DESIGN AND ANALYSIS OF ALGORITHMS

LAB WORKBOOK WEEK-6

NAME : M. SRIJA

ROLL.NO: CH.SC.U4CSE24126

DEPARTMENT: CSE-B

Quick sort taking First,Last,Random element as pivot elements

```c
// SRIJA
// CH.SC.U4CSE24126

#include <stdio.h>
#include <stdlib.h>

void swap(int *a, int *b){
    int temp = *a;
    *a = *b;
    *b = temp;
}

int partitionFirst(int a[], int low, int high){
    int pivot = a[low];
    int i = low + 1, j = high;

    while(i <= j){
        while(i <= high && a[i] <= pivot){
            i++;
        }
        while(a[j] > pivot){
            j--;
        }
        if(i < j){
            swap(&a[i], &a[j]);
        }
```

```
    }
    swap(&a[low], &a[j]);
    return j;
}

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++;
            swap(&a[i], &a[j]);
        }
    }
    swap(&a[i + 1], &a[high]);
    return i + 1;
}

int partitionRandom(int a[], int low, int high){
    int randomIndex = low + rand() % (high - low + 1);
    printf("Randomly selected pivot: %d\n", a[randomIndex]);
    swap(&a[randomIndex], &a[high]);
    return partitionLast(a, low, high);
```

```
}

void quickSort(int a[], int low, int high, int choice){
    if(low < high){
        int p;
        if(choice == 1){
            p = partitionFirst(a, low, high);
        }
        else if(choice == 2){
            p = partitionLast(a, low, high);
        }
        else{
            p = partitionRandom(a, low, high);
        }
        quickSort(a, low, p - 1, choice);
        quickSort(a, p + 1, high, choice);
    }
}

void copyArray(int src[], int dest[], int n){
    for(int i = 0; i < n; i++){
        dest[i] = src[i];
    }
}
```

```c
void display(int a[], int n){
    for(int i = 0; i < n; i++){
        printf("%d ", a[i]);
    }
    printf("\n");
}

int main(){
    int n, choice;

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

    int original[n], temp[n];

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

    while(1){
        printf("\n----- QUICK SORT MENU -----\n");
        printf("1. First Element as Pivot\n");
        printf("2. Last Element as Pivot\n");
        printf("3. Random Element as Pivot\n");
        printf("4. Exit\n");
```

```c
        printf("Enter your choice: ");
        scanf("%d", &choice);

        if(choice == 4){
            printf("Exiting program...\n");
            break;
        }

        if(choice < 1 || choice > 3){
            printf("Invalid choice! Try again.\n");
            continue;
        }

        copyArray(original, temp, n);

        printf("\nOriginal array:\n");
        display(temp, n);

        quickSort(temp, 0, n - 1, choice);

        printf("Sorted array:\n");
        display(temp, n);
    }

    return 0;
}
```

OUTPUT:

```
Choose Pivot Type (Method):
1. First element
2. Last element
3. Random element
Enter choice: 1
Sorted array:
110 111 112 117 122 123 133 141 147 149 151 157
Choose Pivot Type (Method):
1. First element
2. Last element
3. Random element
Enter choice: 2
Sorted array:
110 111 112 117 122 123 133 141 147 149 151 157
Choose Pivot Type (Method):
1. First element
2. Last element
3. Random element
Enter choice: 3
Sorted array:
110 111 112 117 122 123 133 141 147 149 151 157
```

TIME COMPLEXITY:- O(n log n)

JUSTIFICATION:-

Quick Sort divides the array into two parts at each partition step. On average, the pivot divides the array into nearly equal halves. Each partition operation takes O(n) time to compare elements. The recursion depth is log n. Therefore, total time complexity = n × log n = O(n log n).

SPACE COMPLEXITY :- O(log n)

JUSTIFICATION :-

The given Quick Sort program uses recursion, hence extra space is required only for the

recursion stack.
Each recursive call stores the following data:

int low → 4 bytes

int high → 4 bytes

int pivot → 4 bytes

return address and control information → 8 bytes

Total memory required per recursive call = 20 bytes

In the average case, the maximum recursion depth is log n.
Therefore, the total stack memory required is:

$$\mathbf{20 \times \log\ n} \text{ bytes}$$

Hence, the space complexity of the given code is O(log n).

Random pivot is better because it avoids unbalanced partitions, reduces the chance of worst-case time complexity, and gives consistent O(n log n)

performance for most inputs.

