CS2002D: Program Design
Lecture-10

Analyzing the algorithms

- **What do we mean by “analyzing the algorithms”**
- Analyzing the algorithms means predicting the resources the algorithm uses
- What are the resources?
- **Computational time and Memory for storage**
- **Why do we analyze algorithms?**
- Analyzing several algorithms for a particular problem results in the most efficient algorithm in terms of computational time/memory

Analyzing the algorithms -contd.

- A bench mark / model of the implementation technology and the resources of that technology and their costs
- We assume a generic one processor **Random Access Machine** (RAM) model of computation
    - We use RAM as an implementation technology
    - Our algorithms will be implemented as computer programs
    - Instructions are executed one after another, with no concurrent operations

RAM model

- As it will be tedious to define each and every operation of RAM and its costs, we assume a realistic RAM
  - RAM contains instructions commonly found in real computers such as:
    - Arithmetic :eg: add, subtract, multiply, divide, remainder, floor, ceil
    - Data movement : eg: load, store, copy
    - Control : conditional & unconditional branch, subroutine call and return

RAM model - contd.

- **Data types** : integer and floating point
- Usually we do not concern ourselves on the precision of the value unless precision is very crucial
- We assume a limit on the size of each word of data
  - Eg: when working with inputs of size *n*, we assume that integers are represented by c lg *n* bits for some constant c >=1
    - We require c >=1, so that each word can hold the value of n
    - We restrict c to be a constant so that the word size does not grow arbitrarily

RAM model - contd.

- RAM model does not assume the details of implementation such as :
  - How is $\boldsymbol{x^y}$ **computed**? Is it a single step operation or multiple steps?
  - RAM model treats $\mathbf{2^k}$ **as constant time operation** when k is a small enough positive number
- RAM does not assume any memory models such as **virtual memory, cache**

Analysis of algorithm

- The **time taken** by an algorithm **grows with the input size**
- **Running time** of an algorithm is described as **a function of its input**
- We will formalize **“input size” and “running time**”
- **Input size**
  - Depends on the problem being studied

    Eg: for sorting problem, it is the number of elements

    Eg: for multiplying two numbers, it is the total number of bits

Running time of an algorithm

- Running time depends on what?
- Example pseudo code:

ADDITION

1. a = 321
2. b = 412
3. c = a + b
4. return c

- How do we find the running time of the above pseudocode?
- We will sum up the running times of lines 1,2,3 and 4
- Hence, we can conclude that the running time depends on the **number of operations or steps executed**

Running time of an algorithm

- Number of **primitive operations or steps** executed
- How do we decide what is one step? Is it machine dependent?
- We assume that a line in the pseudocode is one step
- Constant amount of time is needed to execute each line of a pseudocode
- Different lines may differ in the time taken for execution

Eg: while i > 0 and A[i] > key

A[i+1] = A[i]

- **Basic assumption:** i^{th} line takes time $c_{i,}$ where c_i is a constant

Before moving on to insertion sort analysis

ADDITION

1. a = 3102

2. b =4334

3. c=a+b

4. return c

- Step 1 takes c_1 time to execute or the cost incurred for executing step 1 is c_1
- Step 2 takes c_2 time to execute or the cost incurred for executing step 2 is c_2
- **Total running time T = $c_1 + c_2 + c_3 + c_4$**

ADDITION(n)

1. sum=0

2. for count = 1 to n

3. sum = sum + i

4. return sum

- The cost incurred for executing step 1 is c_1
- The cost incurred for executing step 2 is c_2 and it is executed n+1 times. Hence, the over all cost of step 2 is $(n+1)^* c_2$
- The cost incurred for executing step 3 is c_3 and it is executed n times. Hence, the over all cost of step 3 is $n^* c_3$
- The cost incurred for executing step 4 is c_4
- Total running time $T(n) = c_1 + (n+1)^* c_2 + n^* c_3 + c_4$

Cost and Times

- To analyse an algorithm, we will sum up the cost of each and every line of the pseudocode
- To compute the **cost** of one line, we will take the time taken to execute that line (cost incurred to execute that line) multiplied by how many **times** that line is executed
- Usually we write the running time as a function of n , represented as $T(n)$, where n is the input size of the problem

Pseudocode of Linear Search

LINEAR SEARCH(A, key)

```
found = 0
for i = 1 to A.length
        if A[ i ] = key
            found = 1
            return i
if found = 0
    return 0
```

Best Case of Linear Search

- **Best Case input of Linear Search** : The element to be searched is in the first position of the list
  - Eg: A= 1, 4, 2, 7, 10, 5 & key = 1
- How do we **analyse the linear search in the best case**?
- Step 1: Cost : c_1 , Times : 1
- Step 2: Cost : c_2 , Times : 1
- Step 3: Cost : c_3 , Times : 1
- Step 4: Cost : c_4 , Times : 1
- Step 5: Cost : c_5 , Times : 1
- $T(n) = c_1 + c_2 + c_3 + c_4 + c_5$

```
LINEAR SEARCH(A,key)
1. found = 0
2. for i = 1 to A.length
3.             if A[ i ] = key
4.                     found = 1
5.                     return i
6. if found = 0
7.     return 0
```

