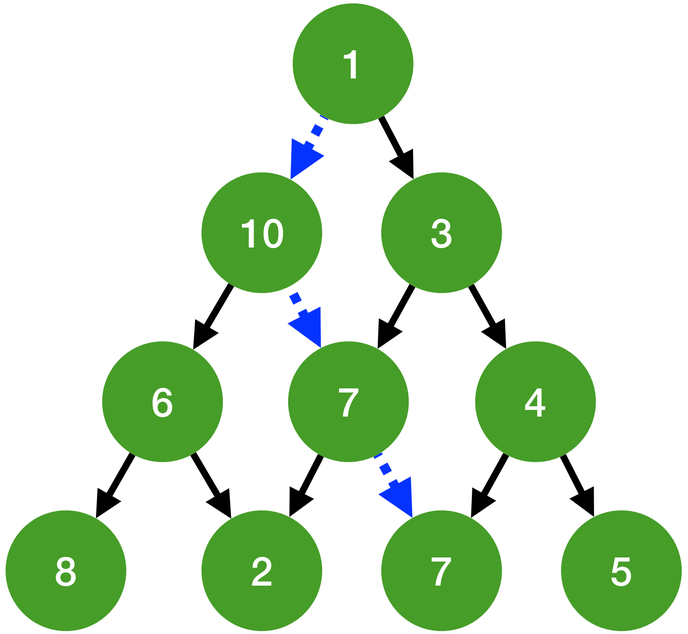
State space tree

Dynamic programming

- Technique to solve a problem with overlapping subproblems.
- Record result of subproblems and use these results instead of solving subproblems repeatedly.
- Used in algorithms such as shortest paths.

Dynamic programming example

Some numbers are arranged in a triangle. Design an algorithm to find largest sum of these numbers on descent from top to bottom while selecting the right or left number at each level.

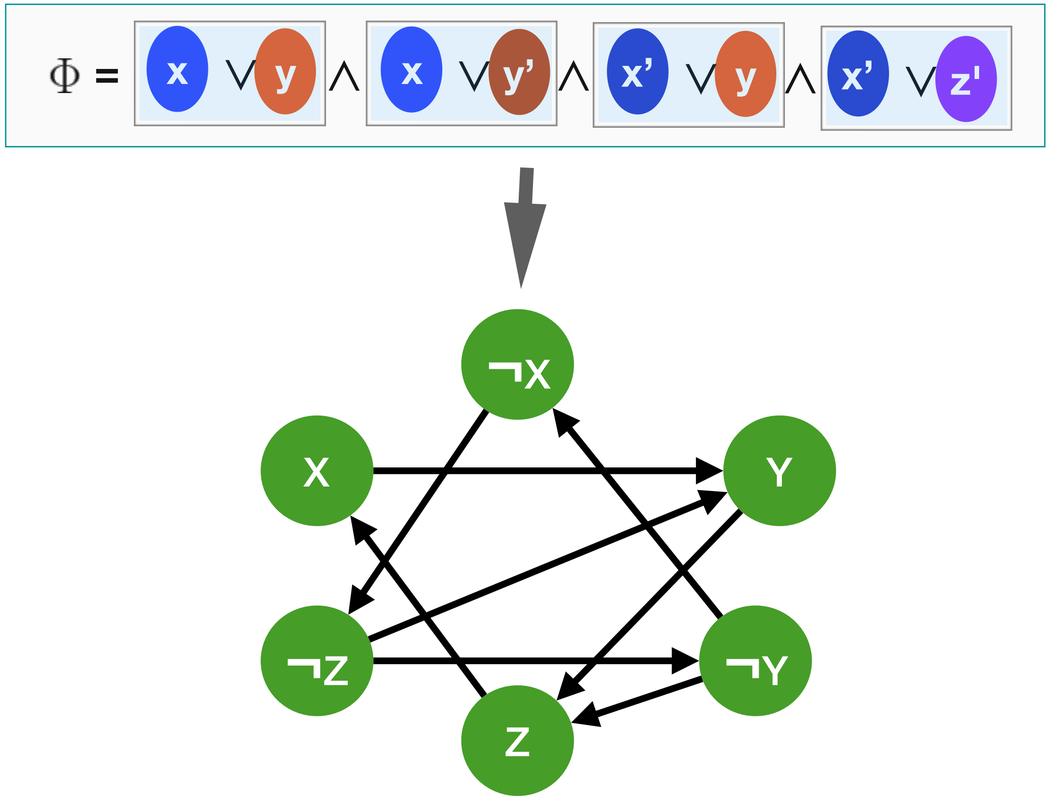


Transform and conquer

Transforms the problem into another problem:

- that may be easier to solve
- whose data structure may be more efficient
- with a known algorithm

Example : 2-SAT using SCC



Greedy algorithm

- Uses a heuristic to find locally optimal choice
- May not find a globally optimal solution
- Example: Knapsack problem

Greedy algorithm example

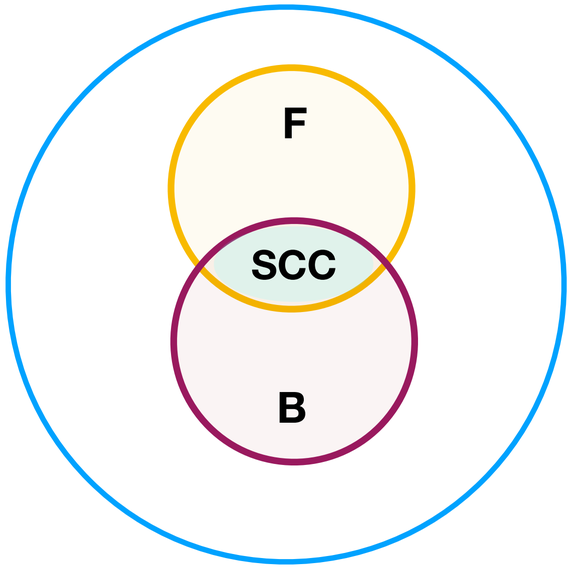
There are 6 identical pots of which #1 is filled with water and the others are empty. Take two vessels and split the total water in them equally between them.

How to minimize the water in pot #1 with a sequence of such operations?

Parallel algorithm

- Algorithm is divided into subproblem that can be executed concurrently on multiple processor to find solution

- Example Forward - Backward algorithm



Approximation algorithm

- Give solutions that are within a factor of optimal solution.
  - α-approximation means the solution for every instance is within factor α of optimum.
- Heuristics don’t provide any guarantees.

Genetic algorithm

- Solve optimization problems
- Based on Darwin’s theory of evolution.
  - Selection, recombination, mutation, acceptance or rejection
- Introduce randomness into the selection
- Inherently parallel

Analysis

Empirical

- Measures execution time for specific (typical) sample of expected inputs
- Machine - dependent

Theoretic

- Machine - independent
- Determines the number of operations as a function of input size

Measure execution time of program

1. Find the total time to execute program for some input size N

   startTime = getCurrentTime( );

   programUnderEvaluation( );

   endTime = getCurrentTime( );

   time to execute = end Time - startTime

2. Repeat step 1 for a different input size

CPU time vs wall time

CPU time

- Time to execute program on processor
- Does not include time waiting for I/O and running other program

Wall - clock time

- Total time to complete task
- Includes disk access and memory access times
- Includes time waiting for cpu

Difficulties in measuring execution times

- Caches
- Compilers
- Pipelining
- I/O
- Interrupts
- OS scheduling
- Concurrency
- Bus contention

Measuring CPU time

- The clock( ) function returns cpu time.
- Find the average time by running program many times.

```
#include <time.h>
clock_t startTime, endTime;
startTime = clock();
for(int i = 0; i < NUM_ITERS; i++)
   programUnderTest();
endTime = clock();
double totalTimeUsed = ((double)(endTime - startTime))/
(CLOCKS_PER_SEC);
double cpuTime = totalTimeUsed/NUM_ITERS;
```

Example : Kosaraju-Sharir CPU time

Input Size (V)	CPU time
10	95.634/200
15	104.414/200
20	150.823/200
30	199.511/200
40	280.144/200
60	404.336/200

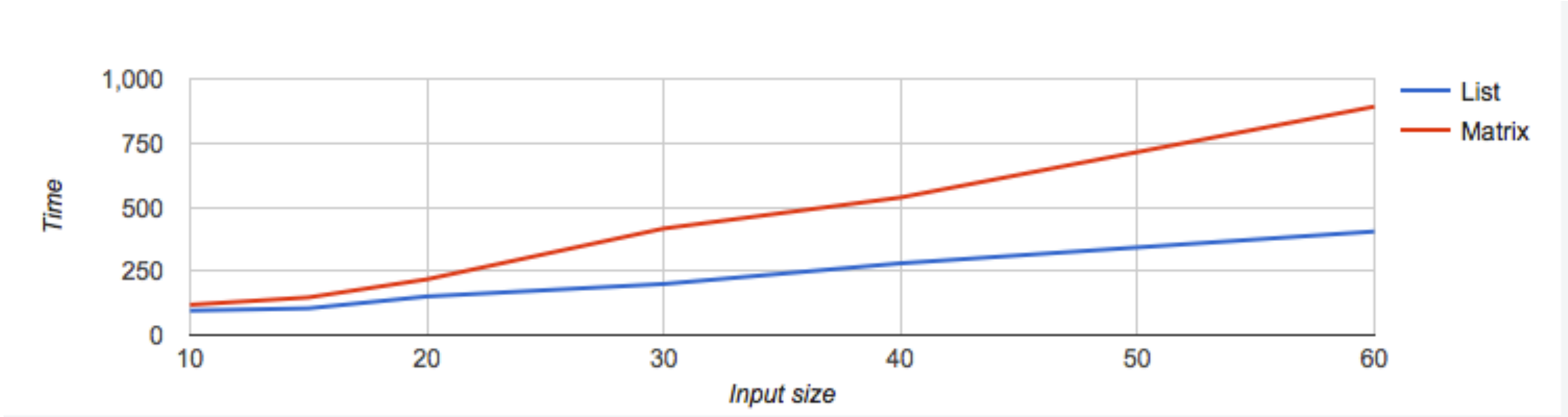
Adjacency list

Input Size (V)	CPU time
10	118.218/200
15	146.964/200
20	217.674/200
30	415.984/200
40	537.106/200
60	895.68/200

