Design and Analysis of Algorithms, MTech-I (1^{st} semester)

Chapter 1: Algorithm Analysis Techniques

August 20, 2022

Devesh C Jinwala,

Professor in CSE, SVNIT, Surat and Adjunct Professor, IITJammu & Dean (R&C), SVNIT

Department of Computer Science and Engineering, SVNIT, Surat

A Program and an Algorithm

- Defining an algorithm

A Program and an Algorithm

- Defining an algorithm
- named after Persian mathematician al-Khowârizmî

A Program and an Algorithm

- Defining an algorithm
- named after Persian mathematician al-Khowârizmî
- How do we relate a program and an algorithm ?

A Program and an Algorithm

- Defining an algorithm
- named after Persian mathematician al-Khowârizmî
- How do we relate a program and an algorithm ?
- A Program = a set of conventions for communicating an algorithm to two different entities. Which ones ?

A Program and an Algorithm

- Defining an algorithm
- named after Persian mathematician al-Khowârizmî
- How do we relate a program and an algorithm ?
- A Program = a set of conventions for communicating an algorithm to two different entities. Which ones ?
- Therefore, an algorithm should not involve any subjective decisions.

A Program and an Algorithm

- Defining an algorithm
- named after Persian mathematician al-Khowârizmî
- How do we relate a program and an algorithm ?
- A Program = a set of conventions for communicating an algorithm to two different entities. Which ones ?
- Therefore, an algorithm should not involve any subjective decisions.
- What is a subjective decision?

A Program and an Algorithm

- Defining an algorithm
- named after Persian mathematician al-Khowârizmî
- How do we relate a program and an algorithm ?
- A Program = a set of conventions for communicating an algorithm to two different entities. Which ones ?
- Therefore, an algorithm should not involve any subjective decisions.
- What is a subjective decision?
- Tutorial-1: Give at least two different real-life examples of a subjective decision and a non-subjective decision.

Why do we study these topics ? Takeways

- Our goal is to emphasize

Why do we study these topics ? Takeways

- Our goal is to emphasize
  - That the practical success of computer science is also built on elegant & solid foundations.

Why do we study these topics ? Takeways

- Our goal is to emphasize
  - That the practical success of computer science is also built on elegant & solid foundations.
- We study algorithms to answer the questions like

Why do we study these topics ? Takeways

- Our goal is to emphasize
    - That the practical success of computer science is also built on elegant & solid foundations.
- We study algorithms to answer the questions like
    - What is an algorithm ?

Why do we study these topics ? Takeways

- Our goal is to emphasize
  - That the practical success of computer science is also built on elegant & solid foundations.
- We study algorithms to answer the questions like
  - What is an algorithm ?
  - What does solving a problem algorithmically mean?

Why do we study these topics ? Takeways

- Our goal is to emphasize
  - That the practical success of computer science is also built on elegant & solid foundations.
- We study algorithms to answer the questions like
  - What is an algorithm ?
  - What does solving a problem algorithmically mean?
  - What can be and what can't be computed?

Why do we study these topics ? Takeways

- Our goal is to emphasize
  - That the practical success of computer science is also built on elegant & solid foundations.
- We study algorithms to answer the questions like
  - What is an algorithm ?
  - What does solving a problem algorithmically mean?
  - What can be and what can't be computed?
  - Is every algorithm conceived a feasible algorithm ?

Why do we study these topics ? Takeways

- Our goal is to emphasize
    - That the practical success of computer science is also built on elegant & solid foundations.
- We study algorithms to answer the questions like
    - What is an algorithm ?
    - What does solving a problem algorithmically mean?
    - What can be and what can't be computed?
    - Is every algorithm conceived a feasible algorithm ?
    - When can an algorithm be considered feasible ?

Why do we study these topics ? Takeways

- Our goal is to emphasize
  - That the practical success of computer science is also built on elegant & solid foundations.
- We study algorithms to answer the questions like
  - What is an algorithm ?
  - What does solving a problem algorithmically mean?
  - What can be and what can't be computed?
  - Is every algorithm conceived a feasible algorithm ?
  - When can an algorithm be considered feasible ?
  - How can we compare two algorithms ?

Analysis Techniques

Comparing two algorithms

- How can we compare two algorithms ?

Comparing two algorithms

- How can we compare two algorithms ?
- We will need to analyze the performance of two algorithms.

Comparing two algorithms

- How can we compare two algorithms ?
- We will need to analyze the performance of two algorithms.
- Why do we need to analyze the performance of algorithms?

Comparing two algorithms

- How can we compare two algorithms ?
- We will need to analyze the performance of two algorithms.
- Why do we need to analyze the performance of algorithms?
- Performance in terms of what ?

Comparing two algorithms

- How can we compare two algorithms ?
- We will need to analyze the performance of two algorithms.
- Why do we need to analyze the performance of algorithms?
- Performance in terms of what ?
- Algorithm analysis - concerned with demand for resources and performance

Comparing two algorithms

- How can we compare two algorithms ?
- We will need to analyze the performance of two algorithms.
- Why do we need to analyze the performance of algorithms?
- Performance in terms of what ?
- Algorithm analysis - concerned with demand for resources and performance
    - A-priori estimates - Performance analysis

Comparing two algorithms

- How can we compare two algorithms ?
- We will need to analyze the performance of two algorithms.
- Why do we need to analyze the performance of algorithms?
- Performance in terms of what ?
- Algorithm analysis - concerned with demand for resources and performance
    - A-priori estimates - Performance analysis
    - A-posteriori analysis - Measurement & testing

Methods of Analysis

- Mathematical Analysis .... are the efforts put-in worth it ?

Methods of Analysis

- Mathematical Analysis .... are the efforts put-in worth it ?
- Empirical Analysis .....is it feasible ?

Methods of Analysis

- Mathematical Analysis .... are the efforts put-in worth it ?
- Empirical Analysis .....is it feasible ?
- Asymptotic Analysis ...is the way to go.

Methods of Analysis

- Mathematical Analysis .... are the efforts put-in worth it ?
- Empirical Analysis .....is it feasible ?
- Asymptotic Analysis ...is the way to go.
- How to convince ourselves?