Worst Case of Linear Search

- **One of the Worst Case input of Linear Search :** The element to be searched is in the last position of the list
  - Eg: A = 1, 4, 2, 7, 10, 5 & key = 5
- How do we analyse the linear search in the worst case – successful search?
- Step 1: Cost : c_1 , Times : 1
- Step 2: Cost : c_2 , Times : n
- Step 3: Cost : c_3 , Times : n
- Step 4: Cost : c_4 , Times : 1
- Step 5: Cost : c_5 , Times : 1
- $T(n) = c_1 + n * c_2 + n* c_3+c_4 + c_5$

```
LINEAR SEARCH(A,key)
1. found = 0
2. for i = 1 to A.length
3.            if A[ i ] = key
4.                 found = 1
5.                 return i
6. if found = 0
7.     return 0
```

Analysis of Worst Case of Linear Search- Unsuccessful search

- **One of the Worst Case input of Linear Search** : The unsuccessful search analysis ie. the element is not present
  - Eg: A = 1, 4, 2, 7, 10, 5 & key = 0
- Step 1- Cost : c_1 , Times : 1
- Step 2- Cost : c_2 , Times : n+1
- Step 3- Cost : c_3 , Times : n
- Step 4- Cost :c_4 , Times : 0
- Step 5- Cost :c_5 , Times : 0
- Step 6- Cost : c_6 , Times : 1
- Step 7- Cost : c_7 , Times : 1
- $T(n) = c_1 + (n+1)^* c_2 + n^* c_3 + c_6 + c_7$

```
LINEAR SEARCH(A,key)
1. found = 0
2. for i = 1 to A.length
3.            if A[ i ] = key
4.                 found = 1
5.                 return i
6. if found = 0
7.    return 0
```

Insertion Sort

ANALYSIS

Analysis of algorithms

- We know : Analysing the algorithms means predicting the resources the algorithm uses
- We focus on the resource : **computational time**
- The time taken by the insertion sort depends on what?
- The time taken by the insertion sort depends on the input
- Does the time taken depend on anything else?
- Also depends on **how sorted is the input already**

Run-time Analysis

- Time taken by Insertion Sort (IS) algorithm depends on the input

  Sorting a million numbers takes longer than sorting ten numbers

- IS take different amounts of time to sort two input sequences of the same size

  Depends on how nearly sorted they already are

Run-time Analysis

- Time taken by the algorithm grows with the size of the input,

  Eg: n = 10, Running time = 1 unit

  n = 10000, Running time = 1000 units

- Running time as a function of the size of its input

Input Size

- **Sorting and Searching**

  Number of elements in the input

- **Finding the gcd of two numbers and Checking whether a number is a prime number or not**

  Total number of bits needed to represent the input in binary notation

- **Graph problems**

  Number of vertices and edges

Running time

- Number of primitive operations or steps executed
- Steps – machine independent
- Assumption (RAM model)

1. Constant amount of time is required to execute each line of pseudocode
2. Each execution of i^{th} line takes time c_i, where c_i is a constant

INSERTION-SORT(A)

for j = 2 to A.length	c_1
key = A[j];	c_2
// Insert A[j] into the sorted sequence A[1… j-1]	
i = j-1	c_3
while i > 0 and A[i] > key	c_4
do A[i+1] = A[i]	c_5
i = i-1	c_6
A[i+1] = key	c_7

While loop within for loop

- **For/while loop** : test is executed one time more than the loop body
- **Let t_j be the number of times** the while loop in line 5 is executed
- Since it is within a for loop, for each j = 2,3,…,n, where n = A.length, **total number of times while loop executed is $\Sigma_{j=2 \text{ to } n}\ t_j$**

INSERTION-SORT(A)	cost	Times
1. for j = 2 to A.length	c_1	n
2. key = A[j];	c_2	$n - 1$
3. // Insert A[j] into the sorted sequence A[1...j-1]		
4. i = j-1	c_3	$n - 1$
5. while i > 0 and A[i] > key	c_4	$\Sigma_{j=2 \text{ to } n}\ t_j$
6. A[i+1] = A[i]	c_5	$\Sigma_{j=2 \text{ to } n} (t_j-1)$
7. i = i -1	c_6	$\Sigma_{j=2 \text{ to } n} (t_j-1)$
8. A[i+1] = key	c_7	$n - 1$

Running time of an algorithm

- Sum of the running times for each statement executed
  - a statement that takes a cost of $\mathbf{c_i}$ to execute and is executed **n times**, **contribute $\mathbf{c_i}$ * n** to the total running time

T(n): running time of IS : sum of the products of the cost and times

T(n) = ?

What do you think is the best case for IS?

Input:
1,2,3,4,5,6,7,8,9,10

Input:
10,9,8,7,6,5,4,3,2,1

Best case of IS – Already sorted array

- For each j = 2,3,…,n, we know that **A[i] <= key** in line 5, i has its initial value of j – 1

  i.e A[1] <= 2, for j = 2, A[2] <= 3, for j = 3, …..

- Condition is FALSE and the body of the while loop will not be executed
- Condition alone will be executed, therefore,

  $t_j = 1$, for j = 2,3,…,n

- Best case running time:

$$T(n) = c_1 n + c_2 (n-1) + c_3 (n-1) + c_4 (n-1) + c_7 (n-1)$$

$$= (c_1 + c_2 + c_3 + c_4 + c_7)n - (c_2 + c_3 + c_4 + c_7)$$

Thank You