Adjacency matrix

Kosaraju-Sharir CPU time plot

Input size : number of vertices
Time : total CPU time for 200 cases



Data analysis

1. Plot program execution time as a function of input
2. Use curve - fitting to find the equation that fits the curve
3. Tools :
    - NUMPY (Python)
    - MATLAB
    - Excel

Asymptotic analysis

- Evaluate performance time/space in terms of input size
- Does not measure actual running time
- Drawbacks -
  - May give better/worse estimate of performance than observed in practice
  - No information on actual running time

Paul Bachmann

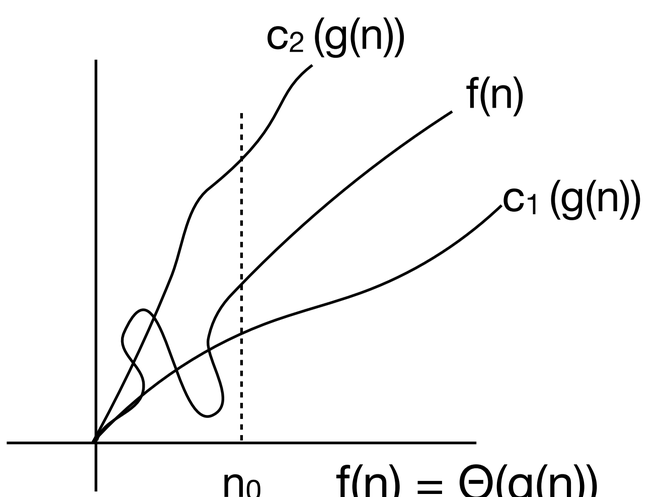
Edmund Landau

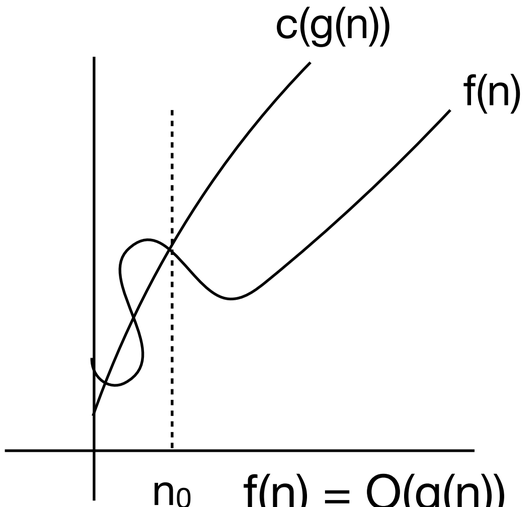
Image Source : Paul Bachmann,[1] Edmund Landau,[2]

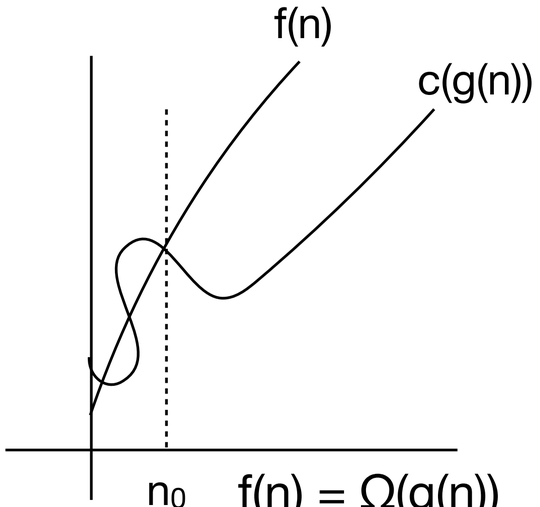
Machine independence

- Time complexity: how many basic operations are performed by algorithm.
  - expressed as a function of input size.
- Space complexity: how much memory is needed by the algorithm

Graphs of functions







Commonly used notations

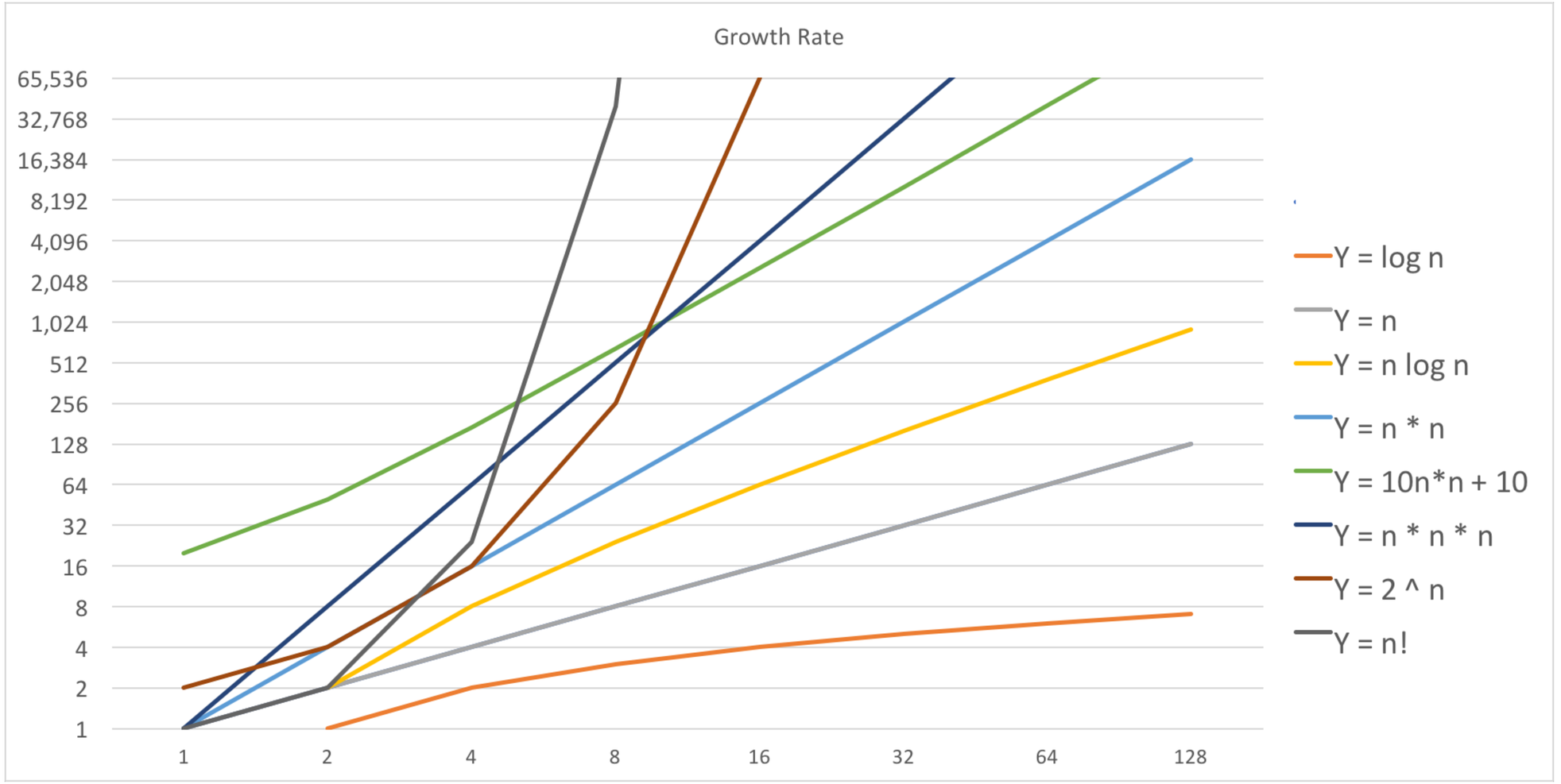
Notation	Description	Example	
O (Big - oh)	Upper bound	O(g(n))	$0 \leq f(n) \leq c_1 g(n)$
o (Little - oh)	Weaker upper bound	o(g(n))	$0 \leq f(n) < c_1 g(n)$
Ω (Big - omega)	Lower bound	Ω(g(n))	$0 \leq c_1 g(n) \leq f(n)$
ω (Little - omega)	Weaker lower bound	ω(g(n))	$0 \leq c_1 g(n) < f(n)$
Θ (Theta)	Tight upper and lower bound	Θ(g(n))	$0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$

O : “at most” Ω : “at least” Θ : “equal”
Note : n_0 , c_1 , c_2 , are positive and $n \geq n_0$

Time complexity classes

Class	Example
Constant	1
Iterated log	log*n
Log logarithmic	log log n
Logarithmic	log n
Poly logarithmic	$(\log n)^c$
Linear	n
Linearithmic	n log n
Quadratic	n^2
Cubic	n^3
Polynomial	n^c c >=1
Sub exponential	2^{n^ε} 0<ε <1
Exponential	c^n c>1
Factorial	n!
n to the n	n^n

Growth of functions



Power law

- A straight line in a log-log plot suggests that the data fits the equation:

$$T(n) = a\, n^b$$

Best case, worst case, average case

- Best case: inputs are such that algorithm completes in the fastest time.
- Worst case: inputs are such that algorithm takes longest time to complete.
- Average case: uses typical set of inputs for average performance.

Example 1

Show that $\frac{n^2}{5} - 2n = \Theta(n^2)$ where $n_0, c_1, c_2 > 0$

Solution :

To prove $c_1 n^2 \leq \left(\frac{n^2}{5} - 2n\right) \leq c_2 n^2 \quad n \geq n_0$

Example 1

Lower Bound : $c_1 n^2 \leq \left(\frac{n^2}{5} - 2n\right)$

$$c_1 \leq \left(\frac{1}{5} - \frac{2}{n}\right) \qquad \text{.....(1)}$$

Choose $n_0 = 12 \Rightarrow c_1 \leq \left(\frac{1}{5} - \frac{1}{6}\right) \leq \frac{1}{30}$

Select c_1 = 1/100. Now (1) is always true for all $n >= n_0$.

