

# **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

```
1 // SRIJA
2 // CH.SC.U4CSE24126
3
4 #include <stdio.h>
5 #include <stdlib.h>
6
7 void swap(int *a, int *b){
8     int temp = *a;
9     *a = *b;
10    *b = temp;
11 }
12
13 int partitionFirst(int a[], int low, int high){
14     int pivot = a[low];
15     int i = low + 1, j = high;
16
17     while(i <= j){
18         while(i <= high && a[i] <= pivot){
19             i++;
20         }
21         while(a[j] > pivot){
22             j--;
23         }
24         if(i < j){
25             swap(&a[i], &a[j]);
26         }
27     }
```

```
27     }
28     swap(&a[low], &a[j]);
29     return j;
30 }
31
32 int partitionLast(int a[], int low, int high){
33     int pivot = a[high];
34     int i = low - 1;
35
36     for(int j = low; j < high; j++){
37         if(a[j] <= pivot){
38             i++;
39             swap(&a[i], &a[j]);
40         }
41     }
42     swap(&a[i + 1], &a[high]);
43     return i + 1;
44 }
45
46 int partitionRandom(int a[], int low, int high){
47     int randomIndex = low + rand() % (high - low + 1);
48     printf("Randomly selected pivot: %d\n", a[randomIndex]);
49     swap(&a[randomIndex], &a[high]);
50     return partitionLast(a, low, high);
```

```
51 }
52
53 void quickSort(int a[], int low, int high, int choice){
54     if(low < high){
55         int p;
56         if(choice == 1){
57             p = partitionFirst(a, low, high);
58         }
59         else if(choice == 2){
60             p = partitionLast(a, low, high);
61         }
62         else{
63             p = partitionRandom(a, low, high);
64         }
65         quickSort(a, low, p - 1, choice);
66         quickSort(a, p + 1, high, choice);
67     }
68 }
69
70 void copyArray(int src[], int dest[], int n){
71     for(int i = 0; i < n; i++){
72         dest[i] = src[i];
73     }
74 }
75
```

```
76 void display(int a[], int n){
77     for(int i = 0; i < n; i++){
78         printf("%d ", a[i]);
79     }
80     printf("\n");
81 }
82
83 int main(){
84     int n, choice;
85
86     printf("Enter number of elements: ");
87     scanf("%d", &n);
88
89     int original[n], temp[n];
90
91     printf("Enter elements:\n");
92     for(int i = 0; i < n; i++){
93         scanf("%d", &original[i]);
94     }
95
96     while(1){
97         printf("\n----- QUICK SORT MENU ----- \n");
98         printf("1. First Element as Pivot\n");
99         printf("2. Last Element as Pivot\n");
100        printf("3. Random Element as Pivot\n");
101        printf("4. Exit\n");
```

```
102     printf("Enter your choice: ");
103     scanf("%d", &choice);
104
105     if(choice == 4){
106         printf("Exiting program...\n");
107         break;
108     }
109
110     if(choice < 1 || choice > 3){
111         printf("Invalid choice! Try again.\n");
112         continue;
113     }
114
115     copyArray(original, temp, n);
116
117     printf("\nOriginal array:\n");
118     display(temp, n);
119
120     quickSort(temp, 0, n - 1, choice);
121
122     printf("Sorted array:\n");
123     display(temp, n);
124 }
125
126     return 0;
127 }
```

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 =  $n \times \log n = O(n \log n)$ .

**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:

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

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

$i = \text{left} + 1$

$j = \text{right}$

Move  $i$  right side till  $A[i] > \text{pivot}$

Move  $j$  left side till  $A[j] < \text{pivot}$

If  $i < j \rightarrow \text{swap } A[i], A[j]$

If  $i \geq j \rightarrow \text{swap pivot with } A[j]$

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  $\leftrightarrow$  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

$i = 1$   $j = 10$

$133 > 110$  ( $i$  moves right)

$133 < 147$  ( $i$  stays at 147)

swap (147, 110)



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  $i = 2$ )

swap (117, 111)

112	110	111	122	117
-----	-----	-----	-----	-----

$\uparrow$   $\uparrow$   
 $i$   $j$

$111 < 112$  (i moves right)

$122 > 112$  (stops at  $i = 3$ )

$i > j$

swap (112, 111)

111	110	112	122	117
-----	-----	-----	-----	-----

112 is fixed

112	110	117	122	111
-----	-----	-----	-----	-----

$\uparrow$   $\uparrow$   
 $i$   $j$

j stops at 111

$117 > 112$  (moves left)

$122 > 112$  (moves left)

$111 < 112$  (stops  $j = 2$ )

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=8

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  $A[i] < \text{pivot}$  (while)

move j left  $A[j] > \text{pivot}$  (while)

IF  $i < j$  swap  $A[i], A[j]$

IF  $i \geq j$  swap  $A[i], \text{pivot}$

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, 122)

