

WEEK - 5

Kavin.J.S
CH.SC.U4CSE24119

QuickSort : First Element as Pivot:
Code:

The screenshot shows a code editor window with a dark theme. The file is named "quicksort_firstelementaspivot.c". The code implements the QuickSort algorithm with the first element as the pivot. It includes a partition function that rearranges elements around a pivot, and a quickSortFirst function that recursively sorts the array. The main function reads input from the user and prints the sorted array.

```
#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 n;
    printf("Enter number of elements: ");
    scanf("%d", &n);

    int A[n];
    printf("Enter elements:\n");
    for (int i = 0; i < n; i++)
        scanf("%d", &A[i]);

    quickSortFirst(A, 0, n - 1);

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

    return 0;
}
```

Output:

```
C:\Users\kavin\OneDrive\Desktop\DAA\WEEK - 5>gcc quicksort_firstelementaspivot.c
C:\Users\kavin\OneDrive\Desktop\DAA\WEEK - 5>a
Enter number of elements: 5
Enter elements:
7 1 2 8 5
Sorted array (First Pivot):
1 2 5 7 8
```

QuickSort : Last Element as Pivot:

Code:

The screenshot shows a code editor window with the title "quicksort_lastelementaspivot.c". The menu bar includes "File", "Edit", and "View". The code itself is a C program that implements the QuickSort algorithm. It starts with an include directive for stdio.h. The partitionLast function takes an array A, and indices low and high. It initializes pivot to A[high], i to low - 1, and then iterates from j = low to high - 1. If A[j] <= pivot, it swaps A[i] and A[j], increments i, and swaps A[i] and A[j]. After the loop, it swaps A[i + 1] and A[high], and A[high] and A[i + 1]. It then returns i + 1. The quickSortLast function is a recursive call to partitionLast, starting at index 0 and ending at index high. The main function prompts the user for the number of elements n, creates an array A of size n, and then calls quickSortLast with indices 0 and n - 1. Finally, it prints the sorted array.

```
#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 n;
    printf("Enter number of elements: ");
    scanf("%d", &n);

    int A[n];
    printf("Enter elements:\n");
    for (int i = 0; i < n; i++)
        scanf("%d", &A[i]);

    quickSortLast(A, 0, n - 1);

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

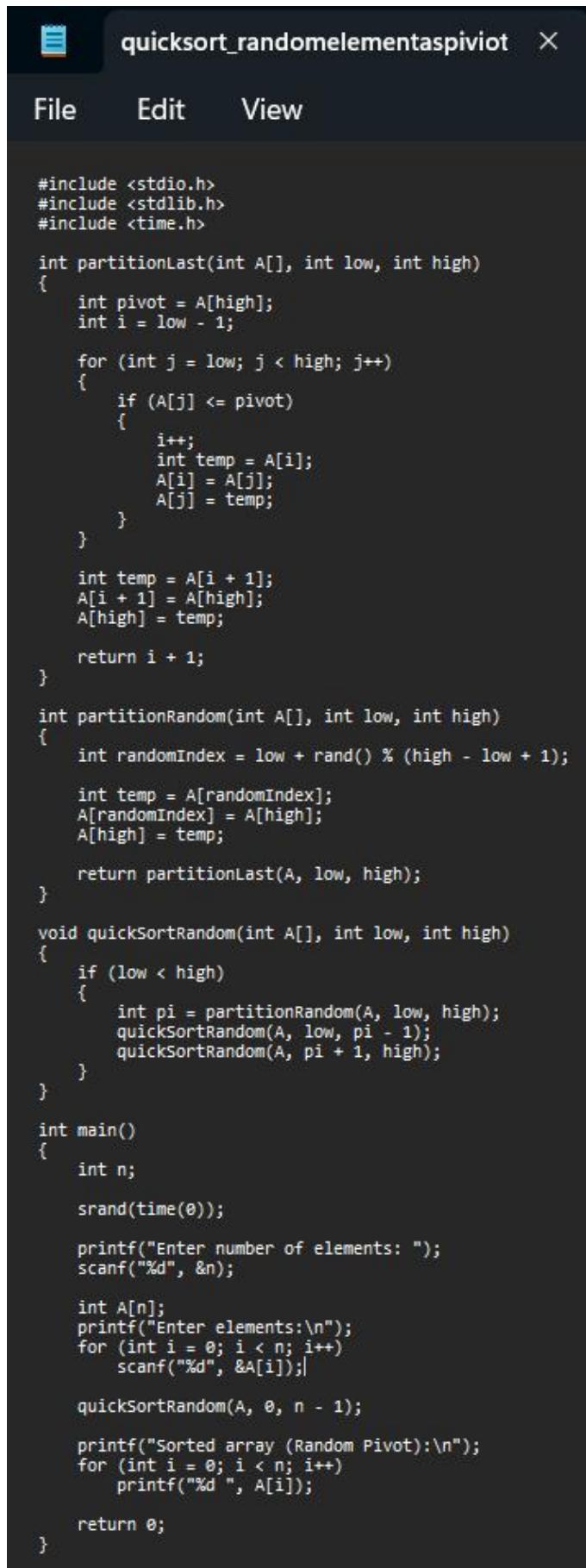
    return 0;
}
```

Output:

```
C:\Users\kavin\OneDrive\Desktop\DAAlWEEK - 5>gcc quicksort_lastelementaspivot.c
C:\Users\kavin\OneDrive\Desktop\DAAlWEEK - 5>a
Enter number of elements: 5
Enter elements:
7 1 2 8 5
Sorted array (Last Pivot):
1 2 5 7 8
```

QuickSort : Random Element as Pivot:

Code:



The screenshot shows a code editor window with the title "quicksort_randomelementaspivot". The menu bar includes "File", "Edit", and "View". The code itself is a C program that implements the Quicksort algorithm with a random pivot element.

```
#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()
{
    int n;

    srand(time(0));

    printf("Enter number of elements: ");
    scanf("%d", &n);

    int A[n];
    printf("Enter elements:\n");
    for (int i = 0; i < n; i++)
        scanf("%d", &A[i]);

    quickSortRandom(A, 0, n - 1);

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

    return 0;
}
```

Output:

```
C:\Users\kavin\OneDrive\Desktop\DA\WEEK - 5>gcc quicksort_randomelementaspivot.c  
C:\Users\kavin\OneDrive\Desktop\DA\WEEK - 5>a  
Enter number of elements: 5  
Enter elements:  
7 1 2 8 5  
Sorted array (Random Pivot):  
1 2 5 7 8
```

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.