Therefore, $\frac{n^2}{5} - 2n = \Omega(n^2)$ .....(2)

Example 1

Upper Bound : $\left(\frac{n^2}{5} - 2n\right) \leq c_2 n^2$

$$=> \left(\frac{1}{5} - \frac{2}{n}\right) \leq c_2 \quad \text{.....(3)}$$

Use the same value of n_0 ($n_0 = 12$) to compute c_2

$$=> \left(\frac{1}{5} - \frac{1}{6}\right) \leq c_2 \quad => \frac{1}{30} \leq c_2 \,.$$

Choose $c_2 = 1$, (3) is always true for $n \geq n_0$ $\quad => \left(\frac{n^2}{5} - 2n\right) = O(n^2)$ .....(4)

$$=> \left(\frac{n^2}{5} - 2n\right) = \Theta(n^2)$$ (from (2) and (4))

Example 2

Determine if $\frac{n^2}{5} - 2n = \Theta(n)$ where $n_0, c_1, c_2 > 0$

Solution :

To prove $c_1 n \leq \left(\frac{n^2}{5} - 2n \right) \leq c_2 n \quad n \geq n_0$

Example 2

Lower Bound : $c_1 n \leq \left(\frac{n^2}{5} - 2n\right)$

$$c_1 \leq \left(\frac{n}{5} - 2\right)$$

Choose $n_0 = 15 => c_1 \leq 1,\ set\ c_1 = 1$

Now as *n* is increased $n \geq n_0$ this condition is still true $1 \leq \frac{n}{5} - 2$

$$=> \frac{n^2}{5} - 2n = \Omega(n)$$

Example 2

Upper Bound: $\left(\frac{n^2}{5} - 2n\right) \leq c_2 n$

$$\left(\frac{n}{5} - 2\right) \leq c_2$$

Set n_0 (n_0 = 15) to compute c_2 => $\left(\frac{15}{5} - 2\right) \leq c_2 => 1 \leq c_2$

Choose $c_2 = 1$

Increase n say n=20 $n \geq n_0$, Is condition always true ? No $\frac{20}{5} - 2 \leq 1 \; is \; false$

We cannot find the constant c_2 for which the condition will be true for all $n \geq n_0$

Exercise

Determine if :

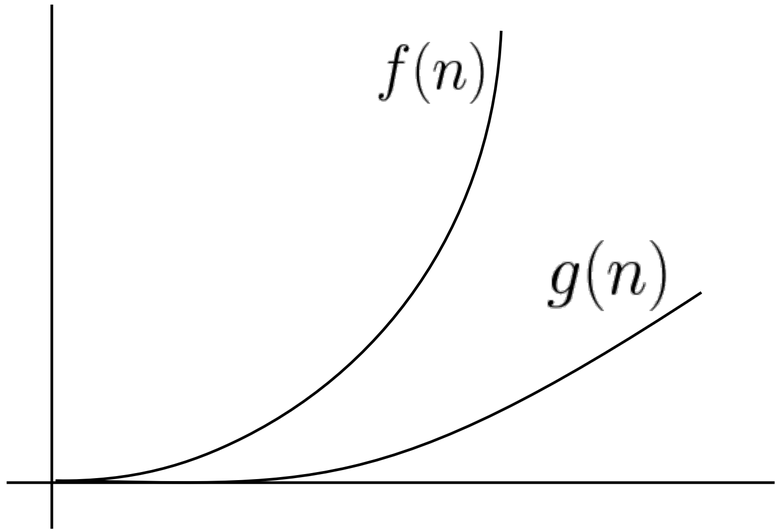
1. $10n + 15 \; is \; O(n)$
2. $10n + 15 \; is \; o(n)$
3. $10n + 15 \; is \; o(n^2)$
4. $3n^2 + \sqrt{n} = O(n^2)$

Using limits to compare functions

Ratio $\frac{f(n)}{g(n)}$ approaches ∞

when n becomes larger

=> $f(n)$ grows faster than $g(n)$



Using limits to compare functions

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty => f(n) = \omega(g(n))$$

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0 => f(n) = o(g(n))$$

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = c \ \ where\ 0 < c < \infty => f(n) = \Theta(g(n))$$

Example with limits

Determine if $2^n = O(n^2)$

Evaluate $\lim_{n\to\infty} \frac{2^n}{n^2}$

$= \lim_{n\to\infty} \frac{2^n ln\ 2}{2n}$ (L'Hospital's Rule)

$= \lim_{n\to\infty} \frac{ln\ 2\ (2^n\ ln\ 2)}{2}$ (L'Hospital's Rule)

$= \infty$

$2^n = \omega(n^2)$

Theorems

- For two functions f(n) and g(n),

  we have f(n) = Θ (g(n)) iff f(n) = O(g(n)) and f(n) = Ω(g(n))

- K is O(1), where K is an arbitrary constant

- A polynomial is O(term with the highest power of n)

  $f(n) = 5n^3 + 2n + 10$ Is $O(n^3)$

- $k\ f(n)$ Is $O(f(n))$

  $f(n) = 10n^3$ Is $O(n^3)$

Examples

$$2n^3 + n^2 + 3n = \Theta(n^3)$$
$$2n^3 + n^2 + 3n = O(n^3) = \Omega(n^3)$$
$$2n^3 + n^2 + 3n \neq o(n^2)$$
$$2n^3 + n^2 + 3n = o(n^4)$$
$$2n^3 + n^2 = \omega(n)$$

Exercise

Is A O, o, Ω,ω or Θ of B?

Assume that $k \geq 1$, $\in > 0$, $c > 1$ are constants.

A	B	O	Ω	ω	Θ	o
n^k	c^n					
2^n	$2^{\frac{n}{2}}$					
lg(n!)	$lg(n^n)$					

Properties : transitivity

$$f(n) = \Theta(g(n)) \; and \; g(n) = \Theta(h(n))$$
$$=> f(n) = \Theta(h(n))$$

$$f(n) = O(g(n)) \; and \; g(n) = O(h(n))$$
$$=> f(n) = O(h(n))$$

$$f(n) = \Omega(g(n)) \; and \; g(n) = \Omega(h(n))$$
$$=> f(n) = \Omega(h(n))$$

$$Similarly, \; o \; and \; \omega \; exhibit \; transitivity$$

Properties : reflexivity and symmetry

$$Reflexivity:$$
$$f(n) = \Theta(f(n))$$
$$f(n) = O(f(n))$$
$$f(n) = \Omega(f(n))$$

$$Symmetry:$$
$$f(n) = \Theta(g(n)) \ if \ and \ only \ if \ g(n) = \Theta(f(n))$$

Properties : transpose symmetry

$$f(n) = O(g(n)) \ if \ and \ only \ if \ g(n) = \Omega(f(n))$$

$$f(n) = o(g(n)) \ if \ and \ only \ if \ g(n) = \omega(f(n))$$

Mathematical expressions

- Can be used to analyze running time
    - Logarithms
    - Exponentials
    - Polynomials
    - Factorials
    - Fibonacci

Iterated logarithm

Grows very slowly

$$lg^*n = \begin{cases} 0 & if\ n \leq 1 \\ 1 + lg^*(lg\ n) & if\ n > 1 \end{cases}$$

$$lg^*(4) = 1 + lg^*(lg\ 2^2) = 1 + lg^*(2) = 2$$

Source X	lg* x
(–∞, 1]	0
(1, 2^1]	1
(2, 2^2]	2
(4, 2^4]	3
(16, 65536]	4
(65536, 2^{65536}]	5

2^{65536} is greater than number of atoms in universe

Table Source : Iterated logarithm

Logarithms

$e^y = x$:

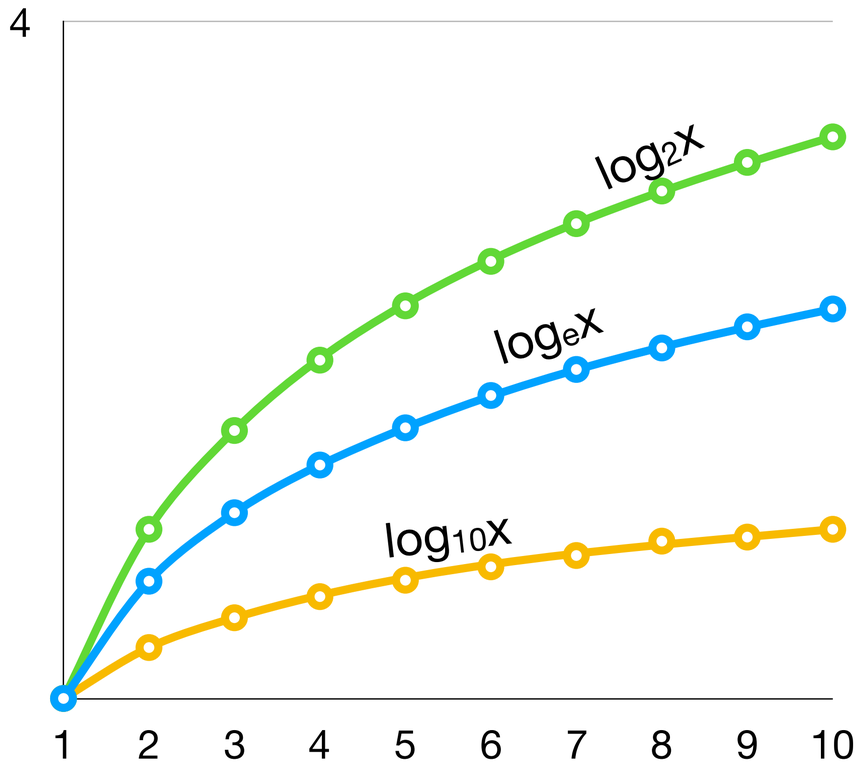
$ln(x) = log_e x = y,\ where\ e = 2.718$