Familiarity with notations and analysis

- Let us first focus on a simple example to understand the notations and conventions

Familiarity with notations and analysis

- Let us first focus on a simple example to understand the notations and conventions
- Assumptions

Familiarity with notations and analysis

- Let us first focus on a simple example to understand the notations and conventions
- Assumptions
  - The algorithms we wish to compare are correct. . .

Familiarity with notations and analysis

- Let us first focus on a simple example to understand the notations and conventions
- Assumptions
  - The algorithms we wish to compare are correct. . .
  - The algorithm that provably takes lesser time is a better one.

Familiarity with notations and analysis

- Let us first focus on a simple example to understand the notations and conventions
- Assumptions
  - The algorithms we wish to compare are correct. . .
  - The algorithm that provably takes lesser time is a better one.
  - Time ???? !!!!

Familiarity with notations and analysis

- Let us first focus on a simple example to understand the notations and conventions
- Assumptions
  - The algorithms we wish to compare are correct. . .
  - The algorithm that provably takes lesser time is a better one.
  - Time ???? !!!!
- For the time being, let us not bother about the exact time taken on a machine. . . .. the computational/storage power on different machines vary

Familiarity with notations and analysis

- Let us first focus on a simple example to understand the notations and conventions
- Assumptions
  - The algorithms we wish to compare are correct...
  - The algorithm that provably takes lesser time is a better one.
  - Time ???? !!!!
- For the time being, let us not bother about the exact time taken on a machine..... the computational/storage power on different machines vary
- But, let us assume that

Familiarity with notations and analysis

- Let us first focus on a simple example to understand the notations and conventions
- Assumptions
    - The algorithms we wish to compare are correct. . .
    - The algorithm that provably takes lesser time is a better one.
    - Time ???? !!!!
- For the time being, let us not bother about the exact time taken on a machine. . . .. the computational/storage power on different machines vary
- But, let us assume that
    - the cost of execution of a statement - in terms of time units - is some absract cost c_i.

Familiarity with notations and analysis

- Let us first focus on a simple example to understand the notations and conventions
- Assumptions
  - The algorithms we wish to compare are correct. . .
  - The algorithm that provably takes lesser time is a better one.
  - Time ???? !!!!
- For the time being, let us not bother about the exact time taken on a machine. . . .. the computational/storage power on different machines vary
- But, let us assume that
  - the cost of execution of a statement - in terms of time units - is some absract cost c_i.
  - An algorithm to find the maximum element of an n-element array.

FM: Familiarity with notations and analysis ...

- An algorithm to find the maximum element of an n-element array.

FM: Familiarity with notations and analysis ...

- An algorithm to find the maximum element of an n-element array.
    - Method1: Algorithm $Largest1(x_i, n)$ takes total steps = total time = $C_1 + nC_2 + (n-1)C_3 + (n-1)C_4 + C_5$

FM: Familiarity with notations and analysis ...

- An algorithm to find the maximum element of an n-element array.
  - Method1: Algorithm $Largest1(x_i, n)$ takes total steps = total time = $C_1 + nC_2 + (n-1)C_3 + (n-1)C_4 + C_5$
  - Method2: Algorithm $Largest2(x_i, n)$ takes total steps = total time = $nC_1 + (n-1)C_2 + (n-1)C_3 + C_4$

FM: Familiarity with notations and analysis ...

- An algorithm to find the maximum element of an n-element array.
  - Method1: Algorithm $Largest1(x_i, n)$ takes total steps = total time = $C_1 + nC_2 + (n-1)C_3 + (n-1)C_4 + C_5$
  - Method2: Algorithm $Largest2(x_i, n)$ takes total steps = total time = $nC_1 + (n-1)C_2 + (n-1)C_3 + C_4$
- Which algorithm is better for the same size of input n ?

Mathematical Analysis...

- Major Assumptions
  - the abstract operations are machine independent.
  - a constant amount of time is required to execute each line of pseudocode
- How to count the program steps?
  - comments, declarations
  - assignment statement
  - iterative statement
- How to instantiate the values of $c_i^{'}s$?

Tutorial Problems 2 to 6

- Devise the algorithm and perform the analysis as illustrated in the previous example
  - To find the sum of n elements in an integer array without using recursion.
  - To perform the bubble sort.
  - To find the smallest element from an integer array.
  - To find the factorial of a given number without using recursion.
  - To find the n_{th} Fibonacci number without using recursion.

Tutorial Problem No 7: Analyzing Insertion Sort

Algorithm Insertion-Sort(A[], n)

```
for j = 2 to n
   do key = A[j]
   i = j -1
   while (i > 0) and (A[i] > key)
   do A[i+1] = A[i]
          i = i-1
   A[i+1]=key
```

j = 2	i = 1	key=12	13	12	10
j = 2	i = 1	key=12	13	13	10
j = 2	i = 0	key=12	12	13	10

Figure: Timing Analysis

- What is the rough estimate of the complexity of the insertion sort ?
- But, how to carry out its mathematical analysis ?

Tutorial Problem No 7: Analyzing Insertion Sort

Homework Assignment

- The Insertion sort code as shown before....and the dry run
- Do the dry run on the input array 10,11,12 and prepare a table as shown below for all iterations.....
- Do the dry run on the input array 12,11,10 and prepare a table as shown below for all iterations.....

