WEEK - 5

Kavin.J.S
CH.SC.U4CSE24119

QuickSort : Random Element as Piviot:

Code:

quicksort_randomelementaspiviot.c

C: > Users > kavin > OneDrive > Desktop > DAA > WEEK - 5 > C quicksort_randomelementaspi

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

int partitionLast(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 partitionRandom(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 partitionLast(A, low, high);
}

void quickSortRandom(int A[], int low, int high)
{
    if (low < high)
    {
        int pi = partitionRandom(A, low, high);

        quickSortRandom(A, low, pi - 1);
        quickSortRandom(A, pi + 1, high);
    }
}

int main()
{
    srand(time(0));

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

    printf("Original Array:\n");
    for (int i = 0; i < n; i++)
        printf("%d ", arr[i]);

    quickSortRandom(arr, 0, n - 1);

    printf("\n\nSorted Array (Random Pivot):\n");
    for (int i = 0; i < n; i++)
        printf("%d ", arr[i]);

    return 0;
}
```

Restricted Mode ⊗0 ⚠0

Output:

```
C:\Users\kavin\OneDrive\Desktop\DAA\WEEK - 5>gcc quicksort_randomelementaspiviot.c

C:\Users\kavin\OneDrive\Desktop\DAA\WEEK - 5>a
Original Array:
157 110 147 122 111 149 151 141 123 112 117 133

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

QuickSort : First Element as Piviot:

Code:

C quicksort_firstelementaspiviot.c | C quicksort_lastelementaspiviot.c | C quicksort_random

C: > Users > kavin > OneDrive > Desktop > DAA > WEEK - 5 > C quicksort_firstelementaspiviot.c

```c
#include <stdio.h>

int partitionFirst(int A[], int low, int high)
{
    int pivot = A[low];
    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 quickSortFirst(int A[], int low, int high)
{
    if (low < high)
    {
        int pi = partitionFirst(A, low, high);
        quickSortFirst(A, low, pi - 1);
        quickSortFirst(A, pi + 1, high);
    }
}

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

    printf("Original Array:\n");
    for (int i = 0; i < n; i++)
        printf("%d ", arr[i]);

    quickSortFirst(arr, 0, n - 1);

    printf("\n\nSorted Array (First Pivot):\n");
    for (int i = 0; i < n; i++)
        printf("%d ", arr[i]);

    return 0;
}
```

Output:

```
C:\Users\kavin\OneDrive\Desktop\DAA\WEEK - 5>gcc quicksort_firstelementaspiviot.c

C:\Users\kavin\OneDrive\Desktop\DAA\WEEK - 5>a
Original Array:
157 110 147 122 111 149 151 141 123 112 117 133

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

QuickSort : Last Element as Piviot:

Code:

quicksort_lastelementaspiviot.c | quicksort_randomelementaspiviot.c

C: > Users > kavin > OneDrive > Desktop > DAA > WEEK - 5 > quicksort_lastelementaspiviot.c

```c
#include <stdio.h>

int partitionLast(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;
}

void quickSortLast(int A[], int low, int high)
{
    if (low < high)
    {
        int pi = partitionLast(A, low, high);
        quickSortLast(A, low, pi - 1);
        quickSortLast(A, pi + 1, high);
    }
}

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

    printf("Original Array:\n");
    for (int i = 0; i < n; i++)
        printf("%d ", arr[i]);

    quickSortLast(arr, 0, n - 1);

    printf("\n\nSorted Array (Last Pivot):\n");
    for (int i = 0; i < n; i++)
        printf("%d ", arr[i]);

    return 0;
}
```

Output:

```
C:\Users\kavin\OneDrive\Desktop\DAA\WEEK - 5>gcc quicksort_lastelementaspiviot.c

C:\Users\kavin\OneDrive\Desktop\DAA\WEEK - 5>a
Original Array:
157 110 147 122 111 149 151 141 123 112 117 133

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

Analysis:

The Random Pivot Quick Sort is better.

Random pivot is better because it reduces the chance of getting the worst-case time complexity.

If we always choose the first or last element as pivot, then for already sorted or reverse sorted arrays, Quick Sort becomes very slow $O(n^2)$.

But if we choose the pivot randomly, the partitions are more balanced in most cases.

Balanced partition means fewer comparisons and faster sorting.

So, Random Pivot Quick Sort gives better performance on average and avoids predictable worst cases.