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 > 133~~ 147 > 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)

151 > 133 (stops at i = 6)

i = j

149 > 133 (moves left)

141 > 133 ( " )

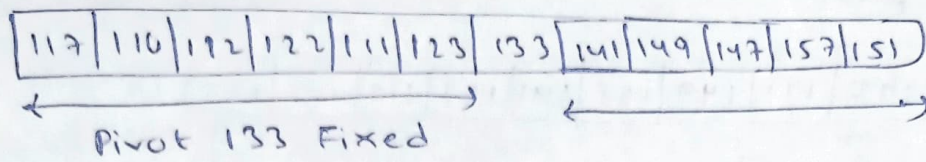
151 > 133 ( " )

123 < 133 ( stops at j = 5)

swap ( pivot, A[i] )

swap ( 133, 151 )





Step 2:- Left (0-5)

Pivot = 123

$i=0$ ,  $j=4$

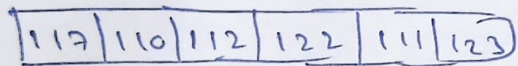
117 < 123 (moves right)

110 < 123 "

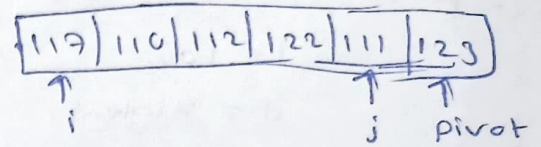
112 < 123 "

111 < 123 "

123 = 123 (stops at  $i=5$ )



Pivot 123 Fixed



111 < 123 (stays at  $j=4$ )  
swap(123, 111)

Step 3:- Left (0-4)

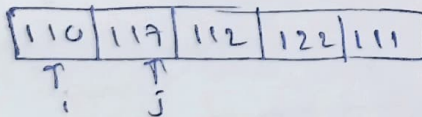
Pivot = 111

$i=0$   $j=3$

117 > 111 (stops)

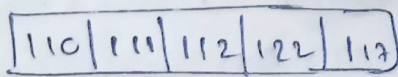
$i=0$

swap(117, 110)

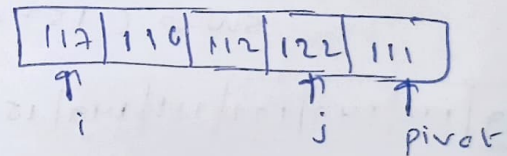


117 > 111 (stops at  $i=1$ )

$i=j$  swap(117, 111)



Pivot 111 is Fixed



122 > 111 (left)

112 > 111 (moves left)

110 < 111 (stops at  $j=1$ )

110 < 111 (stops at  $j=1$ )

Step 4:- (2-4)

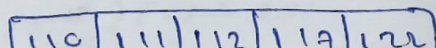
Pivot = 117

$i=2$   $j=3$

112 < 117 ( $i$  moves)

122 > 117 (stops)

$j=2$  swap(117, 122)



Step 5: (7-11)

141	149	147	151	151
7	8	9	10	11

pivot = 151

i = 7 j = 10

141 < 151 (i moves right)

149 < 151 ( " )

147 < 151 ( " )

151 > 151 (i stops at i = 10)

151 > 151 (j moves left)

147 < 151 (stops)

j = 9

i > j

swap (151, 147)

141	149	147	151	151
-----	-----	-----	-----	-----

pivot fixed

Step 6: (7-9)

pivot = 147

i = 7 j = 8

141	149	147
7	8	9

141 < 147 (moves right)

149 > 147 (at j = 8)

swap (149, 147)

149 > 147 (moves left)

141 < 147 (at i = 7)

141	149	147
-----	-----	-----

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	147	151	157	123	112	149	117
0	1	2	3	4	5	6	7	8	9	10	11

Pass 3:-

i stops at 151

j stops at 112

swap 151  $\leftrightarrow$  112

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

Pass 4:-

i stops at 157

j stops at 123

swap 157  $\leftrightarrow$  123

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

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

swap 133  $\longleftrightarrow$  112

j stops at 6

117	110	112	122	111	123	133
-----	-----	-----	-----	-----	-----	-----

pass 2:-

i stops at 122

swap 122  $\longleftrightarrow$  111

j stops at 111

117	110	112	111	122	123	133
-----	-----	-----	-----	-----	-----	-----

Pass 3:-

i = 4

$i \geq j \rightarrow \text{STOP}$

j = 3

swap 111  $\longleftrightarrow$  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:-  $i = 1, j = 2 \Rightarrow i \geq j \Rightarrow \text{Stop}$

already sorted

Right of 117: already sorted

Right of 141  $\Rightarrow$ 

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  $\longleftrightarrow$  151

149	147	157	151
-----	-----	-----	-----

Pass 2:-

i = 10

j = 9

swap 149  $\longleftrightarrow$  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	147	151	157
-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----

At last Random pivot is most efficient than First or last

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