INTRODUCTION

- Session Learning Outcome - SLO – To Understand the concept of Insertion sort, the time complexity of an algorithm and Algorithm design paradigms.

ALGORITHM DESIGN AND ANALYSIS

Efficiency of the algorithm's design is totally dependent on the understanding of the problem.

The important parameters to be considered in understanding the problem are:

- Input
- Output
- Order of Instructions
- Check for repetition of Instructions
- Check for the decisions based on conditions

INSERTION SORT

- Insertion Sort Algorithm sorts array by shifting elements one by one and inserting the right element at the right position

- It works similar to the way you sort playing cards in your hands

PROCEDURE

- We start by making the second element of the given array, i.e. element at index 1, the key.
- We compare the key element with the element(s) before it, i.e., element at index 0:
  - If the key element is less than the first element, we insert the key element before the first element.
  - If the key element is greater than the first element, then we insert it after the first element.
- Then, we make the third element of the array as key and will compare it with elements to it's left and insert it at the right position.
- And we go on repeating this, until the array is sorted.

EXAMPLE

- Step 1:

7	**2**	**8**	**1**	**5**	**3**

- Step 2: (A[1] is compared with A[0], Since 2<7, its swapped)

2	**7**	**8**	**1**	**5**	**3**

- Step 3: (A[2] is compared with A[1] and A[0]. As 8 is the largest of all 3, no swapping is done)

2	**7**	**8**	**1**	**5**	**3**

EXAMPLE (Contd..)

- Step 4: (A[3] is compared with A[2], A[1] and A[0]. Since 1 is the smallest of all, its is swapped)

1	**2**	**7**	**8**	**5**	**3**

- Step 5: (A[4] is compared with A[3], A[2], A[1] and A[0]. Since 5 is lesser than 8 and 7 and greater than 2, it is swapped and placed between 2 and 7)

1	**2**	**5**	**7**	**8**	**3**

EXAMPLE (Contd..)

- Step 6: (A[5] is compared with A[4], A[3], A[2], A[1] and A[0]. Since 3 is lesser than 8,7,5 and greater than 1 &2, it is placed between 2 and 5).

1	**2**	**3**	**5**	**7**	**8**

- Array is sorted.

TIME COMPLEXITY

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 $A[1..j-1]$.	0	$n-1$
4 $i = j-1$	c_4	$n-1$
5 **while** $i > 0$ and $A[i] > key$	c_5	$\sum_{j=2}^{n} t_j$
6 $A[i+1] = A[i]$	c_6	$\sum_{j=2}^{n}(t_j-1)$
7 $i = i-1$	c_7	$\sum_{j=2}^{n}(t_j-1)$
8 $A[i+1] = key$	c_8	$n-1$

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5\sum_{j=2}^{n} t_j + c_6\sum_{j=2}^{n}(t_j-1) + c_7\sum_{j=2}^{n}(t_j-1) + c_8(n-1) .$$

$= O(n^2)$

TIME COMPLEXITY

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5 \sum_{j=2}^{n} j + c_6 \sum_{j=2}^{n} (j-1) + c_7 \sum_{j=2}^{n} (j-1) + c_8(n-1)$$

Since $\sum_{j=1}^{n} j = \frac{n(n+1)}{2}$ and $\sum_{j=2}^{n} j = \frac{n(n+1)}{2} - 1$ and $\sum_{j=2}^{n} (j-1) = \frac{n(n-1)}{2}$

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5\left(\frac{n(n+1)}{2} - 1\right) + c_6\left(\frac{n(n-1)}{2}\right)$$

$$+ c_7\left(\frac{n(n-1)}{2}\right) + c_8(n-1)$$

$$T(n) = k_1 n^2 + k_2 n + k_3 \quad \Rightarrow \quad T(n) = \Theta(n^2)$$

BEST CASE

- The term best-case performance is used describe an algorithm's behaviour under optimal conditions.
- For example, the best case for a simple linear search on a list occurs when the desired element is the first element of the list. i.e., O(1)
- Best case for Insertion sort is O(n)

WORST CASE

- The term worst-case performance is used describe the time taken by the algorithm to compute the result with maximum number of steps.
- Worst case of linear search algorithm on a list occurs when the desired element is in the last position of the array is O(n)
- Best case for Insertion sort is $O(n^2)$

AVERAGE CASE

- Average case is used to describe the time taken by the algorithm to compute the result with an average number of steps on input data.
- Average case for linear search algorithm is O(n)
- Average case for Insertion sort is $O(n^2)$

ALGORITHM DESIGN PARADIGMS

- Specifies the pattern to write or design an algorithm
- Various algorithm paradigms are
  - Divide and Conquer
  - Dynamic programming
  - Backtracking
  - Greedy Approach
  - Branch and Bound
- Selection of the paradigms depends upon the problem to be addressed

DIVIDE AND CONQUER

- The Divide and Conquer Paradigm is an algorithm design paradigm which uses this simple process: It Divides the problem into smaller sub-parts until these sub-parts become simple enough to be solved, and then the sub parts are solved recursively, and then the solutions to these sub-parts can be combined to give a solution to the original problem.
- Examples :
  - Binary search
  - Merge sort
  - Quick sort

DYNAMIC PROGRAMMING

- Dynamic Programming is an algorithmic paradigm that solves a given complex problem by breaking it into sub-problems and stores the results of sub-problems to avoid computing the same results again.
- It is an optimization technique
- Examples :
  - All pairs of shortest path
  - Fibonacci series

BACKTRACKING

- Backtracking is an algorithmic paradigm aimed at improving the time complexity of the exhaustive search technique if possible. Backtracking does not generate all possible solutions first and checks later. It tries to generate a solution and as soon as even one constraint fails, the solution is rejected and the next solution is tried.
- A backtracking algorithm tries to construct a solution incrementally, one small piece at a time. It's a systematic way of trying out different sequences of decisions until we find one that works.
- Examples :
  - 8 Queens problem
  - Sum of subsets

GREEDY APPROACH

- Greedy algorithms build a solution part by part, choosing the next part in such a way, that it gives an immediate benefit.
- This approach is mainly used to solve optimization problems.
- Finding the shortest path between two vertices using Dijkstra's algorithm.
- Examples
  - Prim's
  - Kruskal's algorithm
  - Travelling salesman problem
  - Graph - map coloring

BRANCH AND BOUND

- Branch and bound is an algorithm design paradigm which is generally used for solving combinatorial optimization problems.
- These problems are typically exponential in terms of time complexity and may require exploring all possible permutations in worst case.
- Examples :

  Greedy Algorithm for Fractional Knapsack

  DP solution for 0/1 Knapsack

  Backtracking Solution for 0/1 Knapsack.

HOME ASSIGNMENT

- Calculate the time complexity of binary search using Operation count

REFERENCES

1. *Thomas H Cormen, Charles E Leiserson, Ronald L Revest, Clifford Stein, Introduction to Algorithms, 3rd ed., The MIT Press Cambridge, 2014.*

2. *Ellis Horowitz, Sartajsahni, Sanguthevar, Rajesekaran, Fundamentals of Computer Algorithms, Galgotia Publication, 2010.*

3. *https://www.google.com/search?q=jeff+erickson+algorithms+pdf&oq=jeff&aqs=chrome.2.69i57j46i433j69i59j0i433j46i433l2j46i395i433j0i271.4680j1j7&sourceid=chrome&ie=UTF-8*