[CS203] Design and Analysis of Algorithms

Course Instructor: Dr. Dibyendu Roy
Scribed by: Prakhar jain (202251099)

Autumn 2023-24
Lecture (Week 1 and week 2)

Definition of Algorithm

An algorithm is a systematic set of finite instructions that transforms input into output. It operates in a step-by-step manner, and each step is well-defined and unambiguous.

Key Characteristics of Algorithms

Key characteristics of algorithms include:

1. Algorithms are implementable on a Random Access Machine (RAM). RAM can understand a finite set of instructions. If an instruction is not recognized, it can be broken down into a sequence of recognized instructions.
2. Every computer has a finite number of resources for executing algorithms. These resources include memory, processing power, and storage.
3. It is important to design algorithms that use minimal resources to accomplish their intended tasks, as efficient algorithms contribute to faster execution and reduced resource consumption.

Example: Sum of Two Numbers

A simple algorithm for finding the sum of two numbers is as follows:

```
1. Start
2. Input the first number (A)
3. Input the second number (B)
4. Calculate the sum: $C = A + B$
5. Output the sum ($C$)
6. End
```

Example: Calculating 2^n

Standard Method Algorithm

Here's an algorithm to calculate 2^n using a standard method:

1. Start
2. Input the value of n
3. Initialize result (res) to 1
4. Repeat n times:

   (a) Multiply res by 2 ($\text{res} = \text{res} \cdot 2$)

5. Output the result (res)
6. End

Bit Shifting Algorithm

You can also calculate 2^n using bit shifting:

1. Start
2. Input the value of n
3. Initialize result (res) to 1
4. Repeat n times:
   (a) Left shift res by 1 ($\text{res} = \text{res} \ll 1$)
5. Output the result (res)
6. End

Efficiency Comparison

The bit shifting algorithm is more efficient than the standard method for calculating 2^n. This is because bit shifting is a highly optimized operation at the hardware level, making it faster than multiplication. In the bit shifting algorithm, we are effectively multiplying by 2 in each iteration by shifting the bits to the left, which is much faster than repeatedly performing multiplication operations in the standard method. Therefore, for large values of n, the bit shifting algorithm performs significantly better in terms of execution time.

Insertion Sort Algorithm

Here's the algorithm for Insertion Sort:

```
1. Start
2. Input an array A of n elements
3. For i from 1 to n-1:
4.     key = A[i]
5.     j = i - 1
6.     while j >= 0 and A[j] > key:
7.         A[j + 1] = A[j]
8.         j = j - 1
9.     A[j + 1] = key
10. End
```

Here's a brief implementation of an insertion function:

```
\Procedure{Insertion}{$A, n, key$}
    \State $i \gets n - 1$
    \While{$i \geq 0$ and $A[i] > key$}
        \State $A[i + 1] \gets A[i]$
        \State $i \gets i - 1$
    \EndWhile
    \State $A[i + 1] \gets key$
\EndProcedure
```

This function inserts an element 'key' into the array 'A' of length 'n' in its correct position, assuming the array is already partially sorted.

Analysis

To analyze the Insertion Sort algorithm, we have considered both the best-case and worst-case execution counts.

Best-case Execution Count

In the best-case scenario, the input array is already sorted, so no swaps are needed inside the while loop. Here's the detailed analysis for the best-case execution count:

- Line 3 is executed $n - 1$ times.
- Line 4 is executed $n - 1$ times.
- Line 5 is executed 0 times (no swaps are needed).
- Line 6 is executed 0 times (no swaps are needed).
- Line 7 is executed $n - 1$ times.

Therefore, the best-case execution count in Insertion Sort is $2(n - 1)$.

Worst-case Execution Count

In the worst-case scenario, the input array is sorted in descending order, and each element needs to be moved to the beginning of the array. Here's the detailed analysis for the worst-case execution count:

- Line 3 is executed $n-1$ times.
- Line 4 is executed $n-1$ times.
- Line 5 is executed $(n-1)+(n-2)+\ldots+1$ times, which is the sum of the first $(n-1)$ natural numbers, making it $\frac{n(n-1)}{2}$ times.
- Line 6 is executed $(n-1)+(n-2)+\ldots+1$ times, which is the sum of the first $(n-1)$ natural numbers, making it $\frac{n(n-1)}{2}$ times.
- Line 7 is executed $n-1$ times.

Therefore, the worst-case execution count in Insertion Sort is $2(n-1)+\frac{n(n-1)}{2}+\frac{n(n-1)}{2} = 2n^2-2n$.

Merge Sort Algorithm

```
function mergeSort(arr)
    if length(arr) <= 1
        return arr
    mid = length(arr) / 2
    left = mergeSort(arr[1:mid])
    right = mergeSort(arr[mid+1:end])
    return merge(left, right)

function merge(left, right)
    result = []
    while left is not empty and right is not empty
        if left[1] <= right[1]
            append left[1] to result
            remove first element from left
        else
            append right[1] to result
            remove first element from right
    append remaining elements of left to result
    append remaining elements of right to result
    return result
```

Analysis of Merge Sort

Best Case Scenario

In the best-case scenario, the input array is already sorted, so the merge operation is minimal. Here's the step-by-step breakdown in tabular form:

Step	Description
1	Initial array: $[1, 2, 3, 4, 5]$
2	Split into two halves: $[1, 2]$ and $[3, 4, 5]$
3	Merge $[1, 2]$ and $[3, 4, 5]$: $[1, 2, 3, 4, 5]$

In the best-case scenario, Merge Sort has a time complexity of $O(n \log n)$.

Worst Case Scenario

In the worst-case scenario, the input array is in reverse order, leading to maximum merge operations. Here's the step-by-step breakdown in tabular form:

Step	Description
1	Initial array: $[5, 4, 3, 2, 1]$
2	Split into two halves: $[5, 4]$ and $[3, 2, 1]$
3	Split further: $[5]$ and $[4]$
4	Merge $[5]$ and $[4]$: $[4, 5]$
5	Split $[3, 2, 1]$ into $[3]$ and $[2, 1]$
6	Split further: $[2]$ and $[1]$
7	Merge $[2]$ and $[1]$: $[1, 2]$
8	Merge $[3]$ and $[1, 2]$: $[1, 2, 3]$
9	Merge $[4, 5]$ and $[1, 2, 3]$: $[1, 2, 3, 4, 5]$

In the worst-case scenario, Merge Sort also has a time complexity of $O(n \log n)$.