

Assignment -05

Name : B. Lakesh
Reg No : 192365037
Dept : CSE (Cyber Security)
Course code : CSA0389
Course Name : Data Structures
Faculty Name : Dr. Ashok Kumar
Assignment No : 5.
Date of Submission : 21-Aug-2024
No : of pages : 06.

i. Write the algorithm for insertion sort and sort the following sequence: 3, 1, 4, 1, 5, 9, 2, 6, 5.

ii. Explain the procedure for merged sort and perform the merge sort for the following inputs. Also, show the result for each step of iteration 64, 8, 216, 512, 27, 729, 0, 1, 343, 125.

Algorithm:

1. Start with the second element (index 1) in the array, assuming the first element (index 0) is sorted.
2. Compare the current element with the elements in the sorted part.
3. Shift all the elements of the sorted part that are greater than the current elements to the right.
4. Insert the current element into its current position.
5. Repeat steps 2 to 4 for each element in the unsorted part of the array.
6. Finish when the all element are sorted.

Insertion sort on Sequence: [3, 1, 4, 1, 5, 9, 2, 6, 5]

3	1	4	1	5	9	2	6	5
---	---	---	---	---	---	---	---	---

Compare $3 \& 1$, $3 > 1$
Swap 3, 1

1	3	4	1	5	9	2	6	5
---	---	---	---	---	---	---	---	---

Compare $4 \& 1$, $4 > 1$
Swap 4, 1

1	3	1	4	5	9	2	6	5
---	---	---	---	---	---	---	---	---

Compare $3 \& 1$, $3 > 1$
swap 3, 1

1	1	3	4	5	9	2	6	5
---	---	---	---	---	---	---	---	---

compare $9 \& 2$, $9 > 2$
Swap 9, 2

1	1	3	4	5	2	9	6	5
---	---	---	---	---	---	---	---	---

compare $5 \& 2$, $5 > 2$
swap 5, 2

1	1	3	4	2	5	9	6	5
---	---	---	---	---	---	---	---	---

compare $4 \& 2$, $4 > 2$
swap 4, 2

1	1	3	2	4	5	9	6	5
---	---	---	---	---	---	---	---	---

Compare $3 \& 2$, $3 > 2$
swap 3, 2

1	1	2	3	4	5	9	6	5
---	---	---	---	---	---	---	---	---

compare $9 \& 6$, $9 > 6$
swap 9, 6.

1	1	2	3	4	5	6	9	5
---	---	---	---	---	---	---	---	---

Compare $9 \& 5$, $9 > 5$
swap 9, 5

1	1	2	3	4	5	6	5	9
---	---	---	---	---	---	---	---	---

compare $6 \& 5$, $6 > 5$
swap 6, 5

1	1	2	3	4	5	5	6	9
---	---	---	---	---	---	---	---	---

→ Sorted

Merge Sort procedure:

- * Split the list into halves until each sublist has one element.
- * combine the sublists to produce new sorted sublists until there is one sorted list.

Merge Sort with 64, 8, 216, 512, 27, 729, 0, 1, 343, 125

1. Initial split:

- [64, 8, 216, 512, 27] and [729, 0, 1, 343, 125]

2. Further split:

- [64, 8] and [216, 512, 27]
- [729, 0] and [1, 343, 125]

3. Further split:

- [64] and [8]
- [216] and [512, 27]
- [729] and [0]
- [1] and [343, 125]

4. Merge:

- Merge [64] and [8] \rightarrow [8, 64]
- Merge [512, 27] \rightarrow [27, 512]
- Merge [216] and [27, 512] \rightarrow [27, 216, 512]
- Merge [0] and [729] \rightarrow [0, 729]
- Merge [343, 125] \rightarrow [125, 343]
- Merge [1] and [343, 125] \rightarrow [1, 125, 343]

5. Final Merge:

• Merge $[8, 64]$ and $[27, 216, 512]$
 $\rightarrow [8, 27, 64, 216, 512]$

• Merge $[0, 729]$ and $[1, 125, 343]$
 $\rightarrow [0, 1, 125, 343, 729]$

• Merge $[8, 27, 64, 216, 512]$ and $[0, 1, 125, 343, 729]$
 $\rightarrow [0, 1, 8, 27, 64, 125, 216, 343, 512, 729]$

Sorted list: $0, 1, 8, 27, 64, 125, 216, 343, 512, 729$.

Draw the concept map of partitioning in quick sort, try to write an algorithm for it, which is as follows, & develop a program considering the steps.

Step 1: choose the highest index value as pivot

Step 2: Take two variables to point left and right of the list excluding pivot.

Step 3: left points to the low index.
using elements your own

Algorithm:

- * Select the element at the highest index as the pivot
- * Set 'left' to the low index and 'right' to the high index - 1
- * Move 'left' rightwards and 'right' leftwards until 'left' is greater than or equal to 'right', swapping elements as the needed.
- * Swap the pivot with the elements at the 'left' pointer position
- * Return the index of the pivot element

Program:

```
#include <stdio.h>

int main() {
    int arr[] = {64, 8, 216, 512, 27, 729, 0, 1, 343, 125};
    int n = sizeof(arr) / sizeof(arr[0]);
    int low = 0, high = n - 1;
    while (low < high) {
        int pivot = arr[high];
        int left = low;
        int right = high - 1;
        while (left <= right) {
            while (left <= right && arr[left] < pivot) {
                left++;
            }

```

```
while (right >= low && arr[right] > pivot) &  
    right --;
```

```
}  
if (left < right) {  
    int temp = arr[left];  
    arr[left] = arr[right];  
    arr[right] = temp;  
    left++;  
    right --;
```

```
}
```

```
}  
int temp = arr[left];  
arr[left] = arr[high];  
arr[high] = temp;  
high = left - 1;
```

```
if (high < low) {  
    low = left + 1;  
    high = n - 1;
```

```
}
```

```
}
```

```
printf("sorted array:");
```

```
for (int i = 0; i < n; i++)
```

```
    printf("%d", arr[i]);
```

```
}
```

```
printf("\n");
```

```
return 0;
```

```
}
```

Output :

Sorted array: 0, 1, 8, 27, 64, 125, 216, 343, 512, 729.