Tutorial Problem No 7: Timing Analysis: Mathematical

- How to write the expression for the times the statments are executed assuming c_i is the cost of statment i

j = 2	i = 1	key=12	13	12	10
j = 2	i = 1	key=12	13	13	10
j = 2	i = 0	key=12	12	13	10

Figure: Dry run: A[i] = {10,11,12}

Tutorial Problem No 7: Timing Analysis: Mathematical

- How to write the expression for the times the statments are executed assuming c_i is the cost of statment i
- We need to analyze how many times the while loop is executed ?

j = 2	i = 1	key=12	13	12	10
j = 2	i = 1	key=12	13	13	10
j = 2	i = 0	key=12	12	13	10

Figure: Dry run: A[i] = {10,11,12}

Tutorial Problem No 7: Timing Analysis: Mathematical

- How to write the expression for the times the statments are executed assuming c_i is the cost of statment i
- We need to analyze how many times the while loop is executed ?
- Assume while loop test is executed t_j times for every value of j

$$c_1 n + c_2(n-1) + c_3(n-1) + c4\sum_{j=2}^{n} t_j + c5\sum_{j=2}^{n}(t_j - 1) + c6\sum_{j=2}^{n}(t_j - 1) + c_7(n-1) \quad (1)$$

j = 2	i = 1	key=12	13	12	10
j = 2	i = 1	key=12	13	13	10
j = 2	i = 0	key=12	12	13	10

Figure: Dry run: A[i] = {10,11,12}

Tutorial Problem No 7: Timing Analysis: Mathematical

- How to write the expression for the times the statments are executed assuming c_i is the cost of statment i
- We need to analyze how many times the while loop is executed ?
- Assume while loop test is executed t_j times for every value of j

$$c_1 n + c_2(n-1) + c_3(n-1) + c4\sum_{j=2}^{n} t_j + c5\sum_{j=2}^{n}(t_j-1) + c6\sum_{j=2}^{n}(t_j-1) + c_7(n-1) \quad (1)$$

- Worst case Time = ? Best case Time = ?

j = 2	i = 1	key=12	13	12	10
j = 2	i = 1	key=12	13	13	10
j = 2	i = 0	key=12	12	13	10

Figure: Dry run: A[i] = {10,11,12}

Tutorial Problem No 7: Analyzing Insertion Sort...

- Best case - Apply the fact that the while loop test is executed t_j times for every value of j. What is t_j in this case ?

j = 2	i = 1	key=12	10	12	13
while loop executed only once to test the condition........... i.e. t_j=1			A[i] !> key		

Tutorial Problem No 7: Analyzing Insertion Sort...

- Best case - Apply the fact that the while loop test is executed t_j times for every value of j. What is t_j in this case ?

j = 2	i = 1	key=12	10	12	13
while loop executed only once to test the condition……….. i.e. t_j=1			A[i] !> key		

Tutorial Problem No 7: Analyzing Insertion Sort...

- Best case - Apply the fact that the while loop test is executed t_j times for every value of j. What is t_j in this case ?

j = 2	i = 1	key=12	10	12	13
while loop executed only once to test the condition........... i.e. t_j=1			A[i] !> key		

j = 3	i = 2	key=13	10	12	13
while loop executed only once to test the condition........... i.e. t_j=1			A[i] !> key		

$$Time = c_1 n + c_2(n-1) + c_3(n-1) + c4\sum_{j=2}^{n} t_j + c5\sum_{j=2}^{n}(t_j - 1) + c6\sum_{j=2}^{n}(t_j - 1) + c_7(n-1) \qquad (2)$$

- Best case Time = ?

Tutorial Problem No 7: Analyzing Insertion Sort...

- Worst case - Apply the fact that the while loop test is executed t_j times for every value of j. What is t_j in this case ?

j = 2	i = 1	key=12	13	12	10
j = 2	i = 1	key=12	13	13	10
j = 2	i = 0	key=12	12	13	10

- j=2 and so two iterations.....

Tutorial Problem No 7: Analyzing Insertion Sort...

- Worst case - Apply the fact that the while loop test is executed t_j times for every value of j. What is t_j in this case ?

j = 2	i = 1	key=12	13	12	10
j = 2	i = 1	key=12	13	13	10
j = 2	i = 0	key=12	12	13	10

- j=2 and so two iterations.....

Tutorial Problem No 7: Analyzing Insertion Sort...

- Worst case - Apply the fact that the while loop test is executed t_j times for every value of j. What is t_j in this case ?

j = 2	i = 1	key=12	13	12	10
j = 2	i = 1	key=12	13	13	10
j = 2	i = 0	key=12	12	13	10

- j=2 and so two iterations.....

j = 3	i = 2	key=10	12	13	10
j = 3	i = 2	key=10	12	13	13
j = 3	i = 1	key=10	12	12	13
j = 3	i = 0	key=10	10	12	13

$$Time = c_1 n + c_2(n-1) + c_3(n-1) + c4 \sum_{j=2}^{n} t_j + c5 \sum_{j=2}^{n} t_j - 1 + c6 \sum_{j=2}^{n} t_j - 1 + c_7(n-1)j \tag{3}$$

Reviewing our analysis approach

- Which analysis technique we have been studying so far ?

Reviewing our analysis approach

- Which analysis technique we have been studying so far ?
- What do the c_i's represent ?

Reviewing our analysis approach

- Which analysis technique we have been studying so far ?
- What do the c_i's represent ?
- Our assumptions ?

Reviewing our analysis approach

- Which analysis technique we have been studying so far ?
- What do the c_i's represent ?
- Our assumptions ?
  - We assumed that the time taken by a statement to execute is some abstract units... c_i's

Reviewing our analysis approach

- Which analysis technique we have been studying so far ?
- What do the c_i's represent ?
- Our assumptions ?
  - We assumed that the time taken by a statement to execute is some abstract units... c_i's
  - We analyzed only the best-case or the worst-case performance.

Reviewing our analysis approach

- Which analysis technique we have been studying so far ?
- What do the c_i's represent ?
- Our assumptions ?
  - We assumed that the time taken by a statement to execute is some abstract units... c_i's
  - We analyzed only the best-case or the worst-case performance.
  - Should we compute the exact time taken by a statement to execute ? Would it make sense ?

Reviewing our analysis approach

- Which analysis technique we have been studying so far ?
- What do the c_i's represent ?
- Our assumptions ?
  - We assumed that the time taken by a statement to execute is some abstract units... c_i's
  - We analyzed only the best-case or the worst-case performance.
  - Should we compute the exact time taken by a statement to execute ? Would it make sense ?
- How to find the exact values of c_i's?