Quick sort:-

157, 110, 147, 122, 111, 149, 151, 141, 123, 112, 117, 133

Pivot element = first element

i = left +1

j = right

Move i right side till A[i] > pivot

Move j left side till A[j] < pivot

If i<j → swap A[i], A[j]

If i≥j → swap pivot with A[j]

Index	0	1	2	3	4	5	6	7	8	9	10	11
Array	157	110	147	122	111	149	151	141	123	112	117	133

Step 1:-

Pivot : 157

i = 1   j = 11

No element is greater than 157

j stays at 11

157 > 110 [move i right side]

157 > 147

157 > 122

157 > 133

i moves till end

i = j

so swap 157 ⟷ 133

133	110	147	122	111	149	151	141	123	112	117	157
0	1	2	3	4	5	6	7	8	9	10	11

pivot

step 2:- pivot = 133

i = 1   j = 10

133 > 110 (i mover right)

133 < 147 (i stays at 147)

swap (147, 117)

133	110	117	122	111	149	151	141	123	112	147

133 > 122 (moves i right) 112 < 133 (stops at j=9)

133 > 111 (moves i right) i < j

133 < 149 (stops at i=5) Swap (149, 112)

133	110	117	122	111	112	151	141	123	149	147
					i				j	

112 < 133 (moves i right) 149 > 133 (moves j left)

151 > 133 (stops at i = 6) 123 < 133 (stops at j = 8)

i < j

swap (151, 123)

133	110	117	122	111	112	123	141	151	149	147
						i		j		

123 < 133 (move i right) 151 > 133 (moves j left)

141 > 133 (stops at i = 7) 141 > 133 (moves j left)

i = 7 j = 6 123 < 133 (stops at j = 6)

i > j so swap (pivot, A[j])

(133, 123)

123	110	117	122	111	112	133	141	151	149	147
0	1	2	3	4	5	6	7	8	9	10

Step 3:-

pivot = 123

partition (0-5)

i = 1 j = 5

110 < 123 (i moves right)

117 < 123 (")

122 < 123 (")

111 < 123 (")

112 < 123 (at i = 5)

j at 5

i = 5 swap (A[j], pivot)

(112, 123)

112	110	117	122	111	123

step 4:
pivot = 112
partition (0-4)
i = 1 j = 4
110 < 112 (i moves right)
117 > 112 (stops at i = 2)

112	110	117	122	111
	i			j

j stops at 111

swap (117, 111)

112	110	111	122	117
		i		j

111 < 112 (i moves right)
122 > 112 (stops at i = 3)
i > j
swap (112, 111)

117 > 112 (moves left)
122 > 112 (moves left)
111 < 112 (stops j = 2)

111	110	112	122	117

112 is fixed

step 5:
partition (0-1)
i = 1 , j = 1
i = j
swap (111, 110)

110	111

sorted

110	111	112	122	117
0	1	2	3	4

step 6: partition (3-4)
i = 4 j = 4
pivot = 122
i = j so swap (122, 117)

117	122

110	111	112	117	122

sorted

step 7:- partition (7-10)

pivot = 141

151>141 (i stops at i=8)

i>j

swap (pivot, A[j])

(141,141)

sorted

141	151	149	147
7	8	9	10

141 fixed

141	151	149	147
7	8	9	10
	↑ i		↑ j

147 > 141 (j moves left)

149 > 141 (")

151 > 141 (")

141 ≥ 141 (stops at j=7)

step 8:- (8-10)

pivot = 151

i=9   j=10

149 < 151

147 < 151 (at i=10)

swap (151, 147)

147	149	151

sorted

147 < 151 (at j=10)

133	141	147	149	151

Final sorted array:-

110	111	112	117	122	123	133	141	147	149	151	157

Last element as pivot:-

157	110	147	122	111	149	151	141	123	112	117	133

pivot

pivot = last (high)

i = low

j = high-1

move i right while A[i] < pivot (while)

move j left A[j] > pivot (while)

If i < j swap A[i], A[j]

If i ≥ j swap A[i], pivot

Step 1:- partition (0-11)

pivot = 133

i = 0, j = 10

157 > 133 (i stops at i = 0)

117 < 133 (j stops j = 10)

swap (157 ⟷ 117)

117	110	147	122	111	149	151	141	123	112	157	133

j pivot

110 < 133 (move i right)

147 > 133 (stop at i = 2)

157 > 133 (moves left)

112 < 133 (stops at j = 9)

swap (147, 112)

117	110	112	122	111	149	151	141	123	147	157	133

i j pivot

112 < 133 (moves i right)

111 < 133 (")

149 > 133 (stops at i = 5)

~~147 < 1~~ 141 > 133 (moves left)

123 < 133 (stops at j = 8)

swap (149, 123)

117	110	112	122	111	123	151	140	149	147	157	133

i j

123 < 133 (moves right)

150 > 133 (i stops at i = 6)

i > j

149 > 133 (moves left)

