Time Complexity Analysis:

Best Case, Average Case

In best case, we get recurrence relation as:

$$\therefore T(n) = \begin{cases} 1 & \text{otherwise} \\ 2T(\frac{n}{2}) + n & n > 0 \end{cases}$$

$$\Rightarrow T(n) = 2T\left(\frac{n}{2}\right) + n \quad \text{--- (i)}$$

$$T\left(\frac{n}{2}\right) = 2T\left(\frac{n}{4}\right) + \frac{n}{2} \quad \text{--- (ii)}$$

$$T\left(\frac{n}{4}\right) = 2T\left(\frac{n}{8}\right) + \frac{n}{4} \quad \text{--- (iii)}$$

$$\therefore T(n) = 2\left[2T\left(\frac{n}{4}\right) + \frac{n}{2}\right] + n$$
$$= 4T\left(\frac{n}{4}\right) + 2n$$
$$= 2^2 T\left(\frac{n}{2^2}\right) + 2n$$

$$\therefore T(n) = 2^2 T\left[2T\left(\frac{n}{8}\right) + \frac{n}{4}\right] + 2n$$
$$= 2^3 T(n/2^3) + 3n$$
$$\vdots$$
$$= 2^4 T\left(\frac{n}{2^4}\right) + 4n$$
$$= 2^5 T\left(\frac{n}{2^5}\right) + 5n$$
$$\vdots$$

k times

$$= 2^k T\left(\frac{n}{2^k}\right) + kn$$
$$= 2^k T(1) + kn$$
$$= 2^k \cdot 1 + kn$$
$$= n + n\log n$$
$$\Longrightarrow O(n\log n)$$

$$\left[\ \because \frac{n}{2^k} = 1 \qquad n = 2^k \qquad \text{or, } \log n = \log 2^k \qquad \therefore \log n = k\ \right]$$

Overall, Quick Sort has an average-case time complexity of $O(n\log n)$, making it one of the fastest comparison sorting algorithms.

Experiment : 4

Date : 09/02/2024

Title : Quick Sort

Aim : To implement and analyze quick sort algorithm.

Algorithm :

Step 1 : Start

Step 2 : Read the size (n) and elements of the array from the user.

Step 3 : Apply the QuickSort algorithm to sort the array in ascending order.

Step 4 : Partition the array around a pivot element to separate smaller and larger elements.

Step 5 : Recursively apply QuickSort to the subarrays on the left and right of the pivot.

Step 6 : Print the sorted array.

Step 7 : Stop

Program Implementation

```c
#include <stdio.h>
#include <conio.h>
void printArray (int *A, int n){
    for (int i=0 ; i<n ; i++) {
        printf ("%d", A[i]);
    }
    printf ("\n");
}

int partition (int A[], int low, int high) {
    int pivot = A[low];
    int i = low+1;
    int j = high;
    int temp;
```

Time Complexity Analysis :

- **Worst Case**

In worst case, we get recurrence relation as :

$$\therefore T(n) = \begin{cases} 1 & \text{if } n = \\ T(n-1)+n & \text{otherwise} \end{cases}$$

$$\Rightarrow T(n) = T(n-1)+n$$

$$T(n-1) = T(n-1-1)+n-1$$
$$= T(n-2)+n-1$$

$$T(n-2) = T(n-2-1)+n-2$$
$$= T(n-3)+n-2$$

$$\therefore T(n) = T(n-2)+n-1+n$$

$$T(n) = T(n-3)+n-2+n-1+n$$

$\vdots$

$(n-1)$ times

$$T(n) = T(n-(n-1)) + \ldots + (n-2)+(n-1)+n$$
$$= T(1) \ldots + \ldots (n-2)+(n-1)+n$$
$$= 1+2+\ldots+(n-2)+(n-1)+n$$
$$= \frac{n(n+1)}{2}$$
$$= \frac{n^2+n}{2}$$
$$\Rightarrow O(n^2)$$

It can degrade to $O(n^2)$ in the worst case, particularly if the pivot selection is poorly optimized.

```c
do {
    while (A[i] <= pivot && i <= high) {
        i++;
    }
    while (A[j] > pivot && j >= low) {
        j--;
    }
    if (i < j) {   // Swap A[low] and A[j]
        temp = A[i];
        A[i] = A[j];
        A[j] = temp;
    }
} while (i < j);
temp = A[low];
A[low] = A[j];
A[j] = temp;
return j;
}

Void quickSort (int A[], int low, int high) {
    int partitionIndex; // Index of pivot after partition
    if (low < high) {
        partitionIndex = partition (A, low, high);
        quickSort (A, low, partitionIndex - 1); // sort left subarray
        quickSort (A, partitionIndex + 1, high); // sort right subarray
    }
}

int main () {
    int i, n;
    int A[100];
    clrscr(); // Clear section
    printf("Enter the size of the array : ");
    scanf ("%d", &n);
```

Dry run with sample input and output

Sample Input :

Enter the size of the array : 4

Enter 4 elements :

4

6

3

5

Sample Output :

Original Array : 4 6 3 5

Sorted Array : 3 4 5 6

```
printf ("Enter %d elements : \n",n);
for(i=0; i<n; i++){
    scanf("%d", &A[i]);
}
printf ("Original Array : ");
printArray (A,n);
quickSort (A,0,n-1);
printf ("Sorted Array : ");
printArray (A,n);
getch (); // Wait for a key press
return 0;
}
```

Result :

Quick Sort algorithm is implemented and analyzed successfully