Empirical Analysis

- Do not make any assumptions. . . .

```
typedef struct timeval {long tv_sec;
                        long tv_usec;
} timeval;
```

Empirical Analysis

- Do not make any assumptions. . . .
- How to time a program ? Any Linux commands ?

```
typedef struct timeval {long tv_sec;
                         long tv_usec;
} timeval;
```

Empirical Analysis

- Do not make any assumptions. . . .
- How to time a program ? Any Linux commands ?
- What we need is a multi-resolution timer... How to design one ?

```
typedef struct timeval {long tv_sec;
                        long tv_usec;
} timeval;
```

Empirical Analysis

- Do not make any assumptions. . . .
- How to time a program ? Any Linux commands ?
- What we need is a multi-resolution timer... How to design one ?
- Time the program using *timeval()* structure and *gettimeofday()*, *settimeofday()* family of calls in linux – multiresolution timer

```
typedef struct timeval {long tv_sec;
                        long tv_usec;
} timeval;
```

Empirical Analysis

- Do not make any assumptions. . . .
- How to time a program ? Any Linux commands ?
- What we need is a multi-resolution timer... How to design one ?
- Time the program using *timeval()* structure and *gettimeofday()*, *settimeofday()* family of calls in linux – multiresolution timer
- Logic

```
typedef struct timeval {long tv_sec;
                        long tv_usec;
} timeval;
```

Tutorial Problem No 8, 9

1. Create a data set or find a dataset from the internet - consisting of at least a million interger values in a vector. Write the Insertion sort routine in C and time the function to sort using the approach just discussed. Now repeat the same on the sorted output. Note the difference in time - in sorting an unsorted interger vector and a sorted one.
2. Repeat the above exercise for the Bubble sort, the Merge sort and the Quick sort covered. Note the time diffences.

Empirical Analysis...: Performance is relative

The outputs of such timing program obviously depend

- on many local factors
  - Machine
  - Compiler, Operating System
  - Algorithm
  - Input Data
  - ... ...
- and on many NOT so obvious factors
  - Caching
  - Garbage collection routines
  - Just-in-time compilation
  - CPU sharing/not. . . .

Our observations and inferences

- It is often difficult to get precise measurements of c_i's

Our observations and inferences

- It is often difficult to get precise measurements of c_i's
- Therefore, our assumptions are justified. . . .. let the abstract operations remain machine independent.

Our observations and inferences

- It is often difficult to get precise measurements of c_i's
- Therefore, our assumptions are justified. . . .. let the abstract operations remain machine independent.
- let *a constant amount of time* be required to execute each line of pseudocode

Our observations and inferences

- It is often difficult to get precise measurements of c_i's
- Therefore, our assumptions are justified. . . .. let the abstract operations remain machine independent.
- let *a constant amount of time* be required to execute each line of pseudocode
- Hence, Empirical analysis may not be useful at least in comparing the algorithms. . . ..

Our observations and inferences

- It is often difficult to get precise measurements of c_i's
- Therefore, our assumptions are justified. . . .. let the abstract operations remain machine independent.
- let *a constant amount of time* be required to execute each line of pseudocode
- Hence, Empirical analysis may not be useful at least in comparing the algorithms. . . ..
- The last question that still remains is "How to estimate the values of c_i's?

Estimating the value of c_i's

Integer add	a + b	2.1 ns
Integer multiply	a * b	2.4 ns
Integer divide	a / b	5.4 ns
Floating point add	a + b	4.6 ns
Floating point multiply	a * b	4.2 ns
Floating point divide	a / b	13.5 ns
sine	Math.sin(theta)	91.3 ns
arctangent	Math.atan(theta)	129. Ns

*Source : Sedgewick *Running OS X on Macbook Pro.....

Estimating the value of c_i's ...

- Therefore, now what could be our estimation of a typical c_i value ?

Cost of c_i's

Therefore, now, we shall assume that the abstract costs c_1, c_2, c_3,... are all equal and unity

Estimating the value of c_i's ...

- Therefore, now what could be our estimation of a typical c_i value ?
- Say when the input size is very large - typically mlllion or ten million or so, does this value of c_i have any impact on the time taken ?

Cost of c_i's

Therefore, now, we shall assume that the abstract costs c_1, c_2, $c_3, \ldots$ are all equal and unity

Tutorial Problem No 10: Computing sum of vector recursively

- The recursive algorithm to sum

Tutorial Problem No 10: Computing sum of vector recursively

- The recursive algorithm to sum
- Writing a recursion relation

Tutorial Problem No 10: Computing sum of vector recursively

- The recursive algorithm to sum
- Writing a recursion relation
- Solving the recursion relation.

Tutorial Problem No 10: Computing sum of vector recursively

- The recursive algorithm to sum
- Writing a recursion relation
- Solving the recursion relation.
- Comparing the complexity with the iterative version.

Revisiting our results

A relook at the time complexity expressions we have obtained so far

- Finding the maximum of n elements. Time complexity $= c_1 + c_2$ n $+ c_3$ (n-1) $+ c_4$ (n-1) $+ c_5 = 3n$

Revisiting our results

A relook at the time complexity expressions we have obtained so far

- Finding the maximum of n elements. Time complexity = c_1 + c_2 n + c_3 (n-1) + c_4 (n-1) + c_5 = $3n$
- Finding the sum of n elements of an input vector. Time complexity = c_1 + $c_2(n+1)$ + $c_3 n$ + c_4 = $2n + 3$

Revisiting our results

A relook at the time complexity expressions we have obtained so far

- Finding the maximum of n elements. Time complexity = c_1 + c_2 n + c_3 (n-1) + c_4 (n-1) + c_5 = $3n$
- Finding the sum of n elements of an input vector. Time complexity = c_1 + $c_2(n+1)$ + $c_3 n$ + c_4 = $2n + 3$
- Insertion Sort - Best Case: $5n - 4$. Worst Case:$3n^2 + 7n - 8$

Revisiting our results

A relook at the time complexity expressions we have obtained so far

- Finding the maximum of n elements. Time complexity = c_1 + c_2 n + c_3 (n-1) + c_4 (n-1) + c_5 = $3n$
- Finding the sum of n elements of an input vector. Time complexity = c_1 + $c_2(n+1)$ + $c_3 n$ + c_4 = $2n + 3$
- Insertion Sort - Best Case: $5n - 4$. Worst Case:$3n^2 + 7n - 8$
- Bubble Sort - Best Case: $2n^2 - 2n + 1$. Worst case: $3n^2 - 4n + 2$ Which term dominates the overall result in the above expression, especially at large values of n ?