141 > 133 (")

151 > 133 (")

123 < 133 (stops at j = 5)

swap (pivot, A[i])

swap (133, 151)

117	110	112	122	111	123	133	141	149	147	157	151

Pivot 133 Fixed

Step 2 :- Left (0-5)

Pivot = 123

i = 0 , j = 4

117	110	112	122	111	123
i				j	pivot

117 < 123 (moves right)

110 < 123 ,,

112 < 123 ,,

111 < 123 ,,

123 = 123 (stops at i = 5)

111 < 123 (stays at j = 4)

swap (123, 111)

117	110	112	122	111	123

pivot 123 fixed

Step 3 :- Left (0-4)

pivot = 111

i = 0 j = 3

117	110	112	122	111
i			j	pivot

122 > 111 (left)

112 > 111 (moves left)

110 < 111 (stops at j = 1)

117 > 111 (stops)

i = 0

swap (117, 110)

110	117	112	122	111
i	j			

117 > 111 (stops at i = 1)

110 < 111 (stops at j = 1)

i = j swap (117, 111)

110	111	112	122	117

pivot 111 is fixed

Step 4 :- (2-4)

pivot = 117

i = 2 j = 3

112 < 117 (i moves)

122 > 117 (stops)

j = 2 swap (117, 122)

110	111	112	117	122

Step 5: (7-11)

141	149	147	157	151
7	8	9	10	11

pivot = 151

i = 7 j = 10

141 < 151 (i moves right)

149 < 151 (")

147 < 151 (")

157 > 151 (i stops at i = 10)

157 > 151 (j moves left)

147 < 151 (stops)

j = 9

i > j

swap (157, 151)

141	149	147	151	157

pivot fixed

Step 6:- (7-9)

pivot = 147

141	149	147
7	8	9

i = 7 j = 8

141 < 147 (mover right)

149 > 147 (at j = 8)

swap (149, 147)

149 > 147 (mover left)

141 < 147 (at i = 7)

141	149	149

Final sorted array:-

110	111	112	117	122	123	133	141	147	149	151	157

(iii) Random element as pivot element

Method / logic

1) Choose random index
2) swap with First element
3) Use same method used in First element

sol:

157	110	147	122	111	149	151	141	123	112	117	133
0	1	2	3	4	5	6	7	8	9	10	11

Step 1:- Take 141 as pivot element
swap with first element

141	110	147	122	111	149	151	157	123	112	117	133
0	1	2	3	4	5	6	7	8	9	10	11

Pass 1:-

i stops at 147
j stops at 133 swap 147 133

141	110	133	122	111	149	151	157	123	112	117	147
0	1	2	3	4	5	6	7	8	9	10	11

Pass 2:-

i stops at 149
j stops at 133 swaps 149 117

141	110	133	122	111	149	151	157	123	112	149	147

Pass 3:-

i stops at 151
j stops at 112 swap 151 ⟷ 112

141	110	133	122	111	117	112	157	123	151	149	147

Pass 4:-

i stops at 157
j stops at 123 swap 157 ⟷ 123

123	110	133	122	111	117	112	141	157	151	149	147

pivot index = 7

left subarray [0 6]

step 2:- left subarray [0 6]

Pass 1:-

123	110	133	122	111	117	112

random pivot = 117

After swap:

117	110	133	122	111	123	112

i stops at 1
j stops at 6

swap 133 ⟷ 112

117	110	112	122	111	123	133

pass 2:-

i stops at 122
j stops at 111

swaps 122 ⟷ 111

117	110	112	111	122	123	133

Pass 3:-

i = 4
j = 3

i ≥ j → STOP

swap 111 ⟷ 117

111	110	112	117	122	123	133

left subarray [0 2]

Step 3:- subarray [0, 2] left of 117

take pivot = 110

swap with 111

110	111	112

Pass 1:- i = 1, j = 2 ⇒ i ≥ j ⇒ Stop

already sorted

Right of 117 · already sorted

Right of 141 ⇒

157	151	149	147

Step 4:- subarray [8, 11]

157	151	149	147

take pivot = 149

swap with 157

149	151	157	147

Pass 1 :-

i = 9
j = 11

swaps 147 ⟷ 151

149	147	157	151

Pass 2 :-

i = 10
j = 9

swap 149 ⟷ 147

147	149	157	151

Step 5 :- Right of 149 [10, 11]

157 . 151

pivot = 151

After . swap 151 157

sorted

Finally

110	111	112	117	122	123	133	141	149	144	151	157

At last Random pivot is most efficient than First or last

1. Avoid worst-case performance
2. Giver balanced positions
3. Improve average time complexity
4. Better practical performance.