23CSE211- DESIGN AND ANALYSIS OF ALGORITHMS

AMRITA VISHWA VIDYAPEETHAM

CHENNAI CAMPUS

Name : B.SIDDHARTH REDDY

Roll No: CH.SC.U4CSE24104

Class : CSE-B

Year: 2025-2026

Program-1: QUICKSORT (pivot as last element)

Code:

```c
#include <stdio.h>

int partition(int a[], int low, int high) {
    int pivot = a[high];   // last element as pivot
    int i = low - 1;

    for (int j = low; j < high; j++) {
        if (a[j] <= pivot) {
            i++;
            int temp = a[i];
            a[i] = a[j];
            a[j] = temp;
        }
    }

    int temp = a[i + 1];
    a[i + 1] = a[high];
    a[high] = temp;

    return i + 1;
}

void quickSort(int a[], int low, int high) {
    if (low < high) {
        int pi = partition(a, low, high);
        quickSort(a, low, pi - 1);
        quickSort(a, pi + 1, high);
    }
}

int main() {
    int a[] = {157,110,147,122,111,149,151,141,123,112,117,133};
    int n = sizeof(a) / sizeof(a[0]);

    quickSort(a, 0, n - 1);

    printf("Sorted array (Pivot = Last):\n");
    for (int i = 0; i < n; i++)
        printf("%d ", a[i]);

    return 0;
}
```

Output:

```
Sorted array (Pivot = Last):
110 111 112 117 122 123 133 141 147 149 151 157
```

Time Complexity

Case	Complexity	Explanation
Best case	**O(n log n)**	Pivot splits array evenly
Average case	**O(n log n)**	Random data
Worst case	**O(n^2)**	Array already sorted or reverse sorted

Same issue as first pivot, but now with **last element**

Space Complexity

- **O(log n)** → Best & Average
- **O(n)** → Worst case

✔ In-place sorting
✔ Only recursion stack used.

Program-2: QUICKSORT (pivot as first element)

Code:

```c
#include <stdio.h>

int partition(int a[], int low, int high) {
    int pivot = a[low];   // first element as pivot
    int i = low + 1;
    int j = high;

    while (i <= j) {
        while (i <= high && a[i] <= pivot)
            i++;
        while (a[j] > pivot)
            j--;

        if (i < j) {
            int temp = a[i];
            a[i] = a[j];
            a[j] = temp;
        }
    }

    int temp = a[low];
    a[low] = a[j];
    a[j] = temp;

    return j;
}

void quickSort(int a[], int low, int high) {
    if (low < high) {
        int pi = partition(a, low, high);
        quickSort(a, low, pi - 1);
        quickSort(a, pi + 1, high);
    }
}
```

```c
int main() {
    int a[] = {157,110,147,122,111,149,151,141,123,112,117,133};
    int n = sizeof(a) / sizeof(a[0]);

    quickSort(a, 0, n - 1);

    printf("Sorted array (Pivot = First):\n");
    for (int i = 0; i < n; i++)
        printf("%d ", a[i]);

    return 0;
}
```

Output:

```
Sorted array (Pivot = First):
110 111 112 117 122 123 133 141 147 149 151 157
```

Time Complexity

Case	Complexity	Explanation
Best case	**O(n log n)**	Pivot divides array into two nearly equal halves
Average case	**O(n log n)**	Random distribution of elements
Worst case	**O(n^2)**	Array already sorted or reverse sorted

Worst case happens because **first element is always smallest/largest**

Space Complexity

- **O(log n)** → Best & Average (balanced recursion)
- **O(n)** → Worst case (deep recursion)

✔ No extra array used
✔ Only recursion stack

Program-3: QUICKSORT (pivot as random element)

Code:

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int partition(int a[], int low, int high) {
    int pivot = a[high];
    int i = low - 1;

    for (int j = low; j < high; j++) {
        if (a[j] <= pivot) {
            i++;
            int temp = a[i];
            a[i] = a[j];
            a[j] = temp;
        }
    }

    int temp = a[i + 1];
    a[i + 1] = a[high];
    a[high] = temp;

    return i + 1;
}

int randomPartition(int a[], int low, int high) {
    int randomIndex = low + rand() % (high - low + 1);

    int temp = a[randomIndex];
    a[randomIndex] = a[high];
    a[high] = temp;

    return partition(a, low, high);
}

void quickSort(int a[], int low, int high) {
    if (low < high) {
        int pi = randomPartition(a, low, high);
        quickSort(a, low, pi - 1);
        quickSort(a, pi + 1, high);
    }
}

int main() {
    srand(time(NULL));

    int a[] = {157,110,147,122,111,149,151,141,123,112,117,133};
    int n = sizeof(a) / sizeof(a[0]);

    quickSort(a, 0, n - 1);

    printf("Sorted array (Pivot = Random):\n");
    for (int i = 0; i < n; i++)
        printf("%d ", a[i]);

    return 0;
}
```

Output:

```
Sorted array (Pivot = Random):
110 111 112 117 122 123 133 141 147 149 151 157
```

Time Complexity

Case	Complexity	Explanation
Best case	**O(n log n)**	Random pivot gives balanced partitions
Average case	**O(n log n)**	Highly likely balanced
Worst case	**O(n^2)**	Very rare, unlucky random choices

Worst case is **theoretically possible** but **practically rare**

Space Complexity

- **O(log n)** → Expected
- **O(n)** → Worst case (rare)

✔ Extra work only to pick pivot
✔ Still in-place

.