The Growth of Functions

n	2n	4.5n	$n^3/2$	$5n^2$
5	10	22		
10	20	45		
100	200	450		
1000	2000	4500		
10000	20000	45000		
100000	$2.0 * 10^5$	$4.5 * 10^5$		
1000000	$2.0 * 10^6$	$4.5 * 10^6$		

The Growth of Functions ...

n	2n	4.5n	$n^3/2$	$5n^2$
5	10	22	45	125
10	20	45	500	500
100	200	450	$5*10^5$	$5*10^4$
1000	2000	4500	$5*10^8$	$5*10^6$
10000	20000	45000	$5*10^{11}$	$5*10^8$
100000	200000	450000	$5*10^{14}$	$5*10^{10}$
1000000	2000000	4500000	$5*10^{17}$	$5*10^{12}$

The Growth of Functions...

n	$T(n) = 3n^2 + 7n - 8$	$T(n) = 3n^2$
10	362	300
100	30692	30000
1000	$3.006992 * 10^6$	$3.00 * 10^6$
100000	$3.0000699992 * 10^{10}$	$3.00 * 10^{10}$

The Growth of Functions...

The Time Complexity

Hence, we shall now also drop the all the terms except the highest degree of the polynomial, when analyzing the running time of the algorithm. . . . . . .

Reviewing our Assumptions

So, now we have assumed/abstracted at three different levels viz.

- level 1 – ignored the actual cost of execution of each statement.

Reviewing our Assumptions

So, now we have assumed/abstracted at three different levels viz.

- level 1 – ignored the actual cost of execution of each statement.
- level 2 – ignored even the abstract cost (Ci) of each statement

Reviewing our Assumptions

So, now we have assumed/abstracted at three different levels viz.

- level 1 – ignored the actual cost of execution of each statement.
- level 2 – ignored even the abstract cost (Ci) of each statement
- level 3 – ignore all the terms except for the one with the highest degree in the expression of time complexity

Reviewing our Assumptions

So, now we have assumed/abstracted at three different levels viz.

- level 1 – ignored the actual cost of execution of each statement.
- level 2 – ignored even the abstract cost (Ci) of each statement
- level 3 – ignore all the terms except for the one with the highest degree in the expression of time complexity

Reviewing our Assumptions

So, now we have assumed/abstracted at three different levels viz.

- level 1 – ignored the actual cost of execution of each statement.
- level 2 – ignored even the abstract cost (Ci) of each statement
- level 3 – ignore all the terms except for the one with the highest degree in the expression of time complexity

Asymptotic Analysis

Such analysis is based on the asymptotic growth rate, asymptotic order or order of functions and called asymptotic analysis

Reviewing : Methods of Analysis

There can be at least three different ways of analysing the algorithms

- Empirical analysis. . . . is it feasible ?

Reviewing : Methods of Analysis

There can be at least three different ways of analysing the algorithms

- Empirical analysis.... is it feasible ?
  - The resources on different machines globally vary.....

Reviewing : Methods of Analysis

There can be at least three different ways of analysing the algorithms

- Empirical analysis. . . . is it feasible ?
  - The resources on different machines globally vary. . . ..
  - Not possible to make a universally true judgment about the algorithm

Reviewing : Methods of Analysis

There can be at least three different ways of analysing the algorithms

- Empirical analysis. . . . is it feasible ?
  - The resources on different machines globally vary. . . ..
  - Not possible to make a universally true judgment about the algorithm
- Mathematical analysis. . . ..is it worth ?

Reviewing : Methods of Analysis

There can be at least three different ways of analysing the algorithms

- Empirical analysis. . . . is it feasible ?
  - The resources on different machines globally vary. . . ..
  - Not possible to make a universally true judgment about the algorithm
- Mathematical analysis. . . ..is it worth ?
  - Abstract costs are insignificant, anyway. . .

Reviewing : Methods of Analysis

There can be at least three different ways of analysing the algorithms

- Empirical analysis. . . . is it feasible ?
    - The resources on different machines globally vary. . . ..
    - Not possible to make a universally true judgment about the algorithm
- Mathematical analysis. . . ..is it worth ?
    - Abstract costs are insignificant, anyway. . .
    - At higher magnitude, only the first term in the polynomial matters. . . .

Reviewing : Methods of Analysis

There can be at least three different ways of analysing the algorithms

- Empirical analysis. . . . is it feasible ?
  - The resources on different machines globally vary. . . ..
  - Not possible to make a universally true judgment about the algorithm
- Mathematical analysis. . . ..is it worth ?
  - Abstract costs are insignificant, anyway. . .
  - At higher magnitude, only the first term in the polynomial matters. . . .
  - At lower magnitude, do we need to care about even comparing the algorithms ?!!!!

Reviewing : Methods of Analysis

There can be at least three different ways of analysing the algorithms

- Empirical analysis. . . . is it feasible ?
  - The resources on different machines globally vary. . . ..
  - Not possible to make a universally true judgment about the algorithm
- Mathematical analysis. . . ..is it worth ?
  - Abstract costs are insignificant, anyway. . .
  - At higher magnitude, only the first term in the polynomial matters. . . .
  - At lower magnitude, do we need to care about even comparing the algorithms ?!!!!

Reviewing : Methods of Analysis

There can be at least three different ways of analysing the algorithms

- Empirical analysis. . . . is it feasible ?
  - The resources on different machines globally vary. . . ..
  - Not possible to make a universally true judgment about the algorithm
- Mathematical analysis. . . ..is it worth ?
  - Abstract costs are insignificant, anyway. . .
  - At higher magnitude, only the first term in the polynomial matters. . . .
  - At lower magnitude, do we need to care about even comparing the algorithms ?!!!!

Way to go

Therefore, the asymptotic analysis is the best way to go. . . .

How to relate complexity to input size?