Base -10 log :

$$log(x) = log_{10}(x) = \frac{ln\ x}{ln\ 10}$$

Base -2 log:

$$lg(x) = \frac{ln\ x}{ln\ 2}$$



Logarithms properties

$$log_c(ab) = log_c(a) + log_c(b)$$

$$log_b(a) = \frac{log_c(a)}{log_c(b)}$$

$$log_b(a^n) = n\ log_b(a)$$

$$log_c\left(\frac{a}{b}\right) = log_c(a) - log_c(b)$$

$$log_c\left(\frac{1}{b}\right) = -\ log_c(b)$$

$$(logarithm\ base\ \neq 1,\ in\ these\ equations)$$

Harmonic numbers

$H_n : n^{th}\ harmonic\ number$

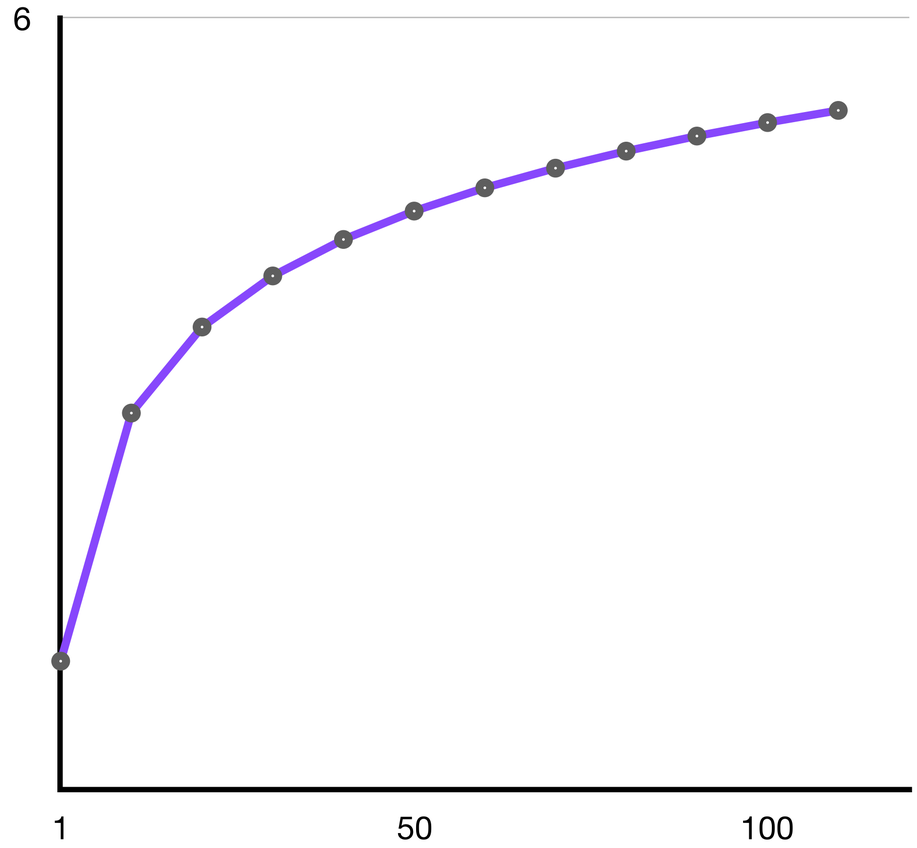
$$1 + \frac{1}{2} + \frac{1}{3} + ... + \frac{1}{n} = \frac{1}{n!}\begin{bmatrix} n+1 \\ 2 \end{bmatrix}$$

$$H_1 = 1$$

$$H_2 = 1 + \frac{1}{2}$$

$$H_3 = 1 + \frac{1}{2} + \frac{1}{3} = 1.8333..$$

$$H_n = ln(n) + \gamma + \frac{1}{2n}\ \ (for\ large\ n)$$



Functional iteration

$$f^{(i)}(n) = \begin{cases} n & i = 0 \\ f(f^{(i-1)}(n)) & i > 0 \end{cases}$$

$$Example \ :$$
$$if \ f(n) = 5n \ then \ f^{(i)}n = 5^i n$$

Fibonacci numbers

$$F_0 = 0$$
$$F_1 = 1$$
$$F_n = F_{n-1} + F_{n-2} \quad for \;\; n \geq 2$$

$$sequence \; 0, 1, 1, 2, 3, 5, 8, 13...$$

$$For \; large \; n \;\; => \;\; F_n \sim= \frac{\Phi^n}{\sqrt{5}}$$

$$\Phi = \frac{1+\sqrt{5}}{2} \approx 1.61803.. \;\; (\Phi = golden \; ratio)$$

$$F_{i+2} \geq \Phi^i \;\; for \; i \geq 0$$

Factorials

$$0! = 1$$
$$n! = n(n-1)!\ for\ n > 0$$
$$for\ large\ n,\ n! = \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$$

$$n! = o(n^n)$$
$$n! = \omega(2^n)$$
$$lg(n!) = \Theta(n\ lg(n))$$

Series sums

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

$$1+x+x^2+...+x^n = \frac{x^{n+1}-1}{x-1}$$

$$\sum_{k=0}^{\infty} x^k = \frac{1}{1-x}$$

$$\sum_{k=0}^{\infty} kx^k = \frac{x}{(1-x)^2} \; for \;\; |x| < 1$$

$$\sum_{k=1}^{n} \frac{1}{k} = 1 + \frac{1}{2} + ... + \frac{1}{n-1} + \frac{1}{n} \; = ln(n) + O(1)$$

$$\sum_{k=1}^{n} a_k - a_{k-1} = \ddot{(}a_n - a_{n-1}) + (a_{n-1} - a_{n-2}) + ... + (a_1 - a_0) = a_n - a_0$$

Recurrence

Define a sequence such that next term is a function of previous terms

Algorithm	Running Time
Binary Search (Worst case)	$T(n) = T\left(\frac{n}{2}\right) + 1$
Quick Sort (Worst case)	$T(n) = T(n-1) + \Theta(n)$
Quick Sort (Best case)	$T(n) = T\left(\frac{n}{2}\right) + \Theta(n)$
Merge Sort (All cases)	$T(n) = 2T\left(\frac{n}{2}\right) + \Theta(n)$

Fibonacci recurrence

$$f(0) = 1$$
$$f(1) = 1$$
$$f(n) = f(n-1) + f(n-2) \quad for\ n > 1$$

Example: Recurrence relation of a simple loop

```
int n;
for(i=0; i<n; i++){
   do_something;
}
```

The statement “do_something” is executed c_n =n times, where c_n

If we change n to n+1 the statement is executed one more time :$T_{n+1} = T_n+1$

need initial conditions

Example: Recurrence relation of a simple loop

- Need n-1 initial condition with n unknowns
- Find the running time for initial values :

Set n = 1 in loop :

```
for(i=0; i<1; i++){
   do_something;
}
```

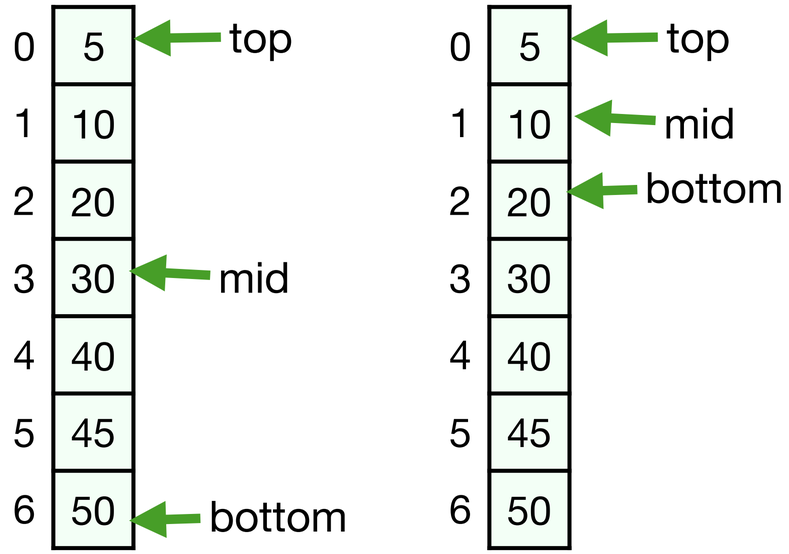
T_1 = 1 (do_something is executed once)

Recurrence relation: $T_1 = 1$ and $T_{n+1} = T_n + 1$

Example : Binary search

- Search by repeatedly dividing array in half

Find 10 in sorted array



Binary search pseudocode

```
binary_search(array, item, size){
   top = 0;
   bottom = size-1;
   while(top <= bottom){
      mid = (top + bottom)/2;
      if(array[mid]==item)  // check if item is at index mid
        return mid;
      if(item < array[mid]) // check top half next
        bottom = mid -1;
      else
        top = mid+1;          // check bottom half next
    }
}
```

Binary search : Recurrence relation

$T(n)$ = computation time to search n elements

= computation time to search $\left(\frac{n}{2}\right)$ elements + *constant*

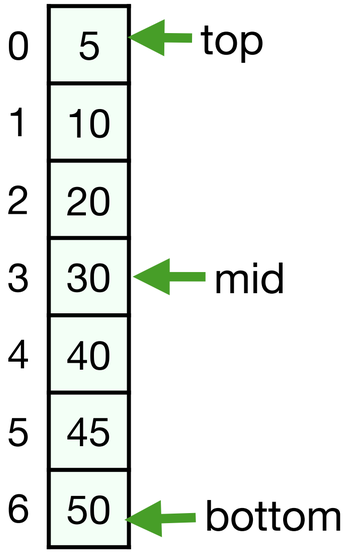
= $T\left(\frac{n}{2}\right)$ + constant

