Devadharshan.S

CH.SC.U4CSE24113

Week – 5

Date - 22/01/2026

Design and Analysis of Algorithm(23CSE211)

1. Quick Sort

Code:

```
#include <stdio.h>
#include <stdlib.h>
void swap(int* a, int* b) {
    int t = *a;
    *a = *b;
    *b = t;
}
int partition(int arr[], int low, int high, int pivotChoice) {
    int pivotIndex;
    if (pivotChoice == 1) {
        pivotIndex = low;
    } else if (pivotChoice == 2) {
        pivotIndex = high;
    } else {
        pivotIndex = low + rand() % (high - low + 1);
        printf("Random Pivot Element Chosen: %d\n", arr[pivotIndex]);
    }
    swap(&arr[pivotIndex], &arr[high]);
    int pivot = arr[high];
    int i = (low - 1);
    for (int j = low; j <= high - 1; j++) {
        if (arr[j] < pivot) {
            i++;
            swap(&arr[i], &arr[j]);
        }
    }
    swap(&arr[i + 1], &arr[high]);
    return (i + 1);
}
void quickSort(int arr[], int low, int high, int pivotChoice) {
    if (low < high) {
        int pi = partition(arr, low, high, pivotChoice);
        quickSort(arr, low, pi - 1, pivotChoice);
        quickSort(arr, pi + 1, high, pivotChoice);
    }
}
int main() {
    printf("CH.SC.U4CSE24113\n");
    int n, pivotChoice;
    printf("Enter number of elements: ");
    if (scanf("%d", &n) != 1) return 1;
    int *arr = (int *)malloc(n * sizeof(int));
    printf("Enter %d elements: ", n);
    for (int i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }
    printf("Choose Pivot Element:\n1. First Element\n2. Last Element\n3. Random 
Element\nChoice: ");
    scanf("%d", &pivotChoice);
    quickSort(arr, 0, n - 1, pivotChoice);
    printf("Sorted array: ");
```

```c
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
    free(arr);
    return 0;
}
```

Output:

```
devadharshan@devadharshan-HP-Pavilion-Laptop-15-eg3xxx: /media/dev...
devadharshan@devadharshan-HP-Pavilion-Laptop-15-eg3xxx:/media/devadharshan/New Volume/DAA/Lab/Week-5$ gcc QuickSort.c -o QuickSort
devadharshan@devadharshan-HP-Pavilion-Laptop-15-eg3xxx:/media/devadharshan/New Volume/DAA/Lab/Week-5$ ./QuickSort
CH.SC.U4CSE24113
Enter number of elements: 12
Enter 12 elements: 157 110 147 122 111 149 151 141 123 112 117 133
Choose Pivot Element:
1. First Element
2. Last Element
3. Random Element
Choice: 3
Random Pivot Element Chosen: 141
Random Pivot Element Chosen: 123
Random Pivot Element Chosen: 111
Random Pivot Element Chosen: 117
Random Pivot Element Chosen: 149
Random Pivot Element Chosen: 157
Sorted array: 110 111 112 117 122 123 133 141 147 149 151 157
devadharshan@devadharshan-HP-Pavilion-Laptop-15-eg3xxx:/media/devadharshan/New Volume/DAA/Lab/Week-5$
```

Time Complexity: O(n log n) (Average) / O(n^2) (Worst Case)
Justification: The algorithm uses a divide-and-conquer strategy. In the average case (especially with the Random Pivot choice), the array is split into two nearly equal halves, requiring log n levels of recursion, with each level performing a linear O(n) partition. However, if the pivot choice consistently results in highly unbalanced splits (e.g., choosing the first or last element of an already sorted array), the recursion depth can reach n, leading to a worst-case performance of O(n^2)

Space Complexity: O(log n) (Average) / O(n) (Worst Case)
Justification: The space complexity is determined by the maximum depth of the function call stack during recursion. Since an AVL tree is strictly balanced, its height h is guaranteed to be O(log n). Even though you start with an unbalanced tree, the recursive depth of balanceTree or printLevelOrder will not exceed the initial height of the tree. No significant auxiliary data structures (like arrays or queues) are used, only the memory for the nodes themselves and the recursion stack.