Tutorial Problem No 11

- Write an algorithm EXPONENT(a, n) to find an using an appropriate method. Analyze the asymptotic complexity of the algorithm and compare it with the conventional method to do so.

Tutorial Problem No 11 ...

Algorithm BINEXPONENT(x,m)

```
1. let ans = 1
2. divide 2 into m giving quotient q &
remainder r
3. if r = 1
4.        then ans = ans * x
5. if q = 0
6.        goto exit
7. let m = q
8. let x = x * x
9. goto step 2
10. exit
```

Analysis of BINEXPONENT

Analysis

- Which are the vital steps ?

Analysis of BINEXPONENT

Analysis

- Which are the vital steps ?
- Can we say in general, how many times the multiplication in step 4 is performed?

Analysis of BINEXPONENT

Analysis

- Which are the vital steps ?
- Can we say in general, how many times the multiplication in step 4 is performed?
- The complexity turns out to be $3 + 3lgm$

Analysis of BINEXPONENT

Analysis

- Which are the vital steps ?
- Can we say in general, how many times the multiplication in step 4 is performed?
- The complexity turns out to be $3 + 3lgm$
- Compare the above with traditional algorithm

Analysis of BINEXPONENT

Analysis

- Which are the vital steps ?
- Can we say in general, how many times the multiplication in step 4 is performed?
- The complexity turns out to be $3 + 3lgm$
- Compare the above with traditional algorithm
- Effect of doubling the input size

Heuristics based approach
OR
Algorithmic approach?

Approaches to solve a problem

We now try to explore the answers to the following questions, w r to, approaches to solved a problem :

- Intuitive and hence iterative

Approaches to solve a problem

We now try to explore the answers to the following questions, w r to, approaches to solved a problem :

- Intuitive and hence iterative
- Algorithmic

Approaches to solve a problem

We now try to explore the answers to the following questions, w r to, approaches to solved a problem :

- Intuitive and hence iterative
- Algorithmic
- Why does one want an algorithmic approach to solve a problem?

Approaches to solve a problem

We now try to explore the answers to the following questions, w r to, approaches to solved a problem :

- Intuitive and hence iterative
- Algorithmic
- Why does one want an algorithmic approach to solve a problem?
- Would an algorithmic approach always work ? That is, would it solve all the computational problems that one encounters ?

Approaches to solve a problem

We now try to explore the answers to the following questions, w r to, approaches to solved a problem :

- Intuitive and hence iterative
- Algorithmic
- Why does one want an algorithmic approach to solve a problem?
- Would an algorithmic approach always work ? That is, would it solve all the computational problems that one encounters ?
- How can one have "faith" in an algorithm? That is, how can one know whether a given algorithm is correct or not ?

Approaches to solve a problem

We now try to explore the answers to the following questions, w r to, approaches to solved a problem :

- Intuitive and hence iterative
- Algorithmic
- Why does one want an algorithmic approach to solve a problem?
- Would an algorithmic approach always work ? That is, would it solve all the computational problems that one encounters ?
- How can one have "faith" in an algorithm? That is, how can one know whether a given algorithm is correct or not ?
- Why did we note that *an algorithm must not allow any subjective decision to be made*?

Exceptions to Non-Subjective Decisions in Algorithms

- There are three exceptions that allow non-subjectivity in algorithms.

Exceptions to Non-Subjective Decisions in Algorithms

- There are three exceptions that allow non-subjectivity in algorithms.
- What are those algorithm categories ?

Exceptions to Non-Subjective Decisions in Algorithms

- There are three exceptions that allow non-subjectivity in algorithms.
- What are those algorithm categories ?
- Why do we tolerate and permit such exceptions ?

Exceptions to Non-Subjective Decisions in Algorithms

- There are three exceptions that allow non-subjectivity in algorithms.
- What are those algorithm categories ?
- Why do we tolerate and permit such exceptions ?
- What is the difference between a heuristic and an approximation algorithm?

A Problem: Robot Tour Optimization

- A problem that arises often in manufacturing, transportation, and testing applications.

A Problem: Robot Tour Optimization

- A problem that arises often in manufacturing, transportation, and testing applications.
- To program the robot arm, mounted with a soldering gun, to solder the contact points on a PCB, such that all the contact points are soldered and the time taken to do so is optimally minimal.

A Problem: Robot Tour Optimization

- A problem that arises often in manufacturing, transportation, and testing applications.
- To program the robot arm, mounted with a soldering gun, to solder the contact points on a PCB, such that all the contact points are soldered and the time taken to do so is optimally minimal.
- In algorithm, we must first construct an ordering of the contact points

A Problem: Robot Tour Optimization

- A problem that arises often in manufacturing, transportation, and testing applications.
- To program the robot arm, mounted with a soldering gun, to solder the contact points on a PCB, such that all the contact points are soldered and the time taken to do so is optimally minimal.
- In algorithm, we must first construct an ordering of the contact points
  - the robot visits (and solders) the first contact point, then the second point, third, and so forth until the job is done.

A Problem: Robot Tour Optimization

- A problem that arises often in manufacturing, transportation, and testing applications.
- To program the robot arm, mounted with a soldering gun, to solder the contact points on a PCB, such that all the contact points are soldered and the time taken to do so is optimally minimal.
- In algorithm, we must first construct an ordering of the contact points
  - the robot visits (and solders) the first contact point, then the second point, third, and so forth until the job is done.
- The robot arm then proceeds back to the first contact point to prepare for the next board, thus the tool-path is a closed tour, or cycle.

Robot Tour Optimization: Formal definition

- Problem: Robot Tour Optimization

Robot Tour Optimization: Formal definition

- Problem: Robot Tour Optimization
- Input: A set S of n points in the plane.

Robot Tour Optimization: Formal definition

- Problem: Robot Tour Optimization
- Input: A set S of n points in the plane.
- Output: What is the shortest cycle tour that visits each point in the set S?

Robot Tour Optimization: Formal definition

- Problem: Robot Tour Optimization
- Input: A set S of n points in the plane.
- Output: What is the shortest cycle tour that visits each point in the set S?
    - i.e. to output an ordering of the contact points that the robot visits (and solders) one after the other until the job is done.