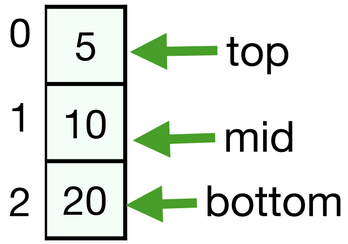
Find initial condition :

- Two unknown variance $T(n)$, $T\left(\frac{n}{2}\right)$ and so there is one initial condition

$T(1)$ = constant

Binary search : Recurrence relation

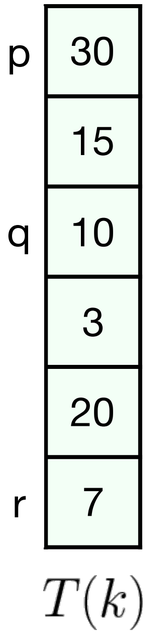




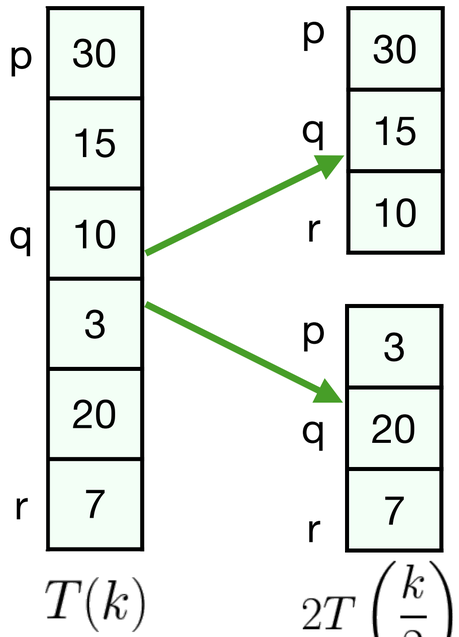
$$T(n) = T\left(\frac{n}{2}\right) + O(1)$$

