Algorithm Analysis: Part-2

Calculate Running Times of the following algorithms....

```
Count-Occurrence(A, key)
count = 0
for i = 1 to A.length
    if A[i] == key
        count = count + 1
return count
```

Running time $= c_1 + c_2(n + 1) + c_3 n + c_4 n + c_5$

- Running time on an input of n items
- $T(n)$, running time as a function of input size, n
- $T(n) = an + b$, a linear function of n

Search-Multiple(A, B)

```
count = 0
for  i  = 1 to B. length
        key  = B[i ]
        for  j  = 1 to A. length
                if A[i] == key
                        count  = count + 1
return count
```

Calculate Running Times of the following algorithms…

Search-Multiple(A, B)	cost	Times
1. count = 0	c1	1
2. for i = 1 to B. length	c2	m+1
3. key = B[i]	c3	n
4. for j = 1 to A. length	c4	m(n+1)
5. if A[i] == key	c5	mn
6. count = count + 1	c6	mn
7. return count	c7	1

<u>Note:</u> A.length = n, B.length = m

Running Time = ?

Cost and Times

- To analyse an algorithm, 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 (i.e the 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

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

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
```

What is the running time of Linear Search?

- Consider the different cases of the input
  - Best case
  - Worst case

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
```

Running time of Linear Search algorithm

- Considering the different cases of the input
  - Best case running time: $T(n) = c_1 + c_2 + c_3 + c_4 + c_5$
  - Worst case running time:
    - Successful Search: $T(n) = c_1 + n * c_2 + n * c_3 + c_4 + c_5$
    - Unsuccessful Search: $T(n) = c_1 + (n+1) * c_2 + n * c_3 + c_6 + c_7$

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?

  - **input size**
- Does the time taken depend on anything else?

  - how sorted is the input already

Run-time Analysis

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

  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

- **GCD of two numbers / 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