Robot Tour Optimization: Formal definition

- Problem: Robot Tour Optimization
- Input: A set S of n points in the plane.
- Output: What is the shortest cycle tour that visits each point in the set S?
  - i.e. to output an ordering of the contact points that the robot visits (and solders) one after the other until the job is done.
  - when the job is done, it proceeds back to the first contact point to prepare for the next board

Robot Tour Optimization: Formal definition

- Problem: Robot Tour Optimization
- Input: A set S of n points in the plane.
- Output: What is the shortest cycle tour that visits each point in the set S?
    - i.e. to output an ordering of the contact points that the robot visits (and solders) one after the other until the job is done.
    - when the job is done, it proceeds back to the first contact point to prepare for the next board
    - this entire tour cycle must be the shortest.

Robot Tour Optimization: Formal definition

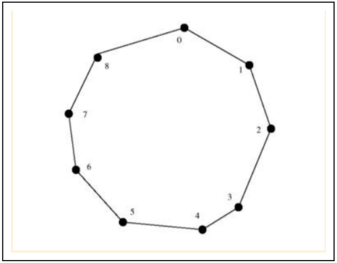
- Problem: Robot Tour Optimization
- Input: A set S of n points in the plane.
- Output: What is the shortest cycle tour that visits each point in the set S?
  - i.e. to output an ordering of the contact points that the robot visits (and solders) one after the other until the job is done.
  - when the job is done, it proceeds back to the first contact point to prepare for the next board
  - this entire tour cycle must be the shortest.
- Assumption: The robot arm moves with fixed speed, so the time to travel between two points is proportional to their distance.

Problem RTO: Heuristic#1: NearestNeighbor(P)

Nearest-neighbor heuristic approach: Starting from some point p_0, we walk first to its nearest neighbor p_1. Repeat the same from p_1.

Algorithm NearestNeighbor(P)

1. Pick and visit an initial point p_0 from P
2. p=p_0
3. i=0
4. while there are still unvisited points
5. i=i+1
6. select p_i to be the closest unvisited point to p_{i-1}
7. visit p_i
8. return to p_0 from p_{n-1}



Heuristic#1: NearestNeighbor(P) Counterexample

How is this figure a counterexample to the NearestNeighbor(P) approach.

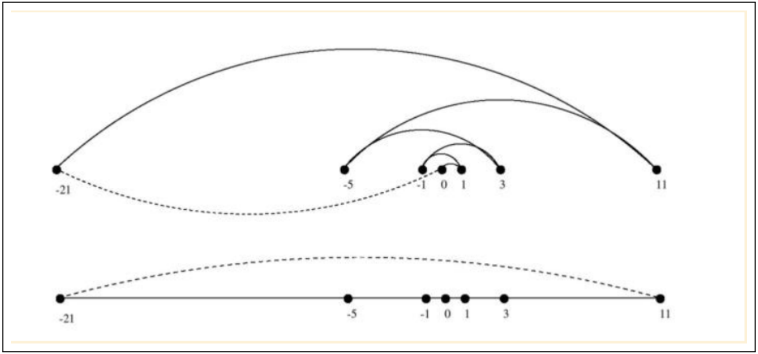


Figure: CounterExample: NearestNeighbor(P)

Problem RTO: Heuristic#2: ClosestPairofPoint(P)

Algorithm ClosestPair(P)

1. Let n be the number of points in set P
2. for i=1 to n-1 do
3. d $= \infty$
4. for each pair of endpoints(s,t) from distinct vertex chains
   if dist(s,t) $\leq$ d then s_m=s, t_m=t and d=dist(s,t)
5. connect (s_m, t_m) by an edge
6. connect the two endpoints by an edge

- each vertex begins as its own single vertex chain.
- after merging everything together, we will end up with a single chain containing all the points in it.
- connecting the final two endpoints gives us a cycle.
- at any step during the execution of this closest-pair heuristic, we will have a set of single vertices and vertex-disjoint chains available to merge.

Problem RTO: Heuristic#2: ClosestPairofPoint(P) Counterexample

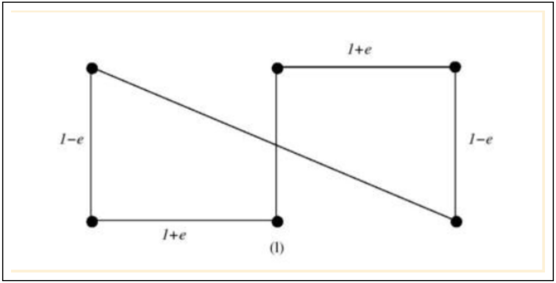


Figure: CounterExample: ClosestPairofPoint(P)

The total path length of the closest-pair tour in this case is $3(1-e)+2(1+e)+\sqrt{(1-e)^2+(2+2e)^2}$ which is over 20% farther than necessary when e is 0.

Problem RTO: What heuristic to use then?

- Then, what heuristic to use ?

Problem RTO: What heuristic to use then?

- Then, what heuristic to use ?
- Heuristics after all are heursitics, not algorithms !!!

Problem RTO: What heuristic to use then?

- Then, what heuristic to use ?
- Heuristics after all are heursitics, not algorithms !!!
- a correct algorithm could try enumerating all possible orderings of the set of points, and then select the ordering that minimizes the total length.

Problem RTO: What heuristic to use then?

- Then, what heuristic to use ?
- Heuristics after all are heursitics, not algorithms !!!
- a correct algorithm could try enumerating all possible orderings of the set of points, and then select the ordering that minimizes the total length.
- That is,

Problem RTO: What heuristic to use then?

- Then, what heuristic to use ?
- Heuristics after all are heursitics, not algorithms !!!
- a correct algorithm could try enumerating all possible orderings of the set of points, and then select the ordering that minimizes the total length.
- That is,

Problem RTO: What heuristic to use then?