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

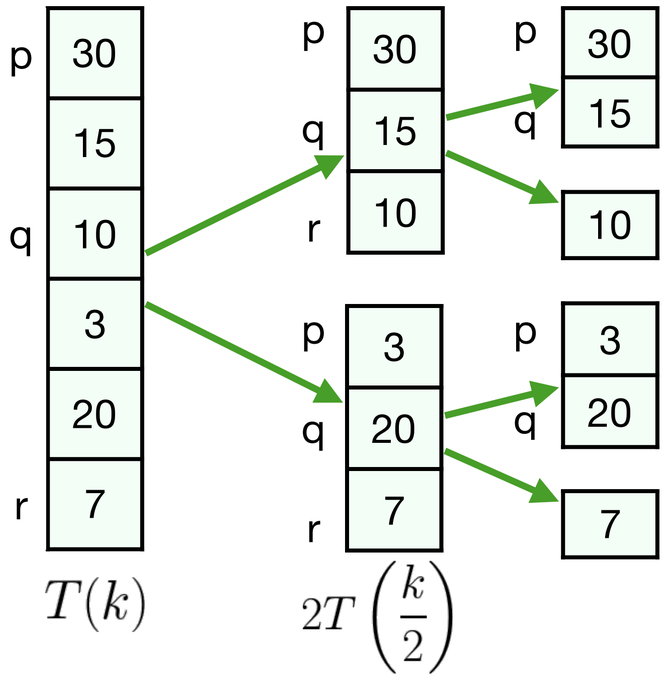
Merge sort



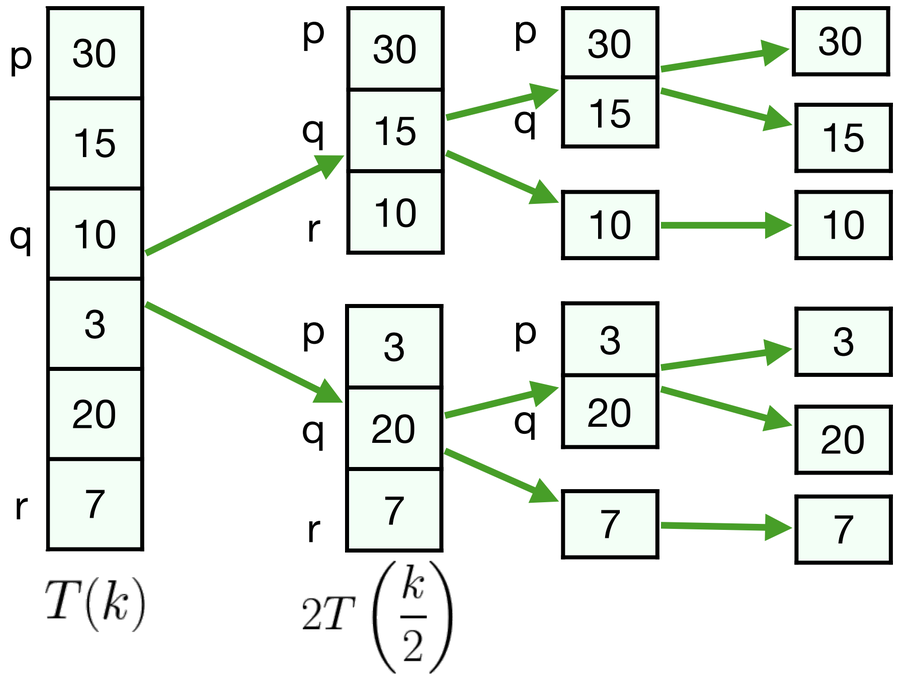
Merge sort



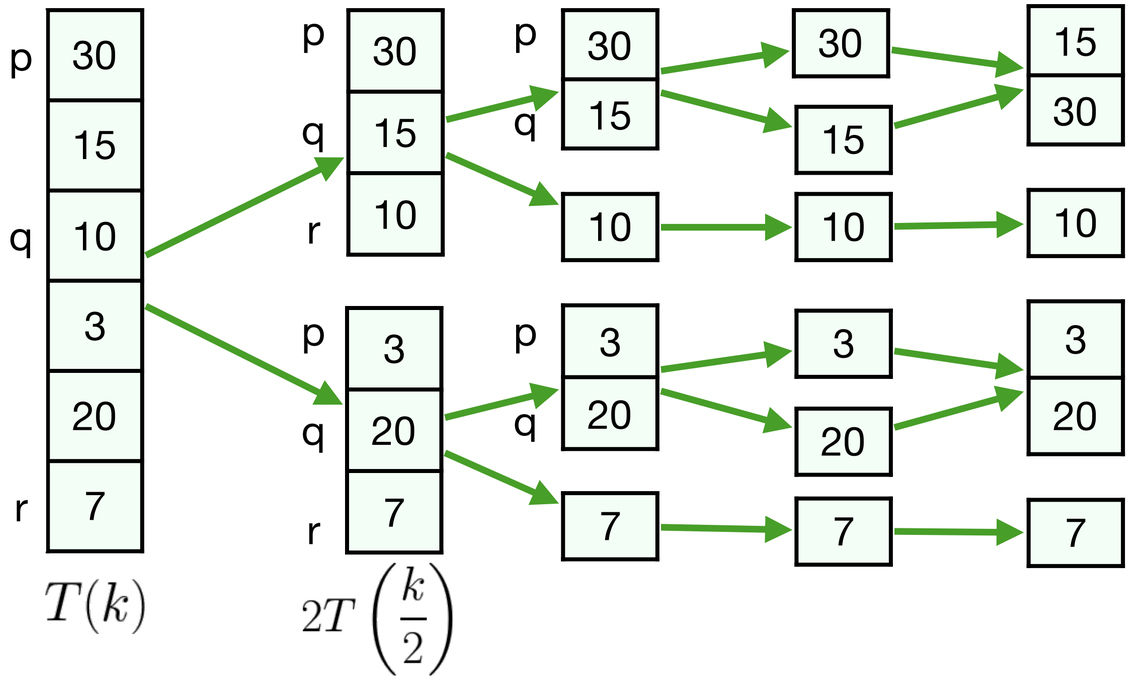
Merge sort



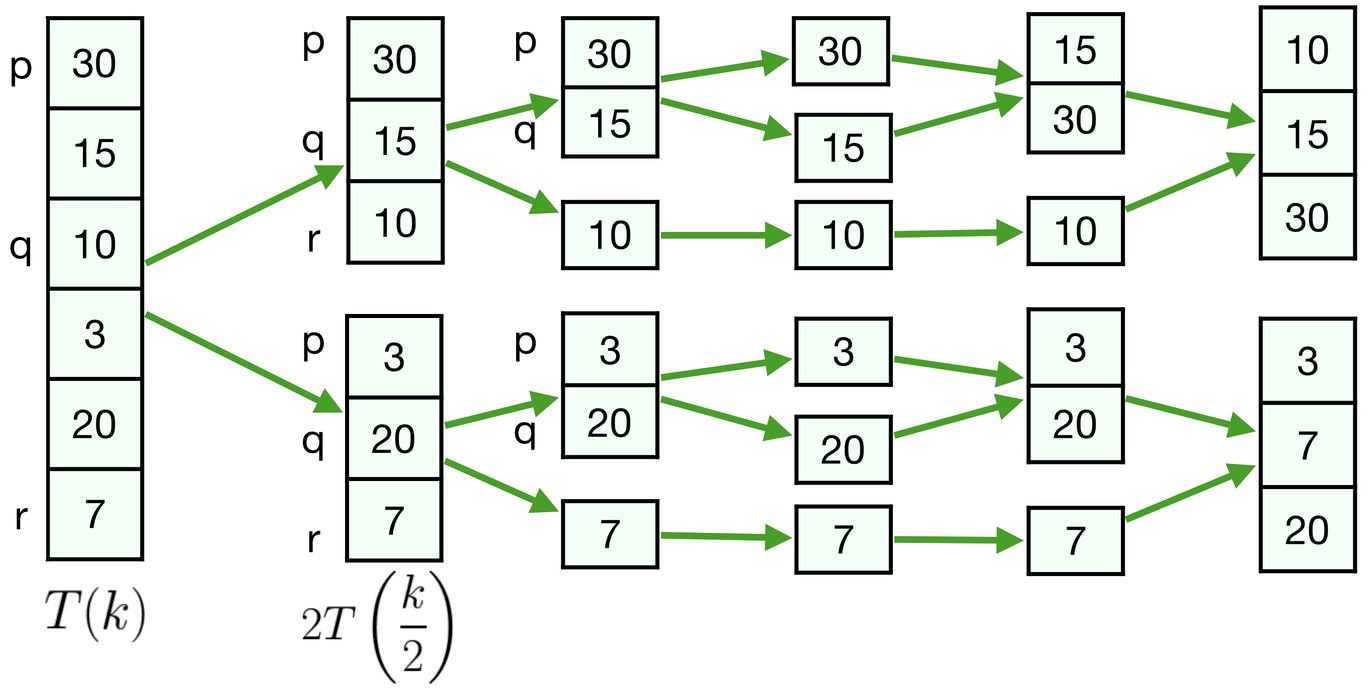
Merge sort



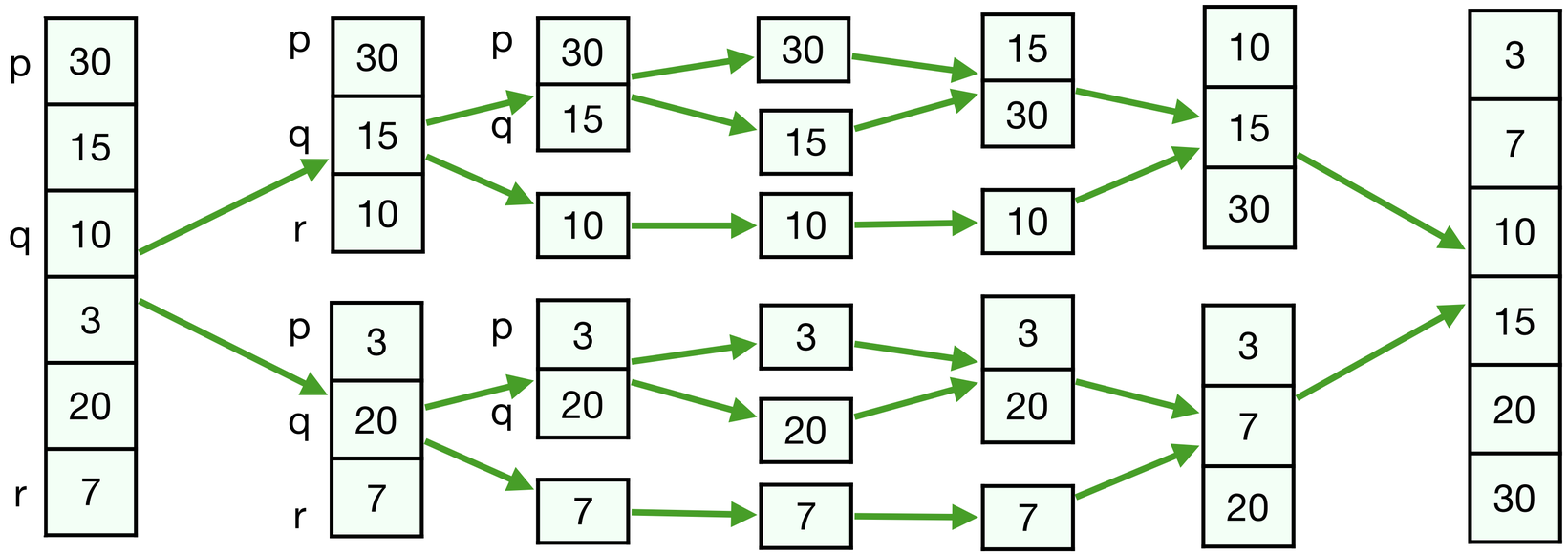
Merge sort



Merge sort



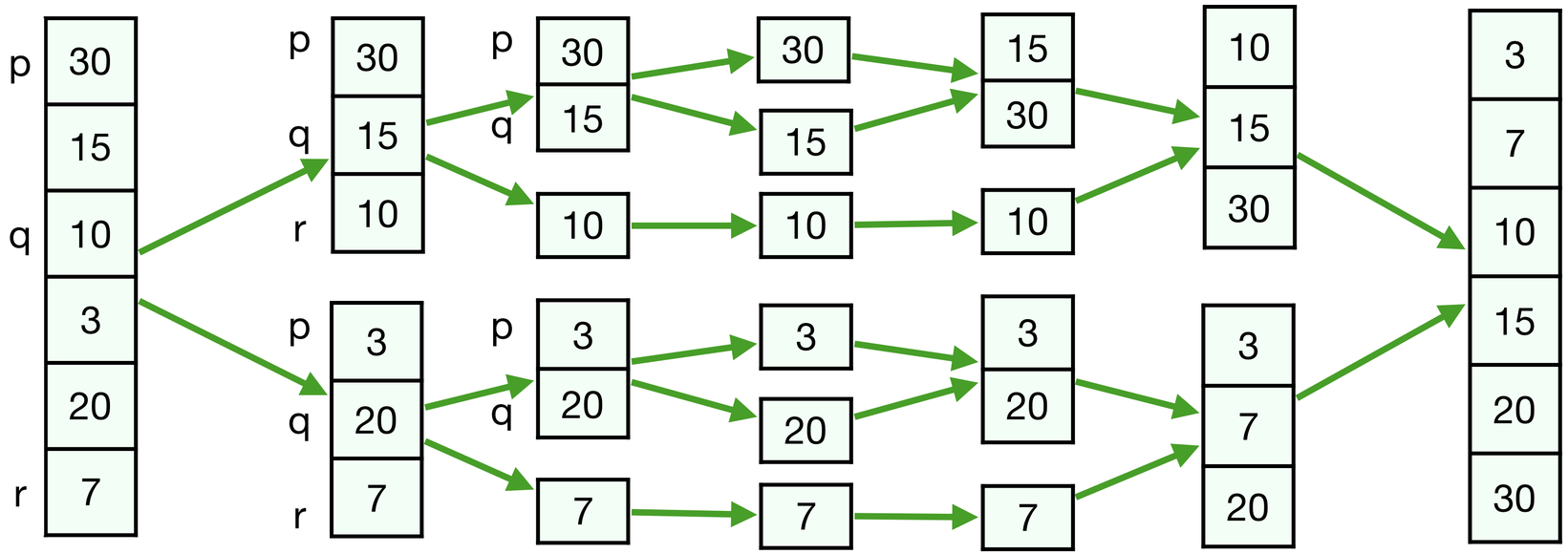
Merge sort



# of comparisons to merge (worst case) : *k*

merge(best case) : $\frac{k}{2}\ where\ k \leq n$

Merge sort



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

Merge sort algorithm

```
mergeSort(array, p, r){
   if(p<r){
    q = (p+r)/2;

    mergeSort(array, p, q);
    mergeSort(array, q+1, r);

    merge(array, p, q, r);
   }
}
```

Merge sort algorithm

```
merge(array, p, q, r){
    n1 = q-p+1;    n2 = r-q;
    create temporary arrays L and R with sizes n1 and n2; copy data to L and R
    for(i=0 to n1-1)
       L[i] = array[i+p];
    for(j=0 to n2-1)
       R[j] = array[q+j+1];
    // merge L and R back to array
    i=0;  j = 0;
    for( k = p to r){
      if(L[i]<= R[j]) {
        array[k] = L[i];
        i++;
      }
      else{
        array[k] = R[j];
        j++;
      }
     }
}
```

Merge sort : Recurrence relation

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

Two unknowns => one initial condition

Set n =1

$T(1) = 1$

$$T(n) = \begin{cases} \Theta(1) \\ 2T\left(\frac{n}{2}\right) + \Theta(n) \end{cases}$$

Solving Recurrences

- Back-substitution method
- Induction method
- Master’s Theorem
- Recurrence tree method

Back-substitution

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

$T(1) = 1$

Solution : Expand T(n) in terms of smaller n (can draw a tree)

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

Substitute : $T\left(\frac{n}{2}\right)$ in *T*(n)

$$T\left(\frac{n}{4}\right) = 4\left[4T\left(\frac{n}{4}\right) + \left(\frac{n}{2}\right)^2\right] + n^2 = 16T\left(\frac{n}{4}\right) + 2n^2$$

Back-substitution

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

Substitute $T\left(\frac{n}{4}\right)$ in $T(n)$

$$T(n) = 16\left[4T\left(\frac{n}{8}\right) + \left(\frac{n}{4}\right)^2\right] + 2n^2$$

$$= 64T\left(\frac{n}{8}\right) + 3n^2$$

Substitute $T\left(\frac{n}{8}\right) = 4T\left(\frac{n}{16}\right) + \left(\frac{n}{8}\right)$ in $T(n)$

$$T(n) = 64\left[4T\left(\frac{n}{16}\right) + \left(\frac{n}{8}\right)^2\right] + 3n^2 = 256T\left(\frac{n}{16}\right) + 4n^2$$ $= k^2T(n/k) + n^2 \lg k$

When $k = n$, $T(n) = n^2T(1) + n^2 \lg n = \theta(n^2 \lg n)$

Induction to prove recurrence method - example

Given that $T(n) = 4T\left(\frac{n}{2}\right) + n^2$, T(1) = 1

Prove that $T(n) = \Omega(n^2 \ lg \ n)$

Proof :

To Prove $T(n) \geq cn^2 \ lg \ n$

Base case : $T(1) = 1^2 \geq c1^2 \lg \ 1 \geq 0$ true

Hypothesis : Assume that $T(m) \geq cm^2 \ lg \ m \quad \forall \ m < n$

Then $T\left(\frac{n}{2}\right) \geq c\left(\frac{n}{2}\right)^2 \ lg \ \left(\frac{n}{2}\right)$

Induction to prove recurrence method - example

Induction :

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

$$\geq 4c\left(\frac{n}{2}\right)^2 \lg\ \left(\frac{n}{2}\right) + n^2\ from\ (2)$$

$$\geq cn^2 \lg(n) - cn^2 lg\ 2 + n^2$$

$$\geq cn^2 \lg(n) + n^2(1-c)$$

$$\geq cn^2 \lg(n)\ for\ c < 1$$

Exercise

Prove that $T(n) = O(n^2 \ lg \ n)$

Master - theorem

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

Where a ≥1 and b > 1 are constants $f(n)$ is a function

then

1. $T(n) = \Theta\left(n^{log_b a}\right)\ if\ f(n) = O\left(n^{log_b a - \varepsilon}\right)\ for\ \varepsilon > 0$
2. $T(n) = \Theta\left(n^{log_b a}\ lg\ n\right)\ if\ f(n) = \Theta\left(n^{log_b a}\right)$
3. $T(n) = \Theta\left(f(n)\right)\ if\ f(n) = \Omega\left(n^{log_b a + \varepsilon}\right)\ for\ \varepsilon > 0\ and\ af\left(\frac{n}{b}\right) \leq cf(n)\ for\ c < 1$

Example - master theorem to solve recurrence

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

Solution :

$$f(n) = n^2 = \Theta(n^2)$$

use (2) of master theorem

$$f(n) = \Theta\left(n^{log_2 4}\right) = \Theta(n^2)$$

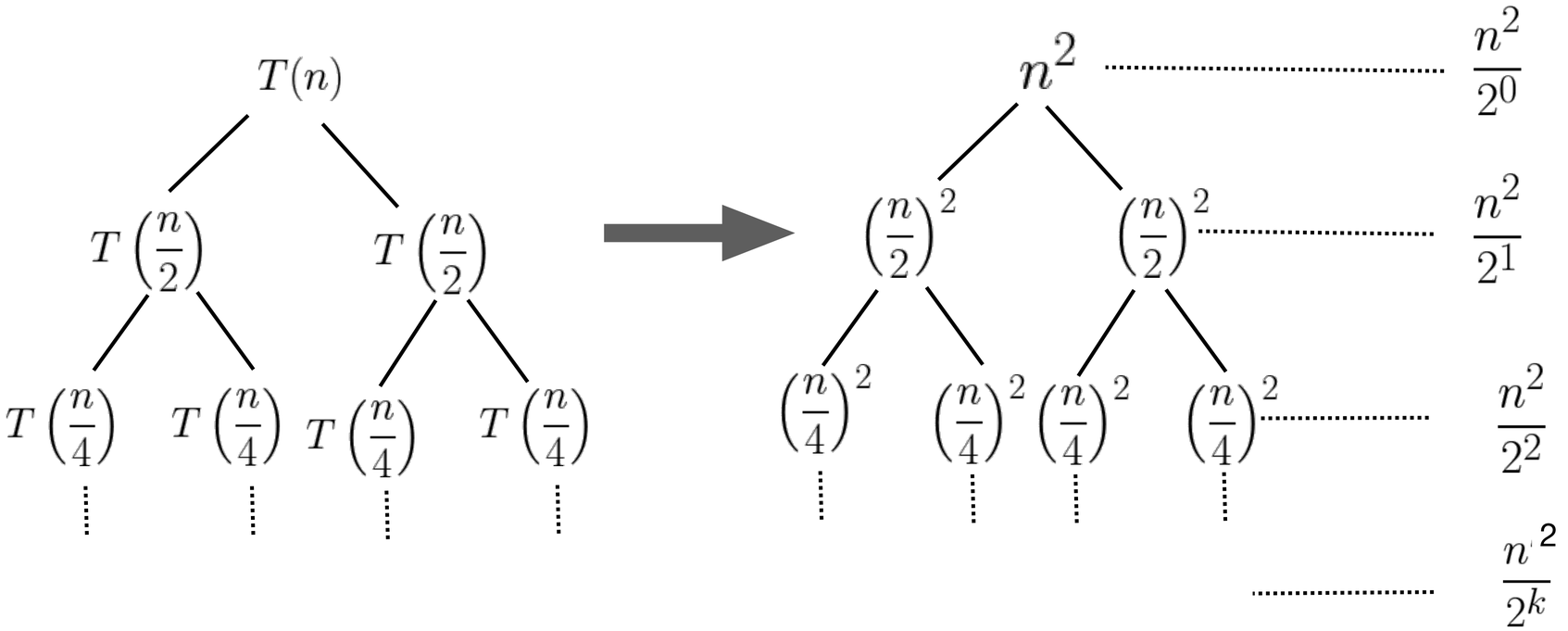
$$T(n) = \Theta\left(n^{log_2 4} \, lg \, n\right)$$

$$= \Theta\left(n^2 \, lg \, n\right)$$

Recursion tree

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

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



Recursion tree - total cost

$$Total\ cost = \sum_{i=0}^{lg\ n} \frac{n^2}{2^i} < n^2 \sum_{i=0}^{\infty} \left(\frac{1}{2}\right)^i$$

$$< n^2 \left[\frac{1}{1-\frac{1}{2}}\right]$$

$$< 2n^2$$

$$= O(n^2)$$

Merge sort - worst case

No comparisons are skipped in merge step => n -1 comparison

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

$$T(n) = 8T\left(\frac{n}{8}\right) + 3n - 7$$

$$= 2^k T\left(\frac{n}{2^k}\right) + kn - (2^k - 1)$$

$$2^k = n => k = lg\ n$$

$$T(n) = nT(1) + n \lg\ n - n + 1$$

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

Merge sort - best case

Largest element of one list is smaller than first element of other list => $\frac{n}{2}$ comparison

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

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

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

$$2^k = n => k = lg\ n$$

$$T(n) = nT(1) + \frac{n\ lg\ n}{2} \quad => \quad T(n) = O(n\ lg\ n)$$

Merge sort - average case

- Calculate the expected number of comparisons for a list of size n.
- Substitute in the recurrence relation and solve.
- See next lecture for detailed analysis.

Methods to bound summations

- Mathematical induction
- Bounding the terms
- Splitting summations
- Approximation by integrals

Mathematical induction

Prove that the n^{th} Fibonacci number f_n is $O\left(\left(\frac{7}{4}\right)^n\right)$

Proof :

base: $n = 0,\ f_0 = 1 \leq \left(\frac{7}{4}\right)^0$ which is true

$n = 1,\ f_0 = 1 \leq \left(\frac{7}{4}\right)^1$ which is true

hypothesis : assume that $f_i \leq \left(\frac{7}{4}\right)^i$ i = 1, 2, 3, … k

Mathematical induction

Induction : To prove $f_{i+1} \leq \left(\frac{7}{4}\right)^{i+1}$

$$f_{i+1} = f_i + f_{i-1}$$

$$\leq \left(\frac{7}{4}\right)^{i} + \left(\frac{7}{4}\right)^{i-1} \text{ - hypothesis}$$

$$\leq \left(\frac{7}{4}\right)^{i-1} \left(1 + \frac{7}{4}\right)$$

$$\leq \left(\frac{7}{4}\right)^{i-1} \left(\frac{7}{4}\right)^{2} \left[1 + \tfrac{7}{4} < \left(\tfrac{7}{4}\right)^{2}\right]$$

$$\leq \left(\frac{7}{4}\right)^{i+1}$$

Bounding the terms

Bounding by largest term in series

$$(a) \quad \sum_{k=1}^{n} a_k \leq n \; a_{max}$$

where $a_{max} = \max_{1 \leq k \leq n} a_k$

Bounding the terms

Bounding by a geometric series

$$(b) \sum_{k=0}^{n} a_k \leq \frac{a_0}{1-r}$$

when $\frac{a_{k+1}}{a_k} \leq r \quad for\ all\ k \geq 0$ and 0< r < 1 and r is constant

Note :

- Geometric series has a constant ratio between successive terms
- Geometric series are infinite series but have finite sums when ratio r is between -1 and 1

Splitting summations

Split the summation and ignore a constant number of initial terms

$$\sum_{k=0}^{n} a_k = \sum_{k=0}^{k-1} a_k + \sum_{k=k_0}^{n} a_k$$

$$= \Theta(n) + \sum_{k=k_0}^{n} a_k$$

Splitting summations - example

Find an asymptotic upper bound on $\sum_{k=0}^{\infty} \frac{k^2}{2^k}$

find the ratio of consecutive terms

Ratio $= \frac{(k+1)^2 / \ 2^{k+1}}{k^2 / \ 2^k} = \frac{(k+1)^2}{2k^2}$

$= \frac{(k^2+2k+1)}{2k^2} = \left(\frac{1}{2} + \frac{1}{k} + \frac{1}{2k^2}\right) \leq \frac{8}{9}$