- Then, what heuristic to use ?
- Heuristics after all are heursitics, not algorithms !!!
- a correct algorithm could try enumerating all possible orderings of the set of points, and then select the ordering that minimizes the total length.
- That is,

OptimalTSP(P)

1. d $= \infty$
2. for each of the $n!$ permutations P_i of the point set P
3. if $(cost(P_i) \leq d)$ then d$=$$(cost(P_i)$ and $P_min$$=$$P_i$
4. return P_min

Problem RTO: What heuristic to use then?...

- What could be the issues with the brute-force approach algorithm ?

Problem RTO: What heuristic to use then?...

- What could be the issues with the brute-force approach algorithm ?
  - The algorithm is correct but it is also extremely slow.

Problem RTO: What heuristic to use then?...

- What could be the issues with the brute-force approach algorithm ?
  - The algorithm is correct but it is also extremely slow.
  - The fastest computer in the world couldn't hope to enumerate all the 20! =2,432,902,008,176,640,000 orderings of 20 points within a day.

Problem RTO: What heuristic to use then?...

- What could be the issues with the brute-force approach algorithm ?
  - The algorithm is correct but it is also extremely slow.
  - The fastest computer in the world couldn't hope to enumerate all the 20! =2,432,902,008,176,640,000 orderings of 20 points within a day.
  - For real circuit boards, where n $\approx$ 1, 000 what could be the time required?

Problem RTO: What heuristic to use then?...

- What could be the issues with the brute-force approach algorithm ?
  - The algorithm is correct but it is also extremely slow.
  - The fastest computer in the world couldn't hope to enumerate all the 20! =2,432,902,008,176,640,000 orderings of 20 points within a day.
  - For real circuit boards, where n $\approx$ 1, 000 what could be the time required?
    - all of the world's computational power working full time wouldn't come close to finishing the problem before the end of the universe.

Heuristics and Algorithms

Key takeaway

- There is a fundamental difference between algorithms and heuristics.
- heuristics may usually do a good job but without providing any guarantee
- but algorithms on the other side, if found, always produce a correct result.

FM: Related areas of study

- Computability theory

FM: Related areas of study

- Computability theory
- Computational Complexity study

FM: Related areas of study

- Computability theory
- Computational Complexity study
- Undecidability

FM: Related areas of study

- Computability theory
- Computational Complexity study
- Undecidability
- An example of an undecidable problem.

Some interesting information

FM: Various Asymptotic Orders

lg n	$n^{1/2}$	n	n lg n	$n (\lg n)^2$	n^2
3	3	10	33	110	100
7	10	100	664	4414	10000
10	32	1000	9966	99317	10^6
13	100	10000	132877	1765633	10^8
17	316	100000	16660964	27588016	10^{10}
20	1000	1000000	19931569	397267426	10^{12}

FM: An interesting “seconds” conversion

10^2	1.7 min
10^4	2.8 hours
10^5	1.1 days
10^6	1.6 weeks
10^7	3.8 months
10^8	3.1 years
10^9	3.1 decades
10^{10}	3.1 centuries

FM: An interesting observation

	n	$n \lg n$	N^2	N^3	1.5^n	2^n	$n!$
n=10	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 4 sec
n=30	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	18 min	10^{25} yrs
n=50	< 1 sec	< 1 sec	< 1 sec	< 1 sec	11 min	36 yrs	very long
n=100	< 1 sec	< 1 sec	< 1 sec	1 sec	12.89 yrs	10^{17} yrs	< 4 sec
n=1000	< 1 sec	< 1 sec	1 sec	18 min sec	very long	very long	very long
n=10K	< 1 sec	< 1 sec	2 min	12 days	very long	very long	very long
n=100K	< 1 sec	2 sec	3 hrs	32 yrs	very long	very long	very long
n=1M	1 sec	20 sec	12 days	31.71 yrs	very long	very long	very long

FM: Basic Asymptotic Efficiency classes

1	constant
$\log n$	logarithmic
n	linear
$n \log n$	$n \log n$
n^2	quadratic
n^3	cubic
2^n	exponential
$n!$	factorial

Common Expressions & Complexity

Growth Rate	Typical Code Framework	Description	Example
1	a = b + c;	statement	add two statements
log n	while (n>1) { n = n/2; ……}	divide in half	binary search
n	for i= 1 to n { …….}	loop	find the max
n log n	……………	divide & conquer	mergesort
n^2	for i=1 to n { for j = 1 to n { … … …}}	double loop	check all pairs
n^3	for i=1 to n { for j = 1 to n { for k = 1 to n } … … …}}}	triple loop	check all triples
2^n	… … … … …	exhaustive search	check all possibilities

A few Tutorial Problems

FM: Tutorial Problem no 12 and 13

- Design the recursive version of the Fibonacci algorithm and only obtain the recurrence relation.

FM: Tutorial Problem no 12 and 13

- Design the recursive version of the Fibonacci algorithm and only obtain the recurrence relation.
- Design an algorithm for matrix addition and analyze its time complexity.

Tutorial Problem No 14

Find the cost of execution of the following code snippet

```
for i = 1 to n
for j = 1 to i
x = x + 1
```

Tutorial Problem No 15

Find the cost of execution of the following code snippet

```
j=n
while (j >= 1){
for i = 1 to j
x = x + 1
j = n/2
}
```

Tutorial Problems 16

Find the cost of execution of the following code snippet

```
for i = 1 to n
for j = 1 to i
for k = 1 to i
x = x + 1
```

FM: Tutorial Problem No 18

What is/could be the input size, in the following ?

- Find x in an array of names

FM: Tutorial Problem No 18

What is/could be the input size, in the following ?

- Find x in an array of names
- Multiply two matrices with real entities

FM: Tutorial Problem No 18

What is/could be the input size, in the following ?

- Find x in an array of names
- Multiply two matrices with real entities
- Sort an array of numbers

FM: Tutorial Problem No 18

What is/could be the input size, in the following ?

- Find x in an array of names
- Multiply two matrices with real entities
- Sort an array of numbers
- Traverse a binary tree

FM: Tutorial Problem No 18

What is/could be the input size, in the following ?

- Find x in an array of names
- Multiply two matrices with real entities
- Sort an array of numbers
- Traverse a binary tree
- Solve a problem concerning graphs

Blank

Blank

Blank

Blank

Blank

Blank