$k = 3 => \left(\frac{1}{2} + \frac{1}{3} + \frac{1}{18}\right) = \frac{8}{9}$

$k = 4 => \left(\frac{1}{2} + \frac{1}{4} + \frac{1}{32}\right) = \frac{25}{32}$

Splitting summations - example

$$\sum_{k=0}^{\infty} \frac{k^2}{2^k} = \sum_{k=0}^{2} \frac{k^2}{2^k} + \sum_{k=3}^{\infty} \frac{k^2}{2^k}$$

$$= \left[0 + \frac{1}{2} + \frac{4}{2^2}\right] + \left[\frac{3^2}{2^3} + \frac{4^2}{2^4} + \frac{5^2}{2^5} + \ldots\right]$$

$$<= \left[0 + \frac{1}{2} + \frac{4}{2^2}\right] + \frac{9}{8}\left[1 + \frac{8}{9} + \left(\frac{8}{9}\right)^2 + \ldots\right]$$

$$= O(1)$$

Calculus

$$f(x) = a^x \qquad f'(x) = a^x \ln\ a$$

$$f(x) = ln\ x \qquad f'(x) = \frac{1}{x}$$

$$f(x) = log_a x \qquad f'(x) = \frac{1}{x\ ln\ a}$$

$$f(x) = x^n \qquad f'(x) = nx^{n-1}$$

$$f(x) = e^x \qquad f'(x) = e^x$$

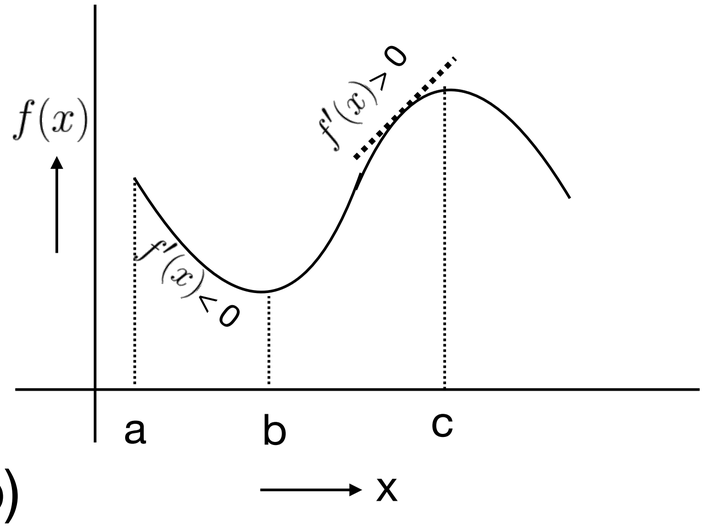
Test for monotonic function

$f'(x) > 0$ ∀ x in (b, c)

=> $f(x)$ is monotonically increasing in (b, c)

$f'(x) < 0$ ∀ x in (a, b)

=> $f(x)$ is monotonically decreasing in (a, b)

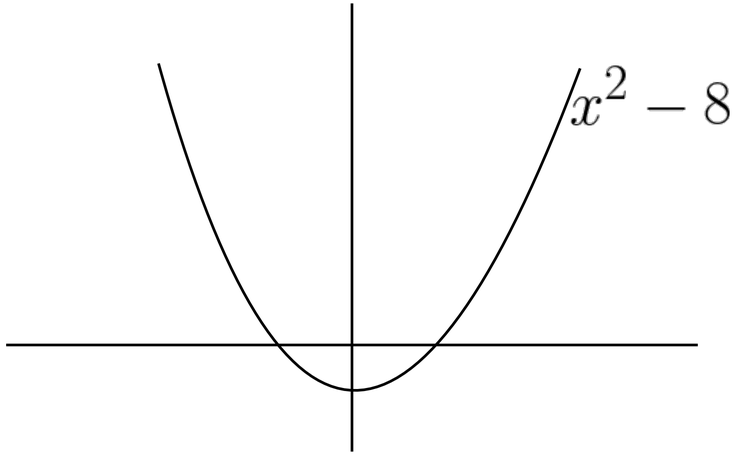


Test for monotonic function : example

Given $f(x) = x^2 - 8 \quad f'(x) = 2x$

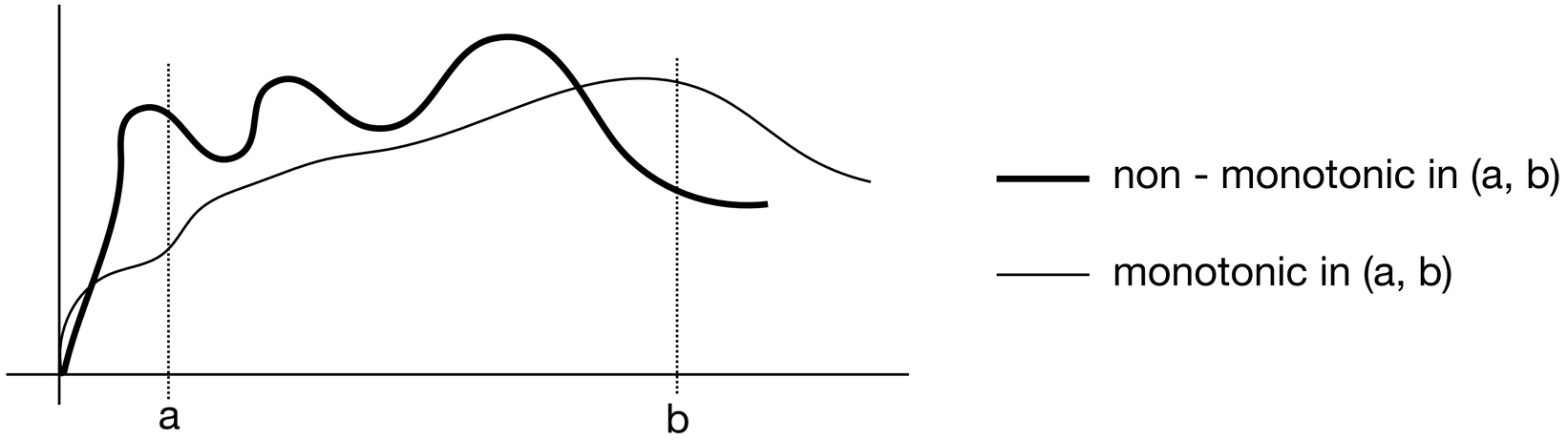
$f'(x) < 0$, when x is negative

$f'(x) > 0$, when x is positive



Approximation by integrals : monotonic function

- Function is monotonic on an interval
- If it is either increasing or decreasing in that interval
- Monotonic functions are integrable



Approximation by integrals : monotonic function

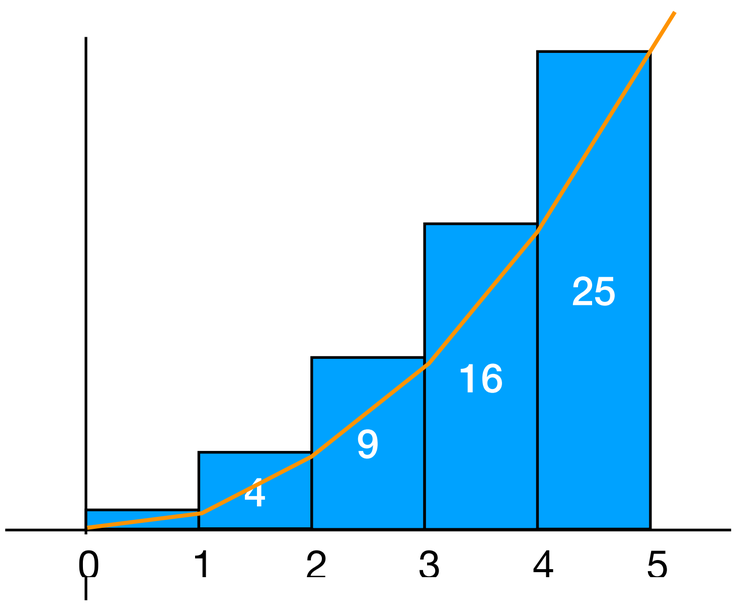
- Monotonically increasing : $x \leq y => f(x) \leq f(y)$
- Monotonically decreasing : $x \geq y => f(x) \geq f(y)$

Approximation by integrals

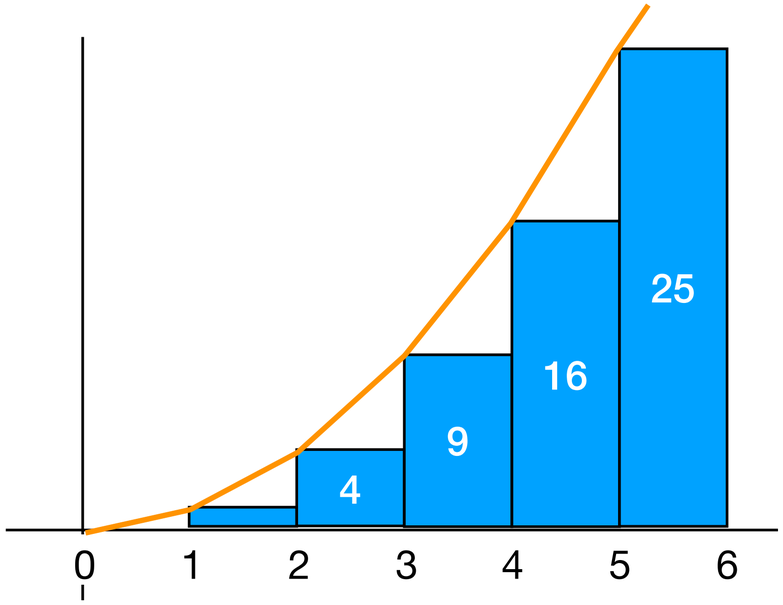
$$\int_0^n f(x)\,dx \leq \sum_{i=1}^{n} f(i) \leq \int_1^{n+1} f(x)\,dx$$

$$\int_0^n x^2\,dx \leq \sum_{i=1}^{n} i^2 \leq \int_1^{n+1} x^2\,dx \quad = \quad \frac{x^3}{3}\,\Big|_{x=1}^{n+1} \quad = \frac{(n+1)^3}{3} - \frac{1}{3}$$

Approximation by integrals - example



$$\int_0^5 x^2\,dx \leq \sum_{x=1}^{5} x^2$$



$$\sum_{x=1}^{5} x^2 \leq \int_1^6 x^2\,dx$$

References

- Cormen, Thomas H., et al. *Introduction to Algorithms*. The MIT Press, 2014.
- Sedgewick, Robert, and Kevin Wayne. *Algorithms*. Addison-Wesley, 2016.
- https://en.wikipedia.org/wiki/Genetic_algorithm
- http://www.gnu.org/software/libc/manual/html_node/CPU-Time.html
- https://www.newscientist.com/article/mg21328473-800-exhaustive-search-solves-fiendish-sudoku-mystery/
- https://en.wikipedia.org/wiki/Sudoku_solving_algorithms
- https://en.wikipedia.org/wiki/Missionaries_and_cannibals_problem